

Oracle® Communication and Mobility Server

Administrator's Guide

10g Release 3 (10.1.3)

E12656-01

July 2008

E12656-01

Copyright © 2006, 2008, Oracle. All Rights Reserved.

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

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

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

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

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

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Copyright © 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

License to copy and use this software is granted provided that it is identified as the 'RSA Data Security, Inc. MD5 Message-Digest Algorithm' in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as 'derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm' in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided 'as is' without express or implied warranty of any kind.

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

Contents

Preface	xi
Intended Audience.....	xi
Documentation Accessibility	xi
Related Documents	xi
Conventions	xii
1 An Overview of Oracle Communication and Mobility Server	
New in this Release	1-1
TLS Support	1-1
Scalable Presence Deployments with User Dispatcher	1-2
Presence Dispatching	1-2
Web Services Improvements	1-2
Introduction to OCMS	1-2
OCMS Three Layer Model	1-3
Proxy Layer	1-3
Application Layer.....	1-4
Data Layer	1-4
OCMS System Components	1-4
SIP Servlets and SIP Servlet Applications	1-5
Differences between HTTP and SIP Servlets	1-6
Typical SIP Servlet Applications	1-6
SIP Servlet Container	1-7
How the OCMS SIP Servlet Container Works	1-7
Edge Proxy Server.....	1-8
Proxy Registrar	1-9
Location Lookup Service.....	1-10
ENUM Lookup Service	1-10
Presence Server.....	1-11
How the Presence Server Works	1-12
Application Router.....	1-13
Modes of Operation.....	1-13
Using the Application Router in Standard Mode: an Example	1-14
Using the Application Router in Incremental Mode: an Example	1-14
Subscriber Data Services	1-15
Authentication and Authorization Data.....	1-15

User Data.....	1-15
Location Lookup Data.....	1-15
Logging.....	1-15
Session Replication.....	1-15

2 Deployment Topologies

About Deployment Topologies	2-1
Topology Components	2-2
Third-Party Load Balancer.....	2-2
Edge Proxy Nodes.....	2-2
SIP Application Servers.....	2-2
Aggregation Proxy	2-3
Proxy Registrar	2-3
User Dispatcher	2-3
Presence Dispatching	2-3
Supported OCMS Topologies.....	2-3
Deploying OCMS as a Highly Available SIP Network.....	2-3
Deploying OCMS as a Presence Server	2-5
Deploying a Scalable Presence Deployment	2-6
Presence Cluster	2-6
XDM Cluster	2-7
Presence Node	2-8
XDM Node	2-8
Complete Presence and XDM Cluster.....	2-8
Deploying OCMS as an Instant Messaging Service	2-9
Deploying an OCMS Testing Environment.....	2-11
Configuration Recommendations	2-13

3 Configuring the SIP Server MBeans

Overview of SIP Server Management.....	3-1
Starting, Stopping and Restarting the OCMS SIP Server.....	3-2
Starting an Application and Stopping a SIP Servlet Application	3-2
Managing OCMS MBeans.....	3-3
Accessing MBeans.....	3-4
Accessing SIP Servlet Container MBeans.....	3-4
Accessing the MBeans for a Selected SIP Application.....	3-5
Configuring the SIP Servlet Container MBeans	3-5
SIP Servlet Container.....	3-6
Setting the Keystore.....	3-11
Enabling TLS.....	3-11
SIP Servlet Container Logging.....	3-12
STUN Service.....	3-12
Configuring SIP Applications	3-13
Subscriber Data Services	3-13
Proxy Registrar	3-16
Application Router	3-17
Setting and Viewing the SIP Port.....	3-19

4 Configuring Security and Login Modules

Overview of Security	4-1
The OCMS JAAS-Compliant Login Modules	4-1
Application Type and Authentication Mode.....	4-2
Configuring Subscriber Data Services	4-3
CommandService	4-4
Configuring Applications to Use Login Modules	4-6
Configuring Login Modules though system-jazn-data.xml and orion-application.xml	4-6
Configuring Login Modules in system-jazn-data.xml	4-6
Declaring the OCMS Login Module in orion-application.xml.....	4-7
Declaring the RADIUS Login Module in orion-application.xml	4-7
Security in SIP Servlets	4-8
Authentication Using the P-Asserted Identity Header	4-9
Authentication of Web Service Calls and XCAP Traffic	4-10
Default Role for All Users	4-10
Configuring Oracle Internet Directory as the User Repository	4-10
Overview of Configuration for OID Support	4-10
Prerequisites for OID Support	4-10
Configuring the OID LDAP Backend.....	4-11
Mapping JAAS Usernames to LDAP User Entries	4-11
Mapping JAAS Realms to LDAP Subscribers.....	4-11
Mapping JAAS Roles to LDAP Groups.....	4-11
Installing OCMS Components into the OID LDAP Tree	4-12
Associating an OCMS Instance with OID	4-12
Installing the OCMS Static Verifiers	4-13
Repackaging Subscriber Data Services	4-13
Configuring User Service and Security Service.....	4-13
Provisioning OCMS Users to OID	4-18
Adding Users to LDAP Groups.....	4-18

5 Configuring High Availability

About Configuring High Availability	5-1
Setting Up a Highly Available Cluster of OCMS Nodes	5-3
Associating Nodes with OPMN.....	5-3
Associating Nodes with OPMN Using the Dynamic Discovery Method	5-3
Associating Nodes with OPMN Using the Discovery Server Method	5-4
Starting the Cluster	5-5
Verifying the Status of the Cluster.....	5-5
Stopping the Cluster	5-5
Configuring the OCMS SIP Containers for High Availability	5-5
Configuring the Edge Proxy Nodes for High Availability	5-6
The NAT Traversal Option Enabled for the Edge Proxy	5-7
Disabling NAT Traversal Enabled by the Edge Proxy	5-7
Configuring Highly Available SIP Servlet Applications	5-8
Enabling High Availability in SIP Servlet Applications.....	5-8
Configuring Application Session Data Replication	5-10

Configuring High Availability for a Deployed SIP Servlet Application	5-11
Disabling High Availability at the Application Level	5-12
Upgrading SIP Servlet Applications in OCMS	5-12
Configuring an Overload Policy	5-13
Overview of Overload Policy Architecture	5-13
Collectors	5-13
Deactivating the Overload Protection for System Tuning	5-19

6 Viewing Statistics and Metrics

Viewing Statistics and Metrics	6-1
SIP Servlet Container Monitor	6-1
Viewing System Status	6-1
Viewing Transactions	6-2
Using the Current, Peak, and Total Usage Statistics to Tune the System	6-4
Application Counters	6-4
Memory Monitor	6-5
Starting and Stopping the Memory Monitor	6-5
SIP Cluster	6-5

7 Configuring Presence and Presence Web Services

Overview of Presence	7-1
Configuring Presence	7-2
Configuring XDMS	7-3
Bus	7-3
PackageManager	7-4
Presence	7-5
PresenceEventPackage	7-6
PresenceWInfoEventPackage	7-7
UA-ProfileEventPackage	7-8
UserAgentFactoryService	7-8
Command Service (XDMS Provisioning)	7-9
XCapConfig	7-9
Configuring Presence Web Services	7-10
PresenceWebServiceDeployer	7-11
PresenceSupplierWebService	7-11
PresenceConsumerWebService	7-12
Aggregation Proxy	7-12
Configuring the Aggregation Proxy to Work with Realms	7-13
Securing the XDMS with the Aggregation Proxy	7-14
Configuring Scalable Presence Deployments with the User Dispatcher	7-14
Failover	7-14
Presentity Migration	7-15
Standby Server Pool	7-15
Failure Types	7-16
Failover Actions	7-16
Overload Policy	7-17
Synchronization of Failover Events	7-17

Expanding the Cluster.....	7-17
Updating the Node Set.....	7-18
Migrating Presentities	7-18
Failover Use Cases	7-18
One Presence Server Overloaded for 60 Seconds.....	7-18
One Presence Server Overloaded Multiple Times for Five Seconds.....	7-19
Overload Policy Triggered by an OCMS Software Failure.....	7-19
A Presence Server Hardware Failure.....	7-19
Expanding the Cluster with One Presence Node.....	7-19
Removing a Node from the Cluster	7-20
OPMN Restart After a Presence Server Crash.....	7-20
503 Responses from an Application	7-20

8 OCMS Parlay X Web Services Architecture

Architecture of Web Service Client Applications	8-1
Web Service Security	8-1
Web Service Security on Notification.....	8-2
Installing the Web Services	8-3

9 OCMS Parlay X Presence Web Services

Introduction.....	9-1
Presence Web Services Interface Descriptions	9-1
Using the Presence Web Services Interfaces	9-3
Interface: PresenceConsumer, Operation: subscribePresence	9-3
Code Example.....	9-3
Interface: PresenceConsumer, Operation: getUserPresence	9-3
Code Example.....	9-4
Interface: PresenceConsumer, Operation: startPresenceNotification	9-4
Code Example.....	9-4
Interface: PresenceConsumer, Operation: endPresenceNotification.....	9-5
Code Example.....	9-5
Interface PresenceSupplier, Operation: publish and Oracle Specific Remove Presence	9-5
Code Example.....	9-5
Interface: PresenceSupplier, Operation: getOpenSubscriptions	9-6
Code Example.....	9-6
Interface: PresenceSupplier, Operation: updateSubscriptionAuthorization.....	9-6
Code Example.....	9-6
Interface: PresenceSupplier, Operation: getMyWatchers	9-7
Code Example.....	9-7
Interface: PresenceSupplier, Operation: getSubscribedAttributes	9-7
Code Example.....	9-7
Interface: PresenceSupplier, Operation: blockSubscription.....	9-7
Code Example.....	9-7
OCMS Parlay X Presence Custom Error Codes	9-7

10 OCMS Parlay X Multimedia Messaging Web Services

Introduction.....	10-1
Multimedia Messaging Web Services Interface Descriptions.....	10-1
Using the Multimedia Messaging Web Services Interfaces	10-3
Interface: SendMessage, Operation: sendMessage.....	10-3
Interface: sendMessage, Operation: getMessageDeliveryStatus.....	10-3
Interface: ReceiveMessage, Operation: getReceivedMessages.....	10-3
Interface: ReceiveMessage, Operation: getMessageURIs.....	10-3
Interface ReceiveMessage, Operation: getMessage.....	10-3
Interface: MessageNotificationManager, Operation: startMessageNotification.....	10-4
Interface: MessageNotificationManager, Operation: stopMessageNotification.....	10-4

11 Provisioning Users with Sash

Overview of Sash	11-1
Launching Sash.....	11-1
Launching Sash from the Command Line.....	11-1
Connecting Sash to an External OCMS Instance	11-2
Connecting to an External Instance of OC4J.....	11-2
Connecting Sash to an External Oracle Application Server Instance.....	11-2
Using Sash	11-2
Viewing Available Commands	11-2
Viewing Subcommands	11-5
Creating a User.....	11-7
Creating a User from the Sash Command-Line Prompt	11-7
Creating a User with the Command Service MBean	11-8
Creating a User with the identity add Command.....	11-9
Deleting a User Account with the identity delete Command.....	11-10
Provisioning the XDMS Using Sash.....	11-10
Provisioning XDMS User Accounts Using the CommandService MBean	11-10
Provisioning XDMS User Accounts from the Sash Prompt.....	11-10
Using xcap Commands	11-11
Provisioning XDMS User Accounts	11-11
Adding XDMS Users	11-11
Removing an XDMS User	11-11
Searching for Application Usage for an XDMS User.....	11-11
Listing XDMS Users.....	11-12
Provisioning Application Usage.....	11-12
Listing All Application Usages	11-12
Scripting with Sash.....	11-12
Error Logging in Sash	11-13

12 Configuring the Logging System

Overview of Oracle Diagnostic Logging in OCMS.....	12-1
Logging Components	12-1
Filtering of Logging Information by Single Class Files	12-1
Log Files.....	12-2

Logger Interfaces	12-2
Logging Levels	12-2
Setting the Log Levels for Components	12-3
13 Deploying Applications	
Overview of SIP Servlet Applications	13-1
Deploying SIP Applications	13-2
Deploying, Undeploying, and Redeploying SIP Applications Using Oracle Application Server Control	13-3
Deploying, Undeploying, and Redeploying SIP Servlet Applications with Application Server Control	13-4
Deploying an Application using the Deployment Wizard.....	13-4
Undeploying an Application Using the Deployment Wizard	13-7
Redeploying an Application Using the Deployment Wizard	13-7
Deploying, Undeploying, and Redeploying an Application Using the admin_client.jar Utility	13-8
Deploying an Application Using admin_client.jar	13-8
Undeploying an Application Using admin_client.jar	13-8
Redeploying an Application Using admin_client.jar	13-8
Deploying the SIP Application Using the admin_client.jar Command-Line Utility	13-8
A Supported Protocols, RFCs, and Standards	
SIP Servlet Container	A-1
RFCs	A-1
Drafts.....	A-2
Specification Requests	A-3
Presence Server	A-3
RFCs	A-3
Drafts Referenced in the Composition Policies	A-4
XDMS Server.....	A-4
Authorization and Privacy Filtering	A-4
Presence Data Modeling and Processing.....	A-5
OMA Extensions.....	A-5
Hard State via XCAP	A-5
B Third-Party Licensing	
Third-Party Licenses	B-1
Index	

Preface

This guide describes how to configure and manage the Oracle Communication and Mobility Server.

Intended Audience

This manual is intended for Oracle Communication and Mobility Server administrators.

Documentation Accessibility

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

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

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

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

Related Documents

For more information, see the following manuals:

- *Oracle Communication and Mobility Server Installation Guide*
- *Oracle Containers for J2EE Configuration and Administration Guide*
- *Oracle Containers for J2EE Deployment Guide*

- *Oracle Containers for J2EE Security Guide*
- *Oracle Internet Directory Administrator's Guide*
- *Oracle Internet Guide to Delegated Administration*
- Oracle Communication and Mobility Server resources on Oracle Technology Network (http://www.oracle.com/technology/products/ocms/otn_front.htm). This is the location for Oracle Communication and Mobility Server guides, release notes, white papers, and updates.

Support—Visit: <http://www.oracle.com/support>

Conventions

The following conventions are also used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
...	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
<>	Angle brackets enclose user-supplied names.
[]	Brackets enclose optional clauses from which you can choose one or none.

An Overview of Oracle Communication and Mobility Server

This chapter provides an introduction to the Oracle Communication and Mobility Server (OCMS) in the following sections:

- ["New in this Release"](#)
- ["Introduction to OCMS"](#)
- ["OCMS Three Layer Model"](#)
- ["OCMS System Components"](#)
- ["OCMS System Components"](#)

New in this Release

This release of Oracle Communication and Mobility Server 10.1.3.4 includes the following enhancements and new features:

- ["TLS Support"](#)
- ["Scalable Presence Deployments with User Dispatcher"](#)
- ["Web Services Improvements"](#)

Oracle Database is now the main, supported database for OCMS subscriber persistence. Oracle TimesTen is no longer included with OCMS 10.1.3.4.

To read about new and improved features, see the following link:

http://www.oracle.com/technology/products/ocms/otn_front.htm.

See also ["Supported Protocols, RFCs, and Standards"](#).

TLS Support

OCMS 10.1.3.4 supports TLS as a network connectivity option.

The OCMS SIP Servlet Container support TLS in two modes, one in which the SIP container acts as a TLS server and another where you configure the SIP Servlet Container to provide mutual TLS, where the container not only provides its server certificate but also requires a client certificate.

Scalable Presence Deployments with User Dispatcher

The User Dispatcher enables the Presence and XDMS applications to scale. The User Dispatcher is a proxy that dispatches SIP, HTTP, and XCAP (over HTTP) requests to their appropriate destinations on a consistent basis.

Presence Dispatching

Because the Presence application maintains the states for all users in the deployment, the User Dispatcher enables scaling (distribution) of the Presence application. The User Dispatcher supports request dispatching to the following Presence sub-applications, which use the SIP and XCAP (over HTTP) protocols:

- Presence server
- Presence XDMS
- Shared XDMS

Web Services Improvements

The following improvements have been made to the Web Services APIs:

- ParlayX 2.1 Presence Web Services API fully supported including asynchronous Web services. For more information see, [Chapter 9, "OCMS Parlay X Presence Web Services"](#).
- A new API created for the SIP-based ParlayX 2.1 Messaging Web Services API. For more information see [Chapter 10, "OCMS Parlay X Multimedia Messaging Web Services"](#).
- A new API for Contact Management API. This is a JAVA utility API that helps managing users contact list and presence rules. For more detail, refer to the javadoc that comes with the installation of OCMS. The javadoc is located at `$ORACLE_HOME/sdp/api-docs/sdpcontactmanagement-10.1.3.4.0-javadoc.zip`

Introduction to OCMS

Oracle Communication and Mobility Server (OCMS) is a carrier-grade SIP application environment for the deployment, and management of SIP applications. Built on a standard Java2 Enterprise Edition (J2EE) platform, OCMS is a flexible, scalable environment enabling easy integration of SIP applications and services.

Among the applications that may be deployed on SIP platforms:

- Voice and video telephony, including call management services such as call forwarding and call barring
- Publication of and subscription to user presence information, such as online/offline status, notifications, permission to access a user's status, and so on
- Instant messaging
- Push to Talk applications, including Push-to-Talk over Cellular (PoC)

OCMS provides standard SIP applications, including a Presence Server, a combination Proxy and Registrar server, and a SIP Server. An integral part of any SIP platform, these applications are automatically installed to the OCMS platform, reducing development resources and time to go live.

OCMS provides a standards-based Presence Server which provides SIMPLE compliant Presence and event notification features. The OCMS Presence Server is robust enough

to support carriers with a heavy load of subscribers, while still being a viable solution for ISVs, system integrators, and enterprises requiring an integration platform and an enterprise Presence Server.

OCMS provides the following functionality:

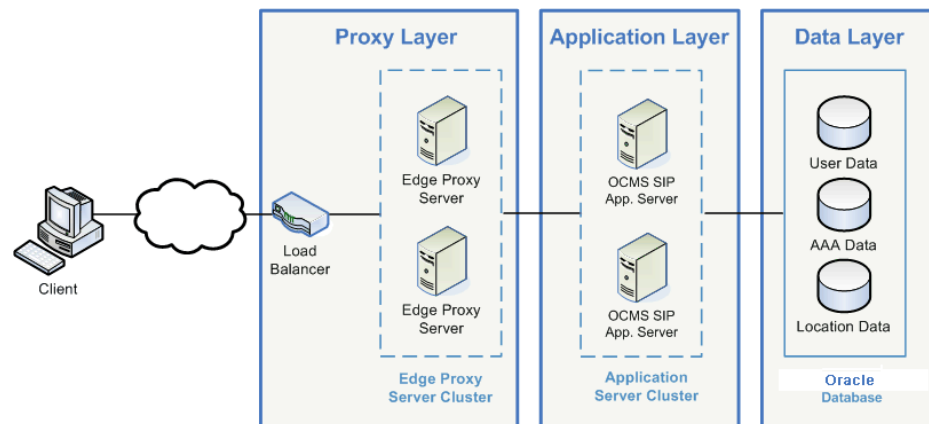
- **Deployment**—SIP Servlet applications are easily packaged and deployed on Oracle Application Server acting as a SIP servlet container.
- **Configuration and Management**—The Oracle Application Server administrator console enables managing SIP applications as well as the SIP server and its components.
- **Authentication and Security**—User, role, and policy data can be stored on an external RADIUS database, or in Oracle Identity Management, enabling OCMS to authenticate connecting users against this data. OCMS secures SIP traffic through digest-based authentication as specified in RFC 3261. In addition, trusted hosts can be configured using p-asserted identity headers.
- **Logging and Monitoring**—OCMS enables monitoring of the deployed SIP applications through comprehensive logging functions as well as metrics exposed as JMX Mbeans.

OCMS Three Layer Model

Oracle Communication and Mobility Server architecture is composed of three layers:

- [Proxy Layer](#)
- [Application Layer](#)
- [Data Layer](#)

Figure 1–1 OCMS Three Layer Model



Proxy Layer

The Proxy layer includes an IP load balancer and the OCMS Edge Proxy Server. The IP load balancer provides a unique public address to which SIP requests are sent. The IP load balancer distributes SIP requests either to the OCMS Edge Proxy Server or directly to an OCMS SIP Server.

The IP load balancer is not SIP-aware - it is unaware of the content of the traffic it forwards, such as the sender and recipient. The IP load balancer distributes requests to OCMS nodes based on the availability of individual servers. This is essential in a

clustered environment, particularly in the event of a node failure. If a node fails, the load balancer redistributes traffic to the remaining nodes until the failure is corrected.

The OCMS Edge Proxy is a SIP load balancer, proxying SIP requests to a particular OCMS SIP Server. The Edge Proxy forms logical pathways between sessions and SIP servers, such that SIP traffic sent from a particular session is always handled by the same server. As the number of SIP clients increases, additional Edge Proxy servers can be added, providing highly scalable and efficient handling of SIP clients.

Application Layer

The Application layer is typically composed of a cluster of OCMS SIP Server nodes. The Application layer provides SIP clients with low response time and high throughput when handling SIP requests. As the Application layer handles a greater number of transactions, it can be scaled up by adding additional OCMS SIP Server nodes. To achieve high availability of session data, replication of session data can be enabled so that sessions survive a failover.

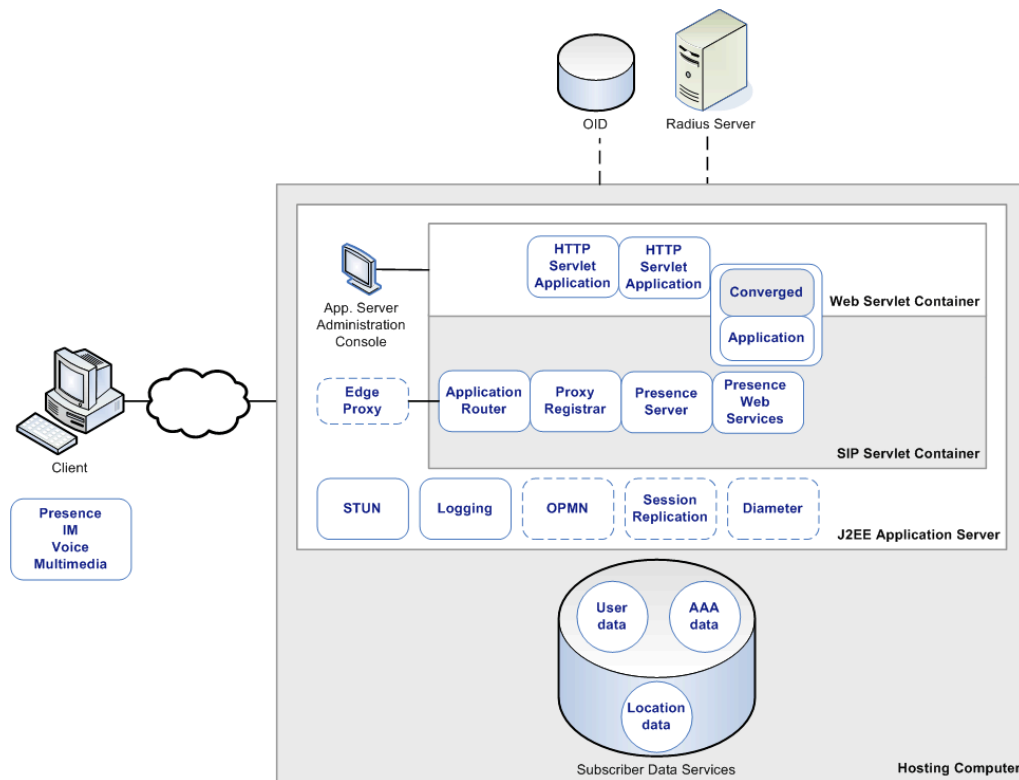
Data Layer

The Data layer is typically composed of a highly available, high performance database for the storage and retrieval of user, authentication, authorization, and location data. This data is replicated among all nodes. Similarly, SIP Servlet session data is replicated among nodes. In the event of a node failure, another node takes over the session data of the failed node.

OCMS System Components

[Figure 1–2](#) illustrates the logical system components of OCMS:

Figure 1–2 Oracle Communication and Mobility Server



The OCMS components are as follows:

- [SIP Servlets and SIP Servlet Applications](#)
- [SIP Servlet Container](#)
- [Edge Proxy Server](#)
- [Proxy Registrar](#)
- [Presence Server](#)
- [User Dispatcher](#) (described in [Scalable Presence Deployments with User Dispatcher](#))
- [Application Router](#)
- [Subscriber Data Services](#)
- [Logging](#)
- [Session Replication](#)

SIP Servlets and SIP Servlet Applications

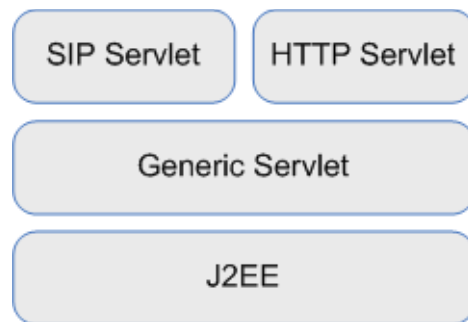
Servlets are dynamic applications that run on a web server using the J2EE platform. Like the HTTP Servlet API, the SIP Servlet API (JSR116) extends the functionality of the Java Servlet to receive SIP requests and generate SIP responses, regardless of the underlying network. A SIP application consists of one or more SIP Servlets which, along with a deployment descriptor, are packaged and deployed on a J2EE SIP Servlet Container.

Differences between HTTP and SIP Servlets

Although they are similar, the SIP Servlet differs from the HTTP Servlet as follows:

- SIP applications include intelligent request routing and the ability to proxy requests as required, even to multiple destinations.
- SIP is a peer-to-peer protocol, with endpoints that can typically initiate and respond to SIP requests.
- SIP Servlet applications may be registered so as to be invoked in response to particular events.
- Unlike the HTTP Servlet, the SIP Servlet is asynchronous. This means that when receiving a SIP request, a SIP Servlet application can initiate another action, return control to the SIP Servlet container, and respond to the request at a later time.
- A SIP Servlet application is often composed of more than one SIP Servlet.
- As SIP and HTTP servlets are based on the same generic Servlet specification, both types of servlets can be easily converged into one application. An application composed of both SIP and HTTP servlets can therefore handle both SIP and HTTP traffic.

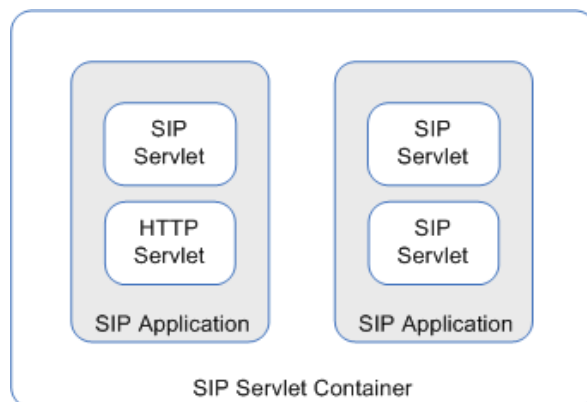
Figure 1–3 SIP Servlet Model versus HTTP Servlet Model



Typical SIP Servlet Applications

Typical SIP-based applications include:

- Telephony over IP, with the following features:
 - Speed dial
 - Wake-up call service
 - Call forwarding service
 - Click-to-call
 - Emergency call service
- Video calls
- Push to talk
- Instant messaging
- Presence information service
- Network gaming

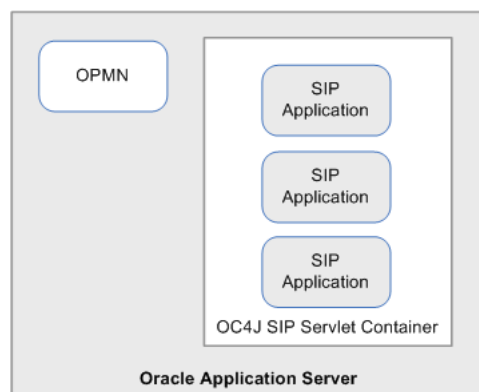
Figure 1–4 SIP Servlet Applications

SIP Servlet Container

A SIP Servlet Container extends the J2EE Application Server, providing a runtime environment for SIP applications, including services such as security, concurrency, life cycle management, transaction, deployment, and other services. A JSR116-compliant SIP Servlet Container provides network services for sending and receiving SIP requests and responses using a combination of transport protocols, IP addresses, and port numbers to listen for incoming SIP traffic.

The OCMS SIP Servlet Container can be installed on an existing instance of Oracle Application Server, running in OC4J. Alternatively, the OCMS SIP Servlet Container can run on its own stand-alone instance of OC4J.

The typical OCMS SIP Servlet Container is composed of an Oracle Application Server instance with OC4J as its J2EE container, and Oracle Process Manager and Notification Server (OPMN) to monitor the server. OCMS currently supports high availability deployments in this configuration only.

Figure 1–5 The SIP Servlet Container on Oracle Application Server

How the OCMS SIP Servlet Container Works

The SIP Servlet Container is configured upon server startup. Once a SIP Servlet application is deployed to the SIP Servlet Container, its deployment descriptor is used to configure its servlets and create a servlet context. The SIP application's listeners and servlets are instantiated, and the servlets are initialized with the servlet configurations.

When the SIP Servlet Container receives an initial incoming request, it processes a set of rules in order to send the request to the correct SIP Servlet. Once the request arrives

at the SIP Servlet, the Servlet must either proxy the request to a new destination, dispatch it to another Servlet, or send a response.

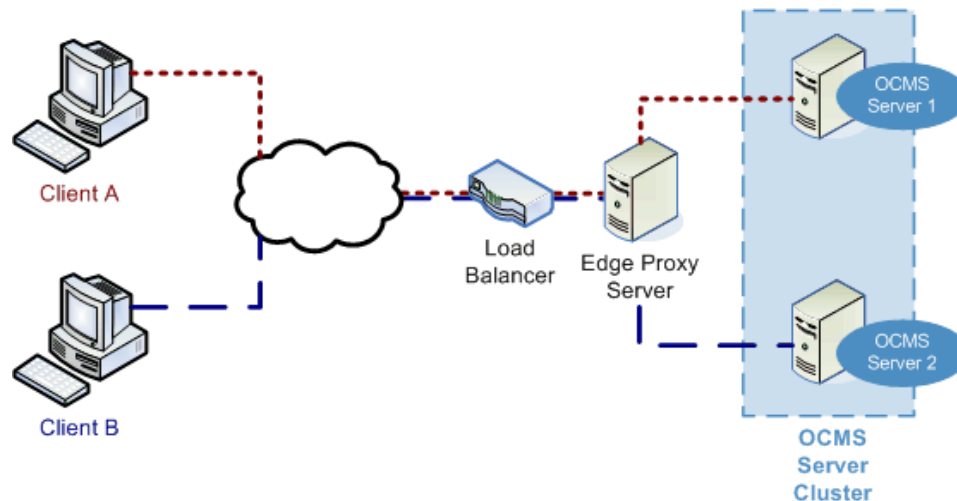
Edge Proxy Server

The Edge Proxy server provides the following functionality:

- Acts as a load balancer for initial incoming SIP requests
- Provides SIP Server affinity and failover for subsequent SIP requests in a session
- Manages the health of the OCMS application servers in a cluster by dynamically constructing a routing table of OCMS application servers

The Edge Proxy distributes incoming SIP traffic among OCMS SIP application servers when used between a SIP-unaware load balancer and an OCMS cluster. A standalone Java application running on its own server, the Edge Proxy establishes an affinity between the client and SIP Application Server for the duration of the session. This means that the same OCMS SIP Application Server always handles traffic from a particular client for the duration of the session, creating a path between the client and server.

Figure 1–6 Edge Proxy Functionality



Multiple Edge Proxy Servers can be deployed in a highly available environment. In this scenario, a load balancer distributes incoming traffic among the Edge Proxy Servers. Together, the Edge Proxy Servers can handle a greater load of subscribers connecting to the cluster of OCMS SIP Application Servers.

Figure 1–7 Multiple Edge Proxy Servers

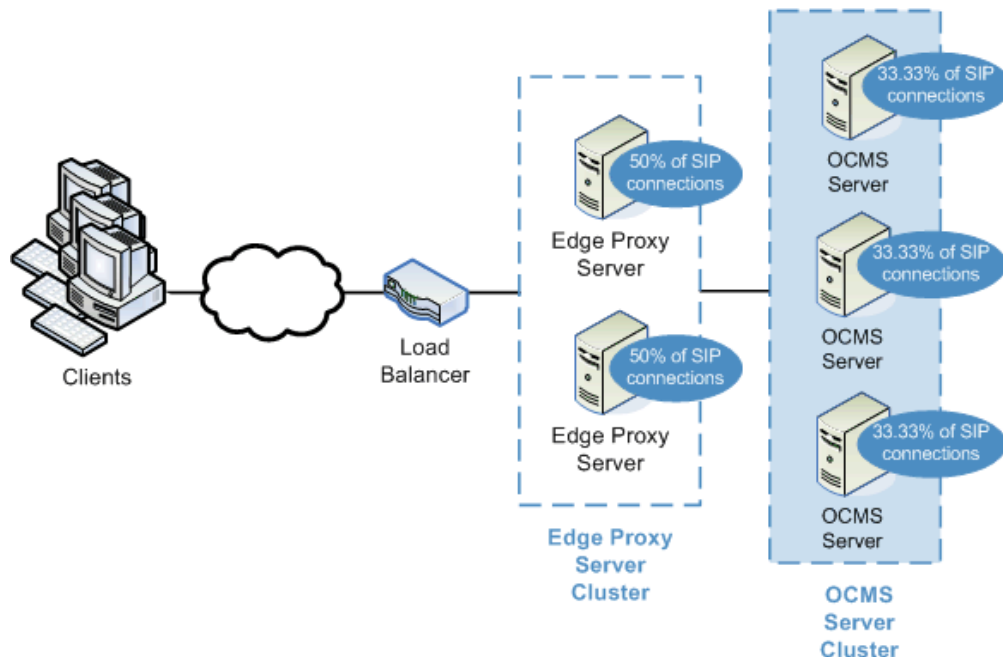
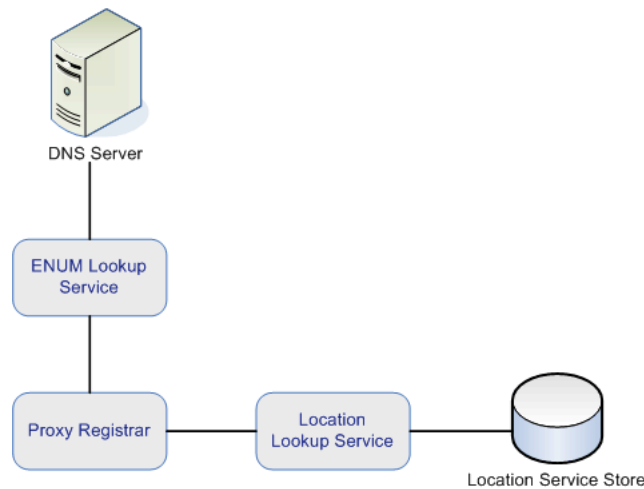


Figure 1–7 illustrates how the use of two Edge Proxy Servers reduces the load of SIP connections in a clustered OCMS environment. A third-party load balancer provides a single virtual IP address to which clients may address requests, and distributes SIP requests to the Edge Proxy Servers. Edge Proxy Servers can be duplicated to enable high availability—if one Edge Proxy fails, the other Edge Proxy takes over the workload of the failed node. When scaling up OCMS Server nodes, it may be necessary to add additional Edge Proxy Servers to the topology in order to handle the additional connections being established in the system.

Proxy Registrar

The OCMS Proxy Registrar combines the functionality of a SIP Proxy Server and Registrar. Its main tasks include:

- **Registering subscribers.** The Proxy Registrar registers a subscriber’s address and maps it to the actual address of the subscriber’s terminal. The Proxy Registrar stores subscriber contact information in the Location Service data store, and uses this data to create paths between the SIP Application Server and the subscriber.
- **Proxying requests onward.** Upon receiving SIP requests, the Proxy Registrar finds the current contact information of the subscriber using the Location Lookup Service or ENUM (*TElephone NUmber Mapping*) Service. The Proxy Registrar replaces the request destination URI with the current, correct SIP address as returned by one of the lookup services, and proxies the request to this destination.

Figure 1–8 Proxy Registrar

Location Lookup Service

The Location Lookup Service stores registration information for all subscribers, as defined by RFC3261. This information is used by the Proxy Registrar to reach subscribers at the right client at any time. For example, a subscriber can connect to OCMS using a client at home or work

Registration data is stored in an Oracle database. The Proxy Registrar uses the Location Service to look up the subscriber's actual, current contact information and proxy requests to that URI. The Proxy Registrar thus creates a direct, reusable connection between the user and the node. The Proxy Registrar uses this connection to route subsequent requests to the correct destination. The client, meanwhile, must regularly refresh its state so as to keep the Location Service data current.

ENUM Lookup Service

If an incoming SIP request destination URI includes a telephone URI, the Proxy Registrar must translate the telephone number to a SIP address, using an ENUM (*TElephone NUmber Mapping*) service. OCMS provides an ENUM Service which uses a configured DNS server to look up the telephone number and translate it into a SIP address. The ENUM Service replaces the telephone number destination URI with the translated SIP address and proxies the request to its destination.

The main tasks of the ENUM Lookup Services are as follows:

1. Convert the telephone destination in an incoming request URI into a host name.
For example:

```
$ORIGIN 2.4.2.4.5.5.5.5.1.4.1.e164.arpa.
```

2. Look up the converted host name in the configured DNS server. The DNS server returns a matching SIP address based on its search for the phone number.

```
IN NAPTR 10 100 "u" "E2U+sip" "!^.*$!sip:4242@555telco.example.com!"
```

3. Replace the telephone number URI with a properly formatted SIP address.
4. Proxy the request to the SIP address.

Presence Server

OCMS includes its own Presence Server, based on the following: RFCs 2778, 3265, 3856, 3857, 3858, 3859, 3863, 3903, as well as OMA Presence Enabler 1.0. The OCMS Presence Server handles registration, storage, and retrieval of presence information.

Presence describes a user's availability and willingness to communicate. The Presence Server can signal whether users are on- or offline and whether they are idle or available. The Presence Server enables users publish their contact details such as instant messaging handle, mobile phone number, and audio and video capability.

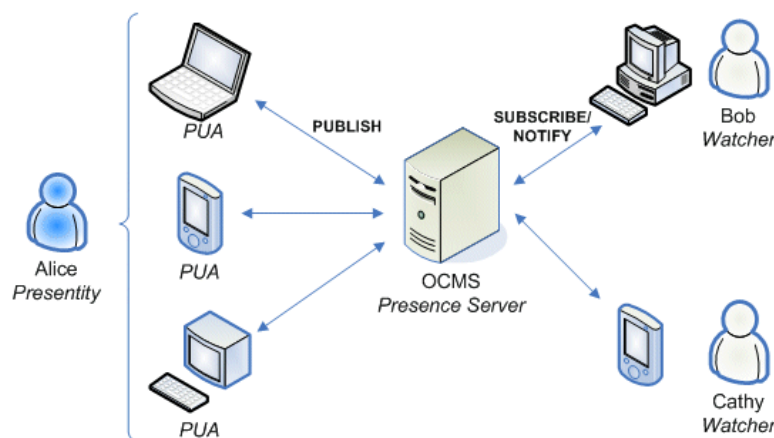
A number of roles are defined in the context of Presence:

- **Presentity**—A Presentity is a user entity that provides presence information to the Presence Server. A presentity can have a number of PUAs, such as a computer at work, a computer at home, and a mobile device.
- **Presence User Agent (PUA)**—A device that provides presence information, such as an instant messaging application (Oracle Communicator), or a mobile device. A PUA provides presentity information to the Presence Server.
- **Watcher**—Requests presence or watcher information from the Presence Server. The two types of watchers include the fetcher and subscriber.
 - **Fetcher**—Retrieves a presentity's presence data from the Presence Server.
 - **Subscriber**—Subscribes to a presentity's presence information, so as to be updated with current information regarding that presentity's presence.

The Presence Server does the following:

- Processes presence PUBLISH requests
- Composites event state into a presence document for a presentity
- Accepts SUBSCRIBE requests from watchers to create subscriptions to a given presentity's presence data
- Acts as a notifier, generating NOTIFY requests to notify subscribers of the state of their subscribed presentity

Figure 1–9 Basic Presence Functionality



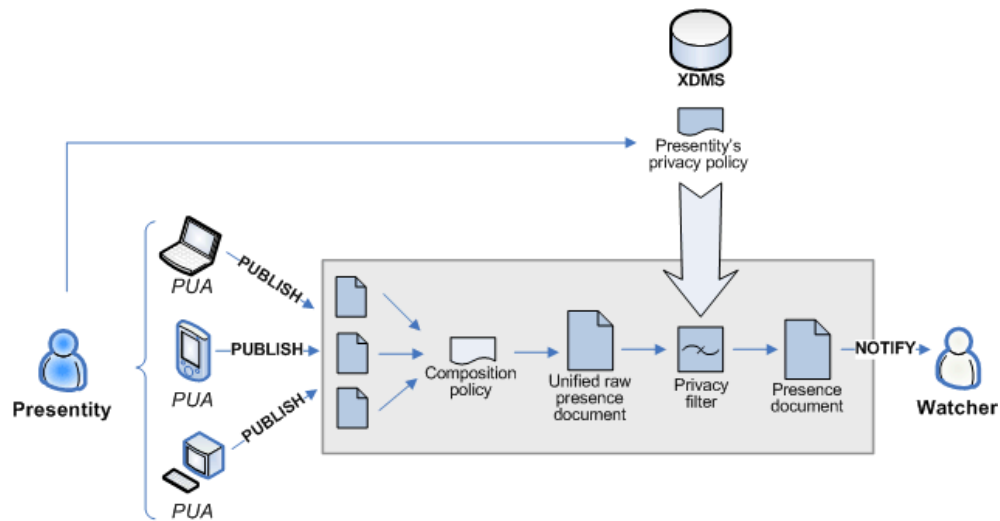
As illustrated in [Figure 1–9](#), presentity Alice has three Presence User Agents (PUAs). A client running on any of these PUAs publishes Alice's presence to the OCMS Presence Server. Meanwhile, watchers Bob and Cathy want to subscribe to Alice's presence information.

Bob and Cathy each run a client that sends the Presence Server a SUBSCRIBE request for Alice's presence. The Presence Server consults Alice's presence policy document in order to determine if Bob and Cathy are permitted to subscribe to his presence. If they are, then the Presence server sends a NOTIFY message containing information about Alice's current presence state. Whenever Alice's presence state changes, the Presence Server sends a NOTIFY message to Bob and Cathy's clients informing them of the change

How the Presence Server Works

The following example illustrates how the Presence Server manages presence information.

Figure 1–10 How the Presence Server Works



A user's presence information can change. For example, the user might be using a different PUA, such as a mobile device or a laptop, or the user may be idle or away. The following describes how the Presence Server handles this change in data:

1. The presentity uploads a policy document that specifies the information to which each watcher is entitled. For example, a user might only want particular watchers to see whether or not she is online.
2. Each PUA sends the new presence information to the Presence Server and issues a SIP PUBLISH request. The presence information is sent in the form of a presence document.
3. The Presence Server receives the presence documents and merges the data into a single document using a composition policy that specifies rules regarding merging presence documents.
4. The unified document is filtered using the privacy policy uploaded to the Presence Server by the presentity (see step 1). This filtering removes any details that the presentity does not want to provide to a given watcher.
5. The Presence Server sends the watcher a NOTIFY request containing the presence document.

Application Router

OCMS Application Router is a SIP application that routes incoming SIP requests to the correct application. The Application Router routes requests by placing route headers in each SIP request it processes. A number of route headers can be placed in a request, each representing a different destination URI. The SIP request is either sent through the chain of destination URIs, or proxied to a new URI upon arriving at its first destination.

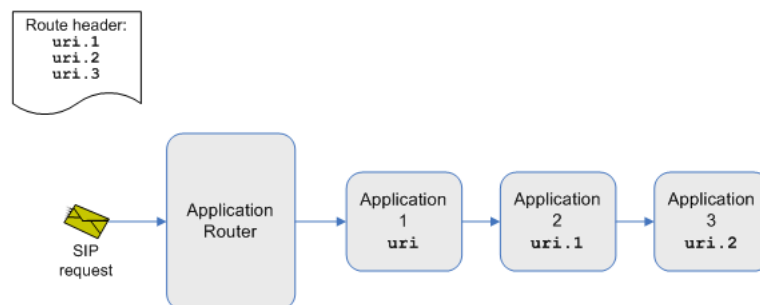
The Application Router typically routes SIP requests to the Proxy Registrar or the Presence Server, respectively. Any number of additional application URIs can be configured in the Application Router.

Modes of Operation

The Application Router operates in two modes: standard and incremental.

Standard Mode The Application Router embeds any number of destination URIs, or routes, in the headers of incoming SIP requests. The SIP request follows this chain of URIs until the request is consumed. The order of URIs is determined by the incremented alias assigned to each route (`uri.1`, `uri.2`, and so on).

Figure 1–11 The Application Router in Standard mode



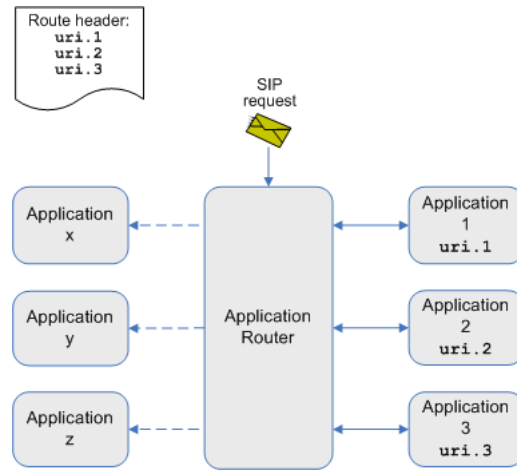
Incremental Mode The Application Router incrementally embeds each route within the incoming SIP request, along with a route back to the Application Router. The SIP request is sent to the first destination, and then returns to the Application Router. The Application Router examines the SIP request's destination URI, which results in one of two possible outcomes:

- If the destination URI has been changed by the application to which the request was sent, the Application Router proxies the SIP request to the new URI.
- If the destination URI has not changed, the Application Router embeds the second route in the header of the SIP request and sends it on its way. Again, the SIP request arrives at the second destination, whereupon it returns to the Application Router. The Application Router must, once again, decide whether to proxy the SIP request to a new URI or embed the third route in the header of the SIP request.

For example, as shown in the following illustration, the Application Router embeds within a SIP request destinations `uri.1`, `uri.2`, and `uri.3`. The SIP request goes first to `uri.1`. If application 1 changes the destination URI of the SIP request, the Application Router routes the request to the new URI, or application *x* (see [Figure 1–12](#)). Otherwise, the Application Router routes the request to the next URI configured in the route header, namely `uri.2`.

Here again, application 2 may change the destination URI of the SIP request, in which case the SIP request continues on to application y (see [Figure 1-12](#)). Otherwise, the Application Router routes the SIP request to uri . 3, and so on.

Figure 1-12 The Application Router in Incremental Mode



Using the Application Router in Standard Mode: an Example

The Application Router can be used in conjunction with a call screening application, for example. In this scenario, the Application Router runs in standard mode with two destination URIs configured in the route header: one to the call screening application and one to the OCMS Proxy Registrar.

An incoming SIP request is intercepted by the Application Router, which embeds both the call screening application URI and the Proxy Registrar URI in the route header of the SIP request. The SIP request continues on to the call screening application, which determines whether or not to put through the request for a call to a given user.

If the call screening application accepts the call, the SIP request continues on to the Proxy Registrar, which forwards the request to the correct destination.

If the call screening application rejects the call, it responds with a "403 Forbidden" error message for example, stopping the SIP request and breaking the routing chain.

Using the Application Router in Incremental Mode: an Example

The Application Router can be used in the context of a call forwarding application. A call forwarding application typically forwards calls by modifying the SIP request destination URI. For example, a call forwarding application might change the destination URI to the URI of a voice mail server.

This is accomplished by using the Application Router in incremental mode to intercept and proxy the SIP requests. SIP requests are sent to the Application Router, which forwards the requests to the call forwarding application and places a return route to itself in the header of the SIP request. The call forwarding application determines whether or not to send the SIP request to the voice mail server and consequently modifies the SIP request destination URI. As the SIP request destination URI has changed, the SIP request returns to the Application Router, which proxies the SIP request to the destination determined by the call forwarding application.

Suppose the SIP request returns to the Application Router, but the call forwarding application has not changed its destination URI. In this case, the Application Router

sends the SIP request to the next application as configured in the Application Router settings.

Subscriber Data Services

The OCMS subscriber data services stores user authentication data, user, role, and policy data, as well as user location data. In a scaled, highly available configuration, the shared database enables all OCMS SIP application servers to access stored data.

The following data is stored:

- [Authentication and Authorization Data](#)
- [User Data](#)
- [Location Lookup Data](#)

Authentication and Authorization Data

Authentication and authorization data provide the primary framework through which access control is configured for the OCMS. Authentication and authorization data can be stored on a RADIUS server, or on Oracle Identity Management.

User Data

The User database stores private subscriber IDs used for subscriber authentication. The Proxy Registrar authenticates users by mapping private subscriber IDs stored in the user database to public subscriber addresses in SIP requests and responses. User data is stored on Oracle Identity Management.

Location Lookup Data

Location data is always stored in a persistent database. The database persists Location data beyond a server restart.

Logging

The Logging service is used to log and monitor system events and SIP traffic. Logs can be used to audit and debug the system.

Session Replication

The Session Replication module handles the replication of session state among multiple SIP Application server nodes. Built on top of OC4J clusters, Session Replication is used only in high availability scenarios.

Deployment Topologies

This chapter discusses OCMS deployment topologies in the following sections:

- ["About Deployment Topologies"](#)
- ["Topology Components"](#)
- ["Supported OCMS Topologies"](#)
- ["Deploying OCMS as a Highly Available SIP Network"](#)
- ["Deploying OCMS as a Presence Server"](#)
- ["Deploying a Scalable Presence Deployment"](#)
- ["Deploying OCMS as an Instant Messaging Service"](#)
- ["Deploying an OCMS Testing Environment"](#)
- ["Configuration Recommendations"](#)

About Deployment Topologies

OCMS supports single-node and clustered, multi-node deployment topologies.

A single-node deployment consists of a single SIP Application Server instance. This deployment, which typically hosts SIP applications along with a database server, is appropriate for running a testing environment or a very small deployment of OCMS.

A SIP Application Server cluster is defined as a set of SIP Application Server instances that share state related to the applications. A cluster consists of one or more application server nodes, with each node running one instance of OCMS.

A highly available OCMS cluster provides the following:

- Replication of objects and values contained in a SIP Application Session.
- Database backed location service data.
- Load balancing of incoming requests across OCMS SIP application servers.
- Overload protection protects the server from malfunctioning in the event of overload and rejects traffic which cannot be handled properly.
- Transparent failover across applications within the cluster. If an instance of an application fails, it becomes unresponsive and the session can fail over to another instance of the application, on another node in a cluster.

Table 2–1 Additional Information

For more information on...	See:
OCMS installation	<i>Oracle Communication and Mobility Server Installation Guide</i>
Operating systems supported by highly available OCMS clusters	<i>Oracle Communication and Mobility Server Certification Guide</i>
Configuring a highly available clustered Oracle Application Server environment	<ul style="list-style-type: none"> ■ The "Application Clustering" chapter in <i>Containers for J2EE Configuration and Administration Guide</i>. ■ The "Active-Active Topologies" chapter in <i>Oracle Application Server High Availability Guide</i>.
Configuring highly available OCMS topologies	Chapter 5, "Configuring High Availability" in this guide

Topology Components

Components of a highly available topology include the following:

- [Third-Party Load Balancer](#)
- [Edge Proxy Nodes](#)
- [SIP Application Servers](#)
- [Aggregation Proxy](#)
- [Proxy Registrar](#)
- [User Dispatcher](#)

Third-Party Load Balancer

A third-party load balancer balances the load of incoming traffic among the Edge Proxy nodes. It also deflects failed Edge Proxy nodes and redirects traffic to other Edge Proxy nodes.

Edge Proxy Nodes

The Edge Proxy nodes form sticky connections between clients and servers for the duration of the session, creating a path between a client and server and sending SIP traffic over that path. The Edge Proxy nodes balance the load of SIP traffic among the SIP Application Servers.

Both Edge Proxy Servers are configured with a virtual IP address. Each Edge Proxy node detects failed SIP Application Server nodes and fails over to the remaining SIP Application Server nodes.

SIP Application Servers

The SIP Application Servers are all linked to each Edge Proxy node. The SIP Application Servers are linked to each other through OPMN. The OCMS SIP state on each computer is replicated to the other two nodes. If one SIP Application Server node fails, another node takes over, using the replicated state of the failed node.

For more information on replicating states among OAS instances and configuring clustering, see "[Setting Up a Highly Available Cluster of OCMS Nodes](#)".

Aggregation Proxy

The Aggregation Proxy authorizes Web Service calls and authenticates XCAP traffic. The Aggregation Proxy then proxies this traffic to the Parlay X Web Service and XDMS. This is an optional component.

Proxy Registrar

The OCMS Proxy Registrar combines the functionality of a SIP Proxy Server and Registrar. Its main tasks include registering subscribers, looking up subscriber locations, and proxying requests onward. The Proxy Registrar stores user location and registration data on the Oracle database. This is an optional component.

For more information, see "[Proxy Registrar](#)".

User Dispatcher

The User Dispatcher enables the Presence and XDMS applications to scale. The User Dispatcher is a proxy that dispatches SIP and XCAP (over HTTP) requests to their appropriate destinations on a consistent basis.

Presence Dispatching

Because the Presence application maintains the states for all users in the deployment, the User Dispatcher enables scaling (distribution) of the Presence application. The User Dispatcher supports request dispatching to the following Presence sub-applications, which use the SIP and XCAP (over HTTP) protocols:

- Presence server
- Presence XDMS
- Shared XDMS

Supported OCMS Topologies

Supported topologies include:

- [Deploying OCMS as a Highly Available SIP Network](#)
- [Deploying OCMS as a Presence Server](#)
- [Deploying OCMS as an Instant Messaging Service](#)
- [Deploying an OCMS Testing Environment](#)

Note: Only the Oracle Application Server installation mode supports high availability. For more information, refer to the *Oracle Communication and Mobility Server Installation Guide*.

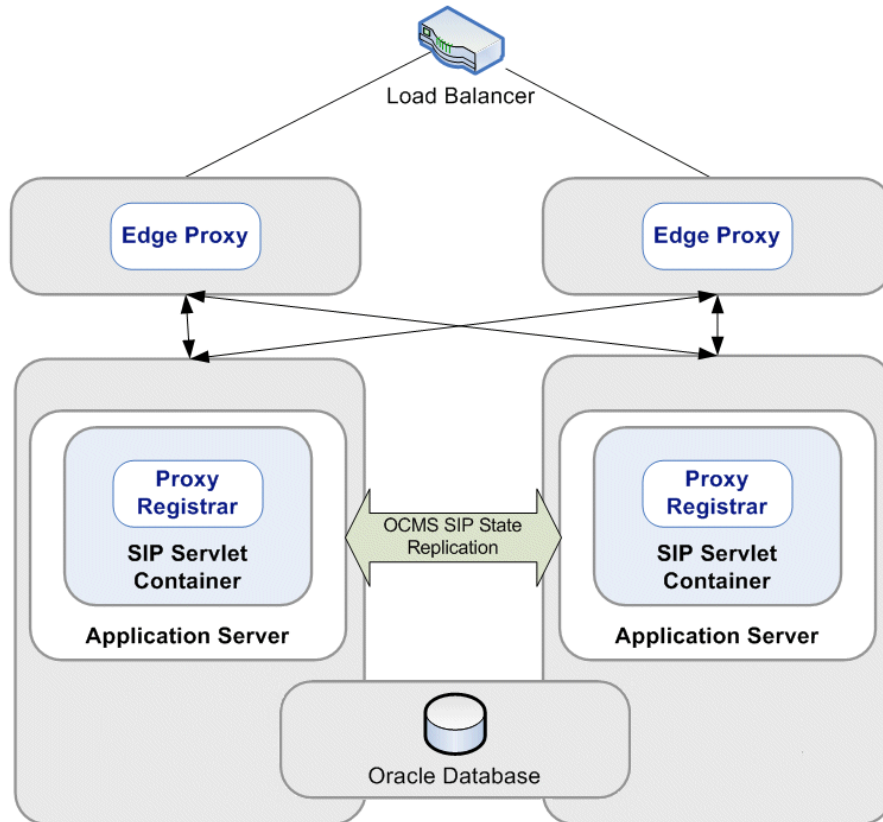
Deploying OCMS as a Highly Available SIP Network

When deployed as a highly available SIP network, OCMS can be used to implement a basic VoIP system, enabling voice and video calls. This topology (illustrated in [Figure 2-1](#)) includes the following:

- A hardware load balancer
- A cluster of Edge Proxy Servers

- A cluster of two SIP Application Servers, each running a SIP Servlet Container
- Replicated databases, including user data, authentication data, and user location information

Figure 2–1 OCMS as a Highly Available SIP Network



This topology provides a highly available SIP network capable of handling millions of users. Each SIP Application Server must run an Oracle database and Proxy Registrar application.

The SIP network topology includes hardware and software components described in [Table 2–2](#).

Table 2–2 SIP Network Topology Hardware and Software Requirements

Hardware	Software	Installation Type ¹
Load balancer	N/A	N/A
Two computers with at least 4 GB of RAM and a dual 2.8 Ghz CPU	Edge Proxy	Custom installation
Two computers with at least 4 GB of RAM and a dual 2.8 Ghz CPU	<ul style="list-style-type: none"> ■ OAS 10.1.3.4 ■ Oracle database ■ Proxy Registrar Application 	Typical installation

¹ Refer to the *OCMS Installation Guide* for more information.

Load Balancer

A load balancer or DNS round-robin algorithm balances the load of incoming traffic among the Edge Proxy nodes. Using the DNS round-robin algorithm requires all clients to implement DNS lookup.

See also: "[Topology Components](#)" for a description of the components used in this topology.

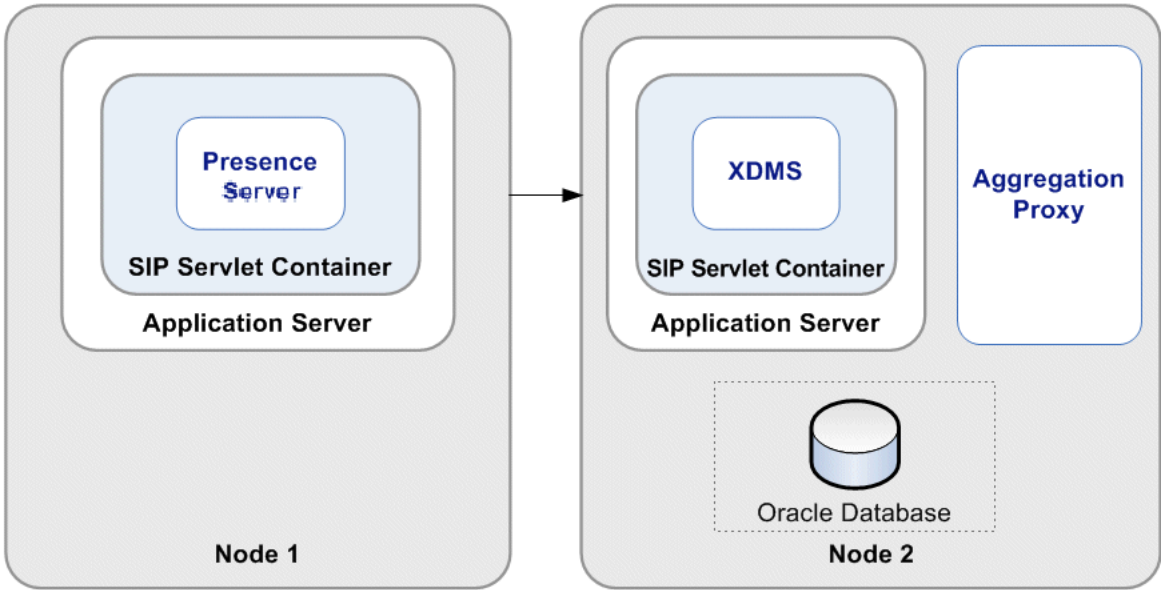
Deploying OCMS as a Presence Server

OCMS can be deployed as a Presence Server. The Presence Server topology is deployed on two nodes: one running the Presence Server and the other running the Aggregation Proxy and XDMS. This topology (illustrated in [Figure 2-2](#)) can be implemented within an IMS network to provide Presence functionality.

The Presence Server topology includes the following:

- One SIP Application Server node with the Presence Server
- One SIP Application Server node with an XDMS and an Aggregation Proxy

Figure 2-2 Presence Server Topology



The Presence Server topology includes hardware and software components described in [Table 2-3](#).

Table 2-3 Presence Server Topology Hardware and Software Requirements

Hardware	Software	Installation Type ¹
One computer with at least 4 GB of RAM and a dual 2.8 Ghz CPU	<ul style="list-style-type: none"> ■ OAS 10.1.3.4 ■ Presence Server 	Typical installation

Table 2–3 (Cont.) Presence Server Topology Hardware and Software Requirements

Hardware	Software	Installation Type ¹
One computer with at least 4 GB of RAM and a dual 2.8 Ghz CPU	<ul style="list-style-type: none"> ■ OAS 10.1.3.4 ■ XDMS ■ Aggregation Proxy ■ Oracle database 10.2.0.3 or 11.1.0.7 (required for the Aggregation Proxy). 	Typical installation

¹ Refer to the *OCMS Installation Guide* for more information.

XDMS

Manual post-installation configuration is required for the XDMS, involving configuring Presence as an XDMS. For more information, refer to the *OCMS Installation Guide*.

See also: See "[Topology Components](#)" for a description of the components used in this topology.

Deploying a Scalable Presence Deployment

This section describes the recommended and supported deployment topology for a large scale Presence Solution requiring Presence, XDMS, and User Dispatcher. It illustrates the typical flows from a multi-node perspective.

To scale across multiple nodes, the User Dispatcher component dispatches all traffic targeting a particular Presenty to the same Presence Server instance.

Presence Cluster

A Presence Cluster is defined as a set of Presence Nodes connected after one or more Load Balancers. The Presence Cluster is responsible for processing incoming subscribe and publish requests made towards the presence event-package and of course for sending out notify's whenever appropriate. The Presence Cluster will also accept and processing subscribe requests for the presence.winfo event-package.

The Presence Cluster will interact with the XDM Cluster in order to obtain information needed to complete its responsibilities. The information queried of the XDM Cluster is user's presence-rules and pdf-manipulation documents.

The Presence Cluster is layered into the following three distinct tiers:

- The load-balancing layer, responsible for dispatching incoming traffic to the User Dispatchers. The load balancers are stateless and do not understand SIP as a protocol.
- The user-dispatching layer, responsible for dispatching traffic based on user information. A user is assigned to a particular Presence Server instance and all traffic destined to that user will be dispatched to the same Presence Server instance. Even though each User Dispatcher is stateless and does not share state with the other User Dispatchers, they still need to have the same view of the Presence Server tier.
- The bottom layer is where the Presence Server instances reside. Each instance is separated from the others and does not share any state with any other instances.

The purpose of the Presence Server tier is to serve incoming SUBSCRIBE and PUBLISH requests destined to the presence event-package as well as servicing subscriptions to the presence.wininfo event-package.

The Presence Cluster consists of the following physical nodes:

- The Load Balancer, such as an F5.
- The Presence Node, which consists of the following components:
 - User Dispatcher
 - Presence Server

XDM Cluster

The XDM cluster is defined as a set of XDM Nodes connected after one or more Load Balancers. The XDM cluster processes all XDM related traffic, that is, SIP subscribe traffic towards the ua-profile event-package and XCAP traffic. As such, it deals with everything that has to do with manipulating XML documents. The XDM Cluster uses a database for actual storage of the XML documents but note that the database, and potentially its cluster, is not part of the XDM Cluster.

The XDM cluster consists of the following layers:

- The load-balancing tier, responsible for dispatching both SIP and XCAP traffic to the next layer. For XCAP traffic the next tier is the Aggregation Proxy but for SIP, the traffic goes directly to the User Dispatcher layer.
- Aggregation Proxy layer – authenticates incoming traffic and upon successful authentication it forwards the requests to the User Dispatcher layer. All XCAP traffic for external traffic goes through the Aggregation Proxy layer. Internal traffic, however, will not go through the Aggregation Proxy but rather directly to the User Dispatchers.
- User Dispatcher layer – from a SIP perspective it carries out the exact same duties and functions as in the Presence Cluster (it is the same kind of traffic after all). The main difference in the XDM Cluster compared to the presence one is that in the XDM Cluster the User Dispatchers will also have to handle XCAP traffic. However, the XCAP traffic is treated in the exact same way as SIP and the purpose of the User Dispatcher for XCAP traffic is the same as for SIP: to extract user information based on the request and then dispatch it to the correct XDMS instance.
- The XDM Server layer has the same function as the Presence Servers in the Presence Cluster. The XDMS instances serve incoming SUBSCRIBE requests for the event-package ua-profile and will whenever appropriate send out NOTIFY messages to all registered subscribers. Note that the XDMS does not accept PUBLISH requests and updating the state of the Resources (which are XML documents) is through XCAP operations. An XDM Client can manipulate the documents managed by an XDMS by issuing appropriate XCAP operations. A successful XCAP operation may alter the content of a document whereby the XDMS would send out NOTIFY messages to the subscriber of that document to inform them about the change. Whenever the XDMS needs to get an XML document it queries the next layer, the database layer.
- The Database tier physically stores the XML documents managed by the XDMS. This tier guarantees high-availability and scalability so that if one of the nodes in the database layer fails, documents that resided on that node will still be accessible to the XDMS without any loss of data or service.

The XDM Cluster consists of the following physical nodes:

- The Load Balancer, such as an F5.
- The XDM Node, which consists of the following components:
 - Aggregation Proxy
 - User Dispatcher
 - The XDM Server (XDMS)
- The database.

Presence Node

The Presence Node is the main component in the Presence Cluster and is responsible for dispatching the incoming traffic to the correct Presence Server instance and of course servicing users with presence information. The User Dispatcher serves the same purpose both in a single node deployment and in a multi-node deployment. That is, its purpose is to dispatch incoming traffic to a particular PS instance and if this instance is running on the same physical node or not is of no relevance to the User Dispatcher. The User Dispatcher identifies a particular node by its full address, that is, the IP address and port, and has no concept of local instances.

A Presence Node will always have a User Dispatcher deployed that serves as the main entrance into the node itself. Typically, the User Dispatchers listen to port 5060 and the other Presence Servers on that node listen on other ports. In this way, a single node will appear as one Presence Server to clients but is in fact multiple instances running behind the User Dispatcher. Each of the components deployed on the Presence Node is executing in their own separate Java Virtual Machine. That is, the User Dispatcher and the Presence Server instances execute in their own OC4J and SIP containers. The reason for this is to be able to utilize all the available memory on that machine.

XDM Node

The XDM Node always has an Aggregation Proxy deployed that typically listens on port 80 for XCAP traffic. The Aggregation Proxy authenticates incoming traffic and upon successful authentication forwards the request to the User Dispatcher. As with the Presence Node, the XDM Node will also have a User Dispatcher deployed (usually on port 5060) and for SIP traffic there is no difference between the XDM and Presence Nodes. The difference between the two types of nodes is that the User Dispatcher will also dispatch XCAP traffic. As it does with SIP, it extracts the user id out of the request and, based on that, maps the request to a particular XDMS instance to which it forwards the request.

There will be a number of XDMS instances deployed to which the User Dispatcher dispatches both SIP and XCAP traffic. Just as in the case of the Presence Server instances on the Presence Node, each XDMS instance is not aware of the others and executes in isolation.

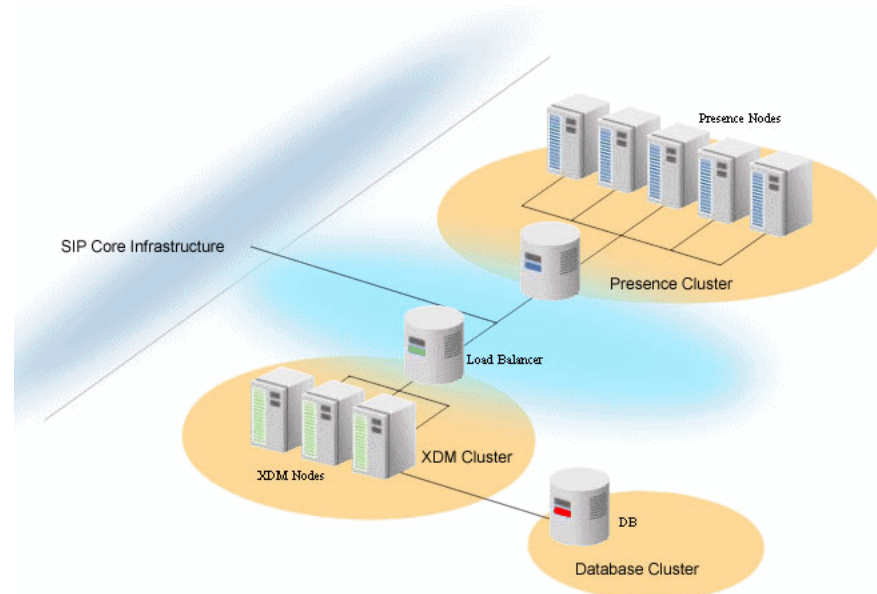
The Aggregation Proxy and User Dispatcher are deployed onto the same OC4J container and use the same Java Virtual Machine.

Complete Presence and XDM Cluster

Figure 2–3 shows a complete Presence and XDM cluster with all necessary components. This figure also illustrates that the two clusters, Presence and XDM, are treated as two separate clusters and the way into those two networks for initial traffic is always through their respective Load Balancers. Even the Presence Servers will

actually go through the Load Balancer of the XDM Cluster when setting up subscriptions. However, once a subscription has been established the subsequent requests will not go through the Load Balancer but rather directly to the XDMS instance hosting the subscription. All nodes in the XDM Cluster are directly accessible from the Presence Cluster.

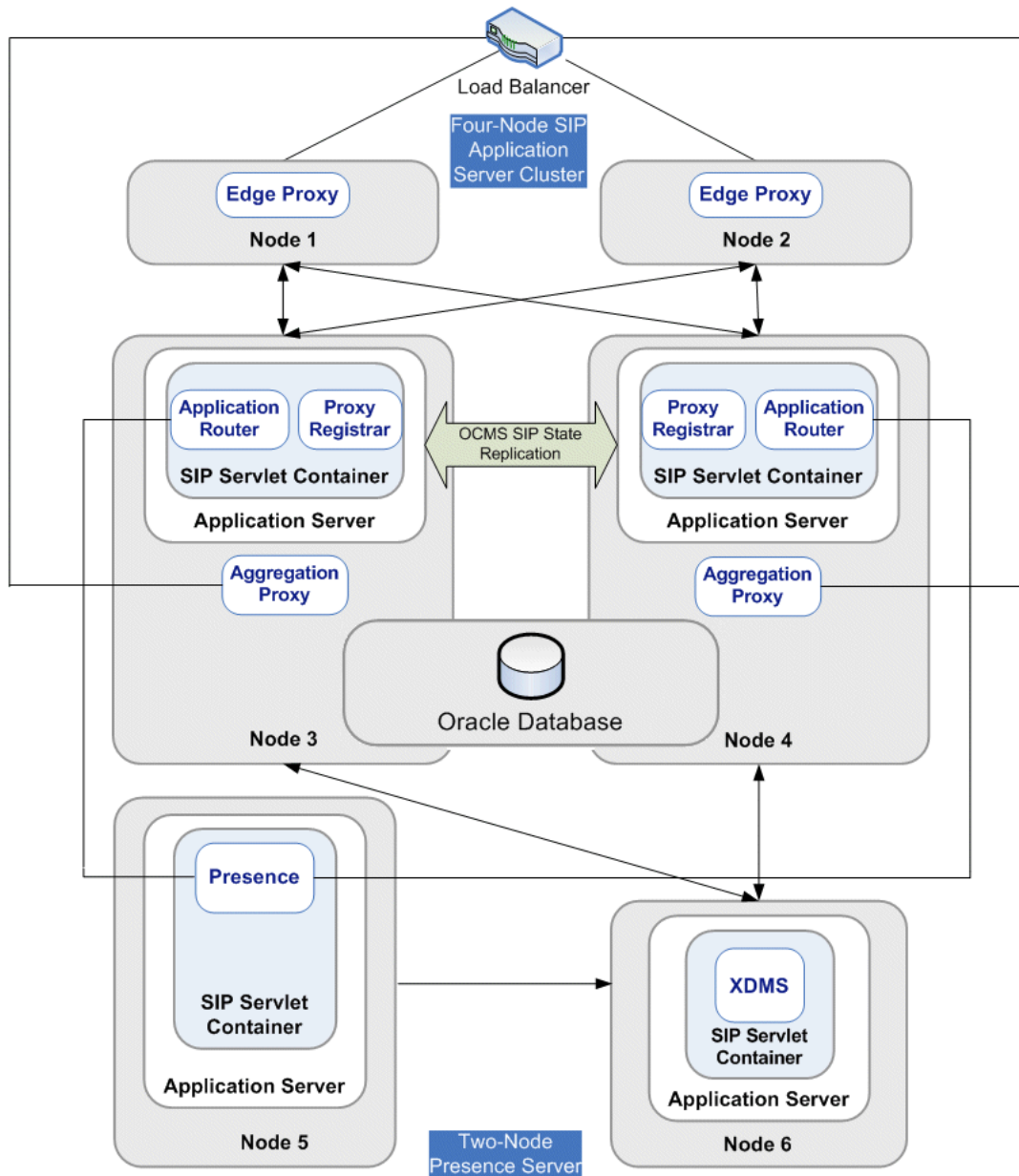
Figure 2–3 Presence and XDM Nodes



Deploying OCMS as an Instant Messaging Service

The OCMS Instant Messaging topology is a highly available, topology that enables messaging, including instant messaging client applications. [Figure 2–4](#) illustrates a sample topology consisting of six nodes in three clusters. The IM topology comprises a four-node SIP network topology and a two-node Presence Server topology, with the addition of the Application Router. The Application Router routes SIP requests to either the Proxy Registrar or the Presence Server, enabling registering and retrieving user contact and location data, as well as handling all aspects of Presence publication and notification. The Aggregation Proxy on either SIP Application Server node is used to authenticate subscriber access to presence documents stored on the XDMS. This topology provides the server-side functionality behind instant messaging client applications.

Figure 2–4 Instant Messaging Service Topology



This topology includes hardware and software components described in [Table 2–4](#).

Table 2–4 Topology Hardware and Software Requirements

Hardware	Software	Installation Type ¹
Load balancer	N/A	N/A
Two computers with at least 4 GB of RAM and a dual 2.8 Ghz CPU	Edge Proxy	Custom installation

Table 2–4 (Cont.) Topology Hardware and Software Requirements

Hardware	Software	Installation Type ¹
Two computers with at least 4 GB of RAM and a dual 2.8 Ghz CPU	<ul style="list-style-type: none"> ■ OAS 10.1.3.4 ■ Oracle Database ■ Aggregation Proxy ■ Application Router ■ Proxy Registrar 	Typical installation
One computer with at least 4 GB of RAM and a dual 2.8 Ghz CPU	<ul style="list-style-type: none"> ■ OAS 10.1.3.4 ■ Presence Server 	Typical installation
One computer with at least 4 GB of RAM and a dual 2.8 Ghz CPU	<ul style="list-style-type: none"> ■ OAS 10.1.3.4 ■ XDMS 	Typical installation

¹ Refer to the *OCMS Installation Guide* for more information.

For more information about scaling this hybrid topology, see "[Deploying OCMS as a Highly Available SIP Network](#)" and "[Deploying OCMS as a Presence Server](#)".

See also: "[Topology Components](#)" for a description of the components used in this topology.

Deploying an OCMS Testing Environment

An OCMS testing environment is deployed on a single SIP application server node. A single-node OCMS topology is appropriate for testing, demonstrations, and small enterprises.

Note: Because the testing environment is deployed on a single node, it cannot provide high availability.

[Figure 2–5](#) illustrates a single-node deployment which includes a Proxy Registrar.

Figure 2–5 Single Node Deployment

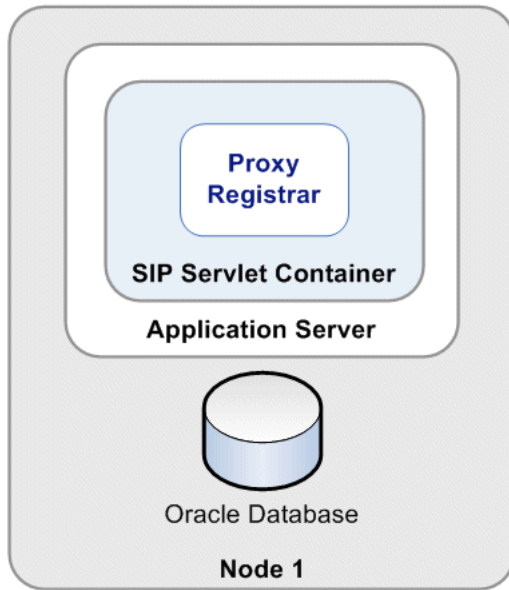
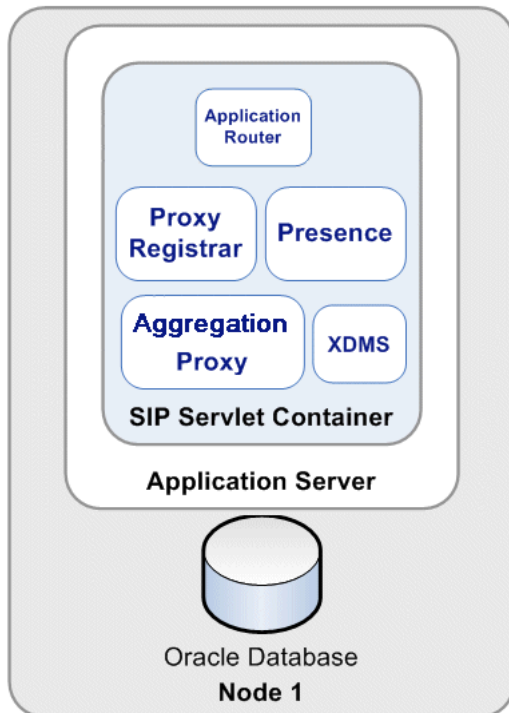


Figure 2–6 illustrates a single-node deployment with Proxy Registrar, Presence, Aggregation Proxy, XDMS, and Application Router.

Figure 2–6 Single Node Deployment with Presence



The single node topology includes hardware and software components described in Table 2–5.

Table 2–5 Single Node Topology Hardware and Software Requirements

Hardware	Software	Installation Type ¹
One computer with at least 4 GB of RAM and a dual 2.8 Ghz CPU	<ul style="list-style-type: none"> ■ OAS 10.1.3.4 ■ Oracle database ■ Proxy Registrar 	Typical installation
	If deployment includes Presence: <ul style="list-style-type: none"> ■ Presence application ■ Aggregation Proxy 	Typical installation

¹ Refer to the *OCMS Installation Guide* for more information.

See also: "[Topology Components](#)" for a description of the components used in this topology.

Configuration Recommendations

The following is recommended for most topologies:

- **Use TCP as the transport protocol.** SIP clients use UDP or TCP to transport SIP messages. If there is a concern about exceeding UDP MTU size limits, TCP should be used. The preference for TCP can be enforced by adding NAPTR and SRV records to the DNS indicating that TCP is the preferred protocol. Make sure that clients connecting to OCMS fully support NAPTR and SRV records.
- **Run the OCMS SIP Server in the default Record-Route mode.** The best way for clients to handle NAT/FW in the network is to establish a TCP connection to the server upon registration, and reuse this connection for all incoming and outgoing SIP traffic. This requires the client to have an `outboundproxy` setting that points to the Proxy Registrar, and that the Proxy Registrar is configured to use `record-route`.

Configuring the SIP Server MBeans

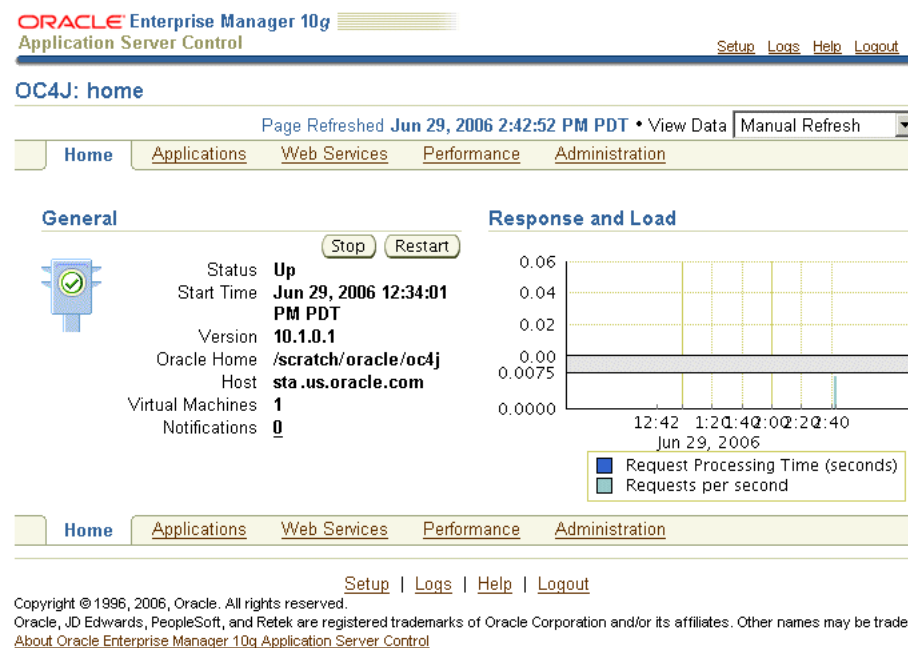
This chapter, through the following sections, describes how to manage the OCMS SIP Server through Oracle 10g Enterprise Manager Application Server Control.

- ["Overview of SIP Server Management"](#)
- ["Starting, Stopping and Restarting the OCMS SIP Server"](#)
- ["Managing OCMS MBeans"](#)
- ["Setting and Viewing the SIP Port"](#)

Overview of SIP Server Management

The OCMS SIP Server is an OC4J container that you manage using the Oracle 10g Enterprise Manager Application Server Control console (Figure 3-1).

Figure 3-1 The Oracle 10g Enterprise Manager Application Server Control



In addition to the standard Application Server Control functions that enable starting, stopping, restarting, deploying, undeploying, and redeploying applications, the Application Server Control MBean browser enables you to configure and manage the OCMS components. Configuring the attributes of the OCMS MBeans (Managed Beans)

enables you to execute administrative tasks and set the basic configuration (port, IP, and host address) of the OCMS SIP Server itself. In addition, the OCMS MBeans enables you to configure and manage presence.

Starting, Stopping and Restarting the OCMS SIP Server

Application Server Control enables you to stop and restart the OCMS SIP Server. Like other OC4J containers, the OCMS SIP Server can be stopped and restarted using the **Stop** and **Restart** buttons on the *Home* page (Figure 3-1). These buttons control the entire OC4J instance; stopping or restarting OC4J also stops or restarts the OCMS SIP Servlet Container and the applications deployed to it.

If you have installed OCMS in standalone mode, you can invoke the start and stop scripts available under `ORACLE_HOME/sdp/bin` from the command line:

```
startocms
```

or

```
stopocms
```

You can also stop or restart OC4J using the `opmnctl` or the `admin_client.jar` command-line utility to stop, start, or restart OC4J or its applications.

To stop all OPMN-managed processes including OC4J on a local Oracle Application Server instance using `opmnctl`:

```
opmnctl stopall
```

To stop the OC4J instance of OCMS on a local Oracle Application Server instance using `opmnctl`:

```
opmnctl stopproc process-type=ocms
```

To start all OPMN-managed processes, including OC4J, on a local Oracle Application Server instance using `opmnctl`:

```
cd ORACLE_HOME/opmn/bin  
opmnctl startall
```

To restart the OC4J instance of OCMS on a local Oracle Application Server instance using `opmnctl`:

```
cd ORACLE_HOME/opmn/bin  
opmnctl restartproc process-type=ocms
```

Use the following syntax when using `admin_client.jar` to start, stop or restart an application and its child applications on a specific OC4J instance or across an entire cluster.

```
java -jar admin_client.jar uri adminId adminPassword -start|-stop appName
```

For more information, see *Oracle Application Server Administrator's Guide*.

Starting an Application and Stopping a SIP Servlet Application

The **Stop**, **Start** and **Restart** buttons on the *Applications* page (Figure 3-2) control the running status for selected SIP servlet applications.

Figure 3–2 The Applications Page of the Application Server Control

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: as10132_ocms.stanf10.us.oracle.com >
OC4J: ocms

Page Refreshed Oct 17, 2006 2:32:56 PM PDT

Home Applications Web Services Performance Administration

This page shows the J2EE applications and application components (EJB Modules, WAR Modules, Resource Adapter Modules) deployed to this OC4J instance.

View Applications

(Start) (Stop) (Restart) (Undeploy) (Redeploy) (Deploy)

Select All | Select None | Expand All | Collapse All

Select	Name	Status	Start Time	Active Requests	Request Processing Time (seconds)	Active EJB Methods	Application Defined MBeans
<input type="checkbox"/>	▼ All Applications						
<input type="checkbox"/>	ascontrol	↓					
<input type="checkbox"/>	▼ default	↑	Oct 16, 2006 11:20:54 PM PDT	0	0.001	0	
<input type="checkbox"/>	presence	↑	Oct 16, 2006 11:20:59 PM PDT	0	0.00	0	
<input checked="" type="checkbox"/>	▶ subscriberdataservices	↑	Oct 16, 2006 11:20:55 PM PDT				

Note: Changes made to the SIP Container applications persist when you restart OC4J; changes made to the logging persist when updated.

Managing OCMS MBeans

The Application Server Control Console’s MBeans browser (Figure 3–3) enables you to view and configure MBeans. An MBean (managed bean) is a Java object that represents a JMX-manageable resource in a distributed environment, such as an application, a service, a component, or a device. MBeans expose a management interface, including a set of attributes and operations. This interface does not change during the lifetime of an MBean instance. MBeans can also send notifications when defined events occur.

The MBean attributes enable you to configure the OCMS SIP Server.

Figure 3–3 Viewing MBeans

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: iashome.stabd54.us.oracle.com > OC4J: ocms > Application: subscriberdataservices >
Application MBeans

For an introduction to the capabilities of the MBean Browser, see [About the MBean Browser](#).

Search MBean Name (Find)

Application: subscriberdataservices

- AccountLockoutService
 - AAService**
 - Constant
 - Exponential
 - Linear
- CommandService
- LoginFailureService
- ModelMBeanDeployer

MBean: AccountLockoutService:AAService

Page Refreshed Jan 28, 2007 4:21:20 PM

Name subscriberdataservices:name=AAService,type=AccountLockoutService
Description This service contains methods for locking and unlocking a user account.
Persist Policy Never Related Link: Cluster

Attributes (4) Operations (6)

Name	Description	Access	Value
DefaultLockDuration	The time to use as basis for calculating the real lock duration or if no math function is found.	RW	600
JndiName	The JNDI Name of this Account Lockout Service.	R	AAService
MathFunction	The type of math function to use when calculating duration of account lockout (must be one of Constant, Linear or Exponential).	RW	Linear
SecurityServiceName	The JNDI Name of the Security Service.	R	ejb/com/hotsip/secur

Attributes (4) Operations (6)

See *Oracle Containers for J2EE Configuration and Administration Guide* for information on managing MBeans.

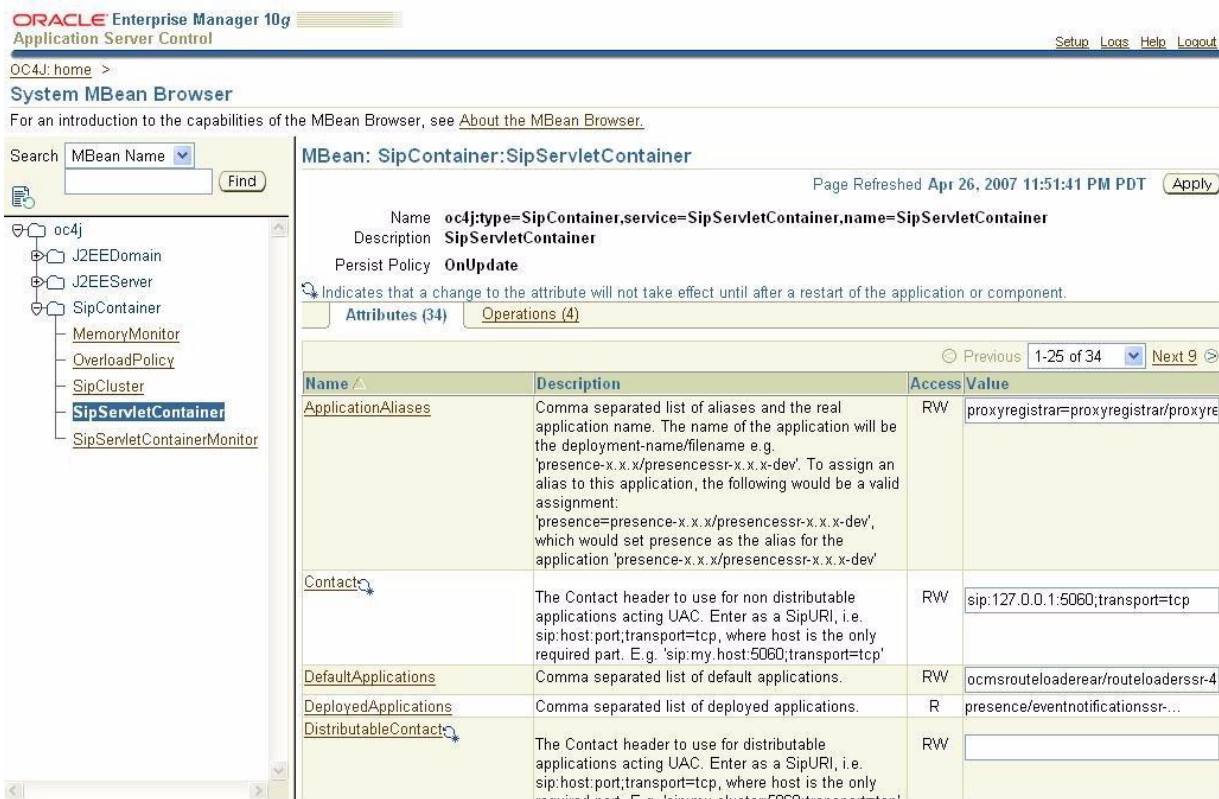
Accessing MBeans

The Application Server Control Console's MBean browser enable you to view, configure and deploy both system MBeans and application-defined MBeans.

You access the MBeans related to the JSR 116-compliant OCMS SIP Servlet Container through the System MBean Browser. For more information, see "[Managing OCMS MBeans](#)".

To display the available System MBeans, the MBean Browser accesses the MBean Server that runs in OC4J.

Figure 3-4 Viewing System MBeans



The SIP Application-based MBeans do not display in the System MBean Browser; instead, they appear in the context of the application that registered them. For example, [Figure 3-3](#) illustrates the MBeans registered by the [Subscriber Data Services](#) application.




Note: MBeans are packaged with the application that they manage.

Accessing SIP Servlet Container MBeans

The System MBean Browser enables you to browse the SIP Servlet Container MBeans ([Figure 3-3](#)). To access the System MBean Browser:

1. Navigate to the OC4J Home page for the SIP Server instance.
2. Click **Administration**. The *Administration* page appears, displaying the available tasks.
3. If needed, expand the JMX section of the task list to display *System MBean Browser*.
4. Click the *Go to Task* icon in the *System Bean Browser* row of the table. The System MBean Browser appears, displaying the available MBeans in the tree view. Selecting an MBean in the tree control enables you to view and edit its attributes, invoke its operations, and manage notification subscriptions. Click **Apply** to commit any changes to the MBean's parameters.

Figure 3–5 The JMX Section of the Task List

JMX		
System MBean Browser		Browse the system MBeans exposed by this OC4J instance.
Notification Subscriptions		View/change subscriptions for notifications for all MBeans.
Notifications Received		View received notifications.

Tip: Use the *Search* function to locate an MBean

Accessing the MBeans for a Selected SIP Application

To view the MBeans for a selected application:

1. Click **Applications** on the SIP Server OC4J home page. A list of applications appears.
2. Click the *Application Defined MBean* icon for the selected application. The *Application MBeans* page appears. The tree view displays the MBeans registered to the application. Selecting an MBean in this tree control lets you view its attributes, operations, statistics, and notifications.
3. Click **Apply** to commit any changes to the MBean's attributes.

Configuring the SIP Servlet Container MBeans

This section describes the configuration parameters for the following System MBeans (listed in [Table 3–1](#)):

Table 3–1 SIP Servlet Container MBeans

Tasks	MBean Name
Tasks include:	SIP Servlet Container
<ul style="list-style-type: none"> ■ Setting or changing the IP address of the SIP Container. ■ Setting or changing the DNS IP address. ■ Setting or changing the domains and realms. ■ Setting or changing the proxy that receives client requests. ■ Setting the default applications for incoming requests. 	
Setting the log levels for the logger components (<i>CUSTOMER</i> , <i>BADMSG</i> , <i>TRAFFIC</i> , <i>CONFIG</i> , <i>TIMER</i> , <i>STATISTICS</i> , <i>FORMAT</i> , and <i>APPLICATION</i>). For more information, see Chapter 12, "Configuring the Logging System" .	SIP Servlet Container Logging

Table 3–1 (Cont.) SIP Servlet Container MBeans

Tasks	MBean Name
Setting bindings that enable clients to traverse NAT (Network Address Translating) entities.	STUN Service
View the current statuses of the system queues.	SIP Servlet Container
Set overload actions when the capacity thresholds for memory usage, Application Queue usage, Network Queue usage, or SIP Session Table usage are reached.	Overload Policy
Set the default timeouts for OC4J clusters.	SIP Cluster

SIP Servlet Container

The SIP servlet container is a standalone Java process that provides the execution environment for the SIP applications. The attributes exposed by the SIP Servlet Container MBean (SipServletContainer) enable you to change values that are set during the installation of OCMS (listed in Table 3–2). For more information on setting the values for these attributes during installation, refer to *Oracle Communication and Mobility Server Installation Guide*.

Table 3–2 Values Populated by Installation of OCMS

Value	SIP Servlet Container Attribute(s)
The IP address of the SIP Container.	IPAddress
The traffic port used by the SIP Container.	SIPPort
The proxy for receiving client requests.	EdgeProxy
The domain (or hostname) of the SIP Server and the SIP realm used for authentication.	DomainsandRealms

Table 3–3 describes the SIP Servlet Container MBean’s attributes. While the format for entering values is *sip:host:port;transport=tcp|udp* for the following attributes, you generally need only enter the host value, as other values are pre-seeded.

- Contact
- DistributableContact
- DistributableRecordRoute
- DistributableVia
- DnsIpAddress
- Via

Table 3–3 Attributes of the SIP Servlet Container

Attribute	Value
ApplicationAliases	A short name for the application. For more information, see "Setting an Alias for an Application" .
Contact	The host, port, and transport used in the <i>Contact</i> header that is embedded in SIP requests by applications acting as User Agent Clients (UACs). Enter this value as a SIP URI. This contact information provides an address that enables the SIP Server to respond.
DefaultApplications	A comma-separated list of applications that are invoked if no application matches a request. By default, the Application Router is set as the default application. If many applications have been deployed to the SIP Container, you should also deploy an application dispatcher as the default application. The application dispatcher (or Application Router in OCMS) routes incoming requests to other applications by addressing them directly. For more information, see "Application Router" in "An Overview of Oracle Communication and Mobility Server" . If no default application is set, then the container designates all of the SIP applications that it locates as the default application. For more information on default application and how the SIP container invokes servlets, see <i>Servlet Mapping</i> in <i>Oracle Communication and Mobility Server Developer's Guide</i> .
DeployedApplications	A read-only list of deployed applications.
DistributableContact	The host, port, and transport placed in the <i>Contact</i> header for distributable applications acting as User Agent Clients (UACs) in a high-availability environment.
DistributableRecordRoute	The host, port, and transport placed in the <i>RecordRoute</i> header for distributable applications in a high-availability environment. Enter this value as a SIP URI in the following format: <code>sip:host:port;transport=tcp.</code> You need only enter the hostname (<code>sip:my.host:5060;transport=tcp.</code>). For high availability configurations, configure this parameter in the following format: <code>sip:<SIP container IP address>:<port>.</code> Remove the transport method to enable the use of any type of transport between the Edge Proxy and OCMS. For more information, see "Configuring the OCMS SIP Containers for High Availability" .
DistributableVia	The host, port, and transport used in the <i>Via</i> header for distributable applications in a high-availability environment. Enter this value as a SIP URI in the following format: <code>sip:host:port;transport=tcp.</code> You need only enter the hostname (<code>sip:my.host:5060;transport=tcp.</code>).

Table 3–3 (Cont.) Attributes of the SIP Servlet Container

Attribute	Value
DnsIpAddress	<p>A comma-separated list with the DNS that the container shall try to resolve against. Enter as a SipURI, that is, sip:host:port;transport=tcp, where host is the only required part. For example, '127.0.0.1:53;transport=udp'. Port and transport defaults to 53 and UDP. An empty string enables the synchronous OS based resolver (no NAPTR or SRV support) which considers the local hosts file. This mode is intended for demonstration or development systems.</p> <p>For details about how DNS resolution takes place, see "RFC 3263: Session Initiation Protocol (SIP): Locating SIP Servers". Because DNS resolution is performed within the context of SIP message processing, any DNS performance problems result in increased latency. It is recommended to use a caching DNS server in a production environment to minimize potential performance problems. You must specify a DNS in production environments.</p>
DomainsAndRealms	<p>This attribute defines a mapping of SIP domains to Java-authenticated realms. This mapping is used to dynamically challenge a SIP request with the appropriate authentication realm during SIP authentication. For example, a domain/realm mapping of <i>voip.com/voip</i> would require a user with the SIP URI of <i>user@voip.com</i> to authenticate against the <i>voip</i> realm in response to the challenge by the SIP servlet container.</p> <p>Enter a comma-separated list of the configured hosted domains and their corresponding realms used for authentication. Applications -- rather than the SIP container itself -- use such a hosted domain to send a 404 (<i>Not Found</i>) message.</p>
DomainLoopDetection	<p>If set to <i>true</i> (the default setting), The SIP container checks whenever a request is about to be sent from the server to a destination that has a resolvable host name. If the host name resolves to one of the server's own listening ports (meaning that the server will send the request back to itself) the server checks the resolved host name against configurations set for the <i>DomainsandRealms</i> and <i>RecordRoute</i> header attributes. If there is no match for the hostname, then the request is blocked and the server responds with a 482 (<i>Loop Detected</i>) message. This protects the server from fake DNS names that point to the server's IP address and then send a request with the fake hostname in the top-most route header. Because the SIP container cannot recognize the hostname, it will never pop the Route header and the request will loop.</p>
IPAddress	<p>The IP address on which the SIP servlet container listens. The default value of 0.0.0.0 designates all IP addresses. For a production environment, change this value to an actual IP address.</p>
NeedClientAuth	<p>Setting this attribute to <i>true</i> enables mutual TLS, where the SIP servlet container not only provides a server certificate during the TLS handshake, but also requires a client certificate. The default value, <i>false</i>, enables the SIP servlet container to act as a TLS server by providing a server certificate during the TLS handshake with the client. This function is only available if <i>useTLS</i> is set to <i>true</i>. Setting this parameter to true will not work with most SIP clients, as they cannot provide a client certificate.</p> <p>Although most SIP client applications cannot provide certificates, other clients, such as proxy servers can.</p>
NetworkThreadCount	<p>The number of network threads spawned by the SIP container to handle network traffic. The value must be an integer.</p>

Table 3–3 (Cont.) Attributes of the SIP Servlet Container

Attribute	Value
Edge Proxy	<p>The proxy that receives client requests. This proxy adds a pre-loaded route. For example: <code>sip:my.host:5060;transport=tcp</code>. This setting is required for clients to maintain a reusable TCP connection to the server. To ensure this connection, clients may implement a keep-alive algorithm, such as sending periodic CRLFs (carriage-return line feeds).</p> <p>In clustered environments, you must configure an Edge Proxy for each OCMS instance to support communication with the Edge Proxy or other proxy applications. For high availability installations, this attribute must be set to the address of the Edge Proxy or to the virtual hostname and port service used by the load balancer in front of the Edge Proxies. Set this value using the following format:</p> <pre>sip:<EdgeProxy or Load Balancer IP address>:<port>;lr</pre> <p>For more information, see "Configuring the OCMS SIP Containers for High Availability".</p>
RecordRoute	<p>The host, port and transport used in the <i>RecordRoute</i> header of non-distributable applications. Enter this value as a SIP URI in the following format:</p> <pre>sip:host:port;transport=tcp.</pre> <p>You need only enter the hostname (<code>sip:my.host:5060;transport=tcp</code>).</p> <p>For high availability configurations, configure this parameter in the following format:</p> <pre>sip:<SIP Container IP address>:<port>.</pre> <p>Remove transports methods to enable any type of transport to be used between the Edge Proxy and OCMS. For more information, see "Configuring the OCMS SIP Containers for High Availability".</p>
SIPPort	<p>The port through which the SIP Servlet Container accepts traffic. The default value is 5060.</p>
SipServletCommanInterceptors	<p>A comma-separated list of SipServlet interceptor classes. The interceptors must implement <code>org.aopalliance.intercept.MethodInterceptor</code>. If the class has a public constructor with <code>javax.servlet.ServletConfig</code> it will be used to instantiate the interceptor. To enable P-Asserted-Identity support (RFC 3325) for authentication when using OCMS UserService and SecurityService, replace <code>oracle.sdp.sipservletengine.SecurityInterceptor</code> with <code>oracle.sdp.extinterceptors.PaiSecurityInterceptor</code>.</p>
SipServletOc4jInterceptors	<p>A comma-separated list of SipServlet interceptor classes specific to OC4J. The interceptors must implement <code>org.aopalliance.intercept.MethodInterceptor</code>. If the class has a public constructor with <code>javax.servlet.ServletConfig</code> it will be used to instantiate the interceptor. This list will be prepended to the list of interceptors entered for the <i>SipServletCommonInterceptors</i> attribute.</p>
StatisticsPeriodicity	<p>The interval, in minutes, at which the SIP servlet container logs statistics that display in the SIP Servlet Container Monitor. Setting this attribute to 0 suspends logging operation.</p>
TimerListenerOc4jInterceptors	<p>A comma-separated list of OC4J-specific <code>TimerListener</code> interceptor classes. The interceptors must implement <code>org.aopalliance.intercept.MethodInterceptor</code>.</p>
TimerT1	<p>The estimate, in seconds, for a round trip. This value must be an integer.</p>

Table 3–3 (Cont.) Attributes of the SIP Servlet Container

Attribute	Value
TimerT2	The maximum interval, in seconds, for non-INVITE requests and INVITE responses. This value must be an integer.
TimerT4	The maximum interval, in seconds, that a message can remain in the network. This value must be an integer.
TlsKeyStore	The file path to a keystore of Sun's JKS type.
TlsKeyStorePassword	A password or a password indirection. For example, '->jazn.com/oc4jadmin' which means that the password for the JAZN user 'oc4jadmin' in the realm 'jazn.com' is used to unlock the store. A JAZN user is created by invoking the jazn.jar in OC4J.
TlsTrustStore	A file path to a truststore of Sun's JKS type.
TlsTrustStorePassword	A password or a password indirection. For example, '->jazn.com/oc4jadmin' which means that the password for the JAZN user 'oc4jadmin' in the realm 'jazn.com' is used to unlock the store. A JAZN user is created by invoking the jazn.jar in OC4J.
Trusted Hosts	A comma-separated list IP of addresses representing trusted hosts as described in RFC 3325. For example, enter 192.168.0.10, 192.168.0.11. Leaving the value field blank means that there are no trusted hosts; entering an asterisk (*), means that all IP addresses can be trusted. Regular expressions are not supported.
UdptoTcpTriggerSize	The maximum number of bytes contained in a request message sent by the SIP servlet container. When a request message reaches this size, the SIP servlet container uses TCP rather than UDP to transport the request. The default value for a request is 1300 bytes.
UseStun	Select <i>true</i> to use STUNbis keepalive traffic.
UseTCP	Select <i>true</i> for the SIP servlet container to listen for TCP traffic. Oracle recommends setting TCP as the transport protocol because of network fragmentation issues. Add NAPTR and SRV records to the DNS to indicate that TCP is the preferred protocol. Ensure that clients connecting to OCMS fully support NAPTR and SRV records.
UseTLS	Select <i>true</i> to enable the SIP servlet container to provide server certificates during a TLS handshake with a client. If set to <i>true</i> , the SIP Servlet container listens to the TLS port. You must also configure the <i>TlsKeystore</i> , <i>TlsKeyStorePassword</i> , <i>TlsTrustStore</i> , and <i>TlsTrustStorePassword</i> attributes.
UseUDP	Select <i>true</i> for the SIP servlet container to listen for UDP traffic.
Via	The host, port, and transport used in the <i>VIA</i> header of non-distributable applications. Enter this value as a SIP URI.

Setting an Alias for an Application

A SIP application in the OCMS SIP servlet container is basically the parsed `sip.xml` file. Hence, the SIP application contains all of the parsed servlet-definitions as well as the servlet-mappings (that is, the rules) and other `sip.xml` components as listeners.

As illustrated in [Chapter 13, "Deploying Applications"](#), The SIP application is usually built as an Enterprise Archive (EAR) that is deployed to OC4J. Once the SIP application has been deployed, the application server parses the `sip.xml` and instantiates the servlets and listeners. The application server also creates a name for the deployed application by piecing together the components of the EAR file. Because this generated name can be lengthy, you can create an alias for the application. You can use this shorter, more intuitive application name when you configure the default application. This short name can be used in the URI parameter of the route header or the request-URI.

For example, the name of a presence application deployed to the server might appear as *presenceapplicationear-4.0.0-dev/eventnotificationssr-4.0.0-dev*. Using the *ApplicationAliases* attribute, you can set a short name for this application by entering key-value pair such as *presence=presenceapplicationear-4.0.0-dev/eventnotificationssr-4.0.0-dev*. The key is the alias (*presence*) and the value is the real name of the application. This alias can be used for the real name of the application.

Note: The value for set for *ApplicationAliases* attribute must also match the *appId* parameter of the [Application Router](#)'s *Slipperiest* attribute to ensure that the OCMS SIP servlet container invokes the applications that are appropriate to incoming requests. The *appId* parameter is case-sensitive.

Using TLS

The *useTLS* attribute, when set to *true*, enables the SIP servlet container to act as a TLS server by providing a server certificate during the TLS handshake. You must also configure the *TlsKeystore*, *TlsKeyStorePassword*, *TlsTrustStore*, and *TlsTrustStorePassword* attributes. In addition to providing the server certificate, the *needClientAuth* attribute enables the SIP servlet container to perform mutual TLS by requiring a client certificate during the TLS handshake.

Setting the Keystore

A keystore is a database of public and private keys that can be grouped into two categories: key entries and trusted certificate entries. The SIP Servlet container works with keystores of Sun's JKS type (PKCS12 is not supported). A keystore of the JKS type may contain both key entries and trusted certificate entries. Using two different files instead of one single keystore file provides for a cleaner separation between the SIP Servlet container's certificates and certificates from other entities (which is also supported by the SIP Servlet container).

Note: Multiple private key entries in the keystore are not supported; the SIP Servlet container supports one certificate for each domain. There is no mechanism to feed the container with a password for a keystore entry.

Enabling TLS

To enable TLS, you must set the following java system properties: *useTLS*, and, optionally, *needClientAuth*. You must also configure the *TlsKeystore*, *TlsKeyStorePassword*, *TlsTrustStore*, and *TlsTrustStorePassword* attributes.

Setting the *useTLS* attribute means that the container will listen to the configured TLS port and provide the server certificate during the TLS handshake.

If you set the *needClientAuth* attribute, the server will as usual provide it's server certificate but will also require a certificate from the client. This is a setting that will not work with Oracle Communicator (and other SIP clients) as SIP clients typically cannot provide a client certificate. In an environment where the client is a proxy server this setting can be used.

SIP Servlet Container Logging

For information on the logging system and configuring log levels, see [Chapter 12, "Configuring the Logging System"](#).

STUN Service

The OCMS STUN Service implements STUN -- Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). As described in RFC 3489, STUN enables STUN clients behind a NAT (that is, clients behind a router) to discover the presence of a NAT, the type of NAT, and then to learn the address bindings (including IP addresses) allocated by the NAT.

STUN is a client-server protocol in which a STUN client sends a request (a Binding Request) to a server, which in turn sends a response. OCMS supports the receipt of Binding Requests from a client, which are sent over UDP and are used to both discover the presence of a NAT and discover the public IP address and the port mappings that it generates. When a STUN client sends a Binding Request to the STUN server, the STUN Server examines the request's source IP address and port and copies them into a response that it sends back to the client. When the STUN client receives the Binding Response, it compares the IP address and port in the packet with the local IP address and port to which it bound itself when it sent the Binding Request to the STUN Server.

The attributes of the STUN Service MBean (described in [Table 3-4](#)) enable you to set the STUN Server's primary and secondary IP addresses and ports that form the four RFC 3489-dictated address-port combinations used by the STUN server to receive client Binding Requests. Per RFC 3489, the combinations are as follows:

- A1, P1 -- The Primary Address and Primary Port
- A2, P1 -- The Secondary Address and the Primary Port
- A1, P2 -- The Primary Address and the Secondary Port
- A2, P2 -- The Secondary Address and the Secondary Port

Typically, the STUN server's Primary Port (P1) is set to UDP port 3478. The Stun server uses the Secondary Address and Secondary Port values (A2, P2) in the CHANGED-ADDRESS attribute included in its Binding Response.

Table 3-4 Attributes of the STUNService MBean

Attribute	Value
Autostart	Set to <i>true</i> for the Stun Server to start automatically when OCMS starts.
PrimaryAddress	The primary STUN address on which to listen for incoming Binding Requests. The default value is 127.0.0.1.
PrimaryPort	The primary STUN port on which to listen for incoming Binding Requests. The value is UDP port 3478, the default STUN Port as described in RFC 3489.
SecondaryAddress	The secondary STUN address on which to listen for incoming Binding Requests. This cannot be the same value as <i>PrimaryAddress</i> .
SecondaryPort	The secondary STUN port to which to listen for incoming Binding Requests. The default value is UDP port 3479.

Configuring SIP Applications

This section describes the configuration enabled by the MBeans registered to the following SIP applications:

- [Subscriber Data Services](#)
- Presence (See "[Configuring Presence and Presence Web Services](#)")
- [Proxy Registrar](#)
- [Application Router](#)
- [Aggregation Proxy](#)

Subscriber Data Services

Subscriber Data Services (`subscriberdataservices`) is the parent application to all SIP applications that require authentication and security against the OCMS user repository. For example, the [Application Router](#), [Aggregation Proxy](#), and [Proxy Registrar](#) require Subscriber Data Services. In the case of the latter, the Location Service and the registrar component of the Proxy Registrar are dependent upon Subscriber Data Services. Subscriber Data Services also provides access to the Oracle Internet Directory (OID). For more information, see "[Overview of Security](#)". See also "[Configuring Oracle Internet Directory as the User Repository](#)" for information on using Oracle Internet Directory (OID), the LDAP data store used by Oracle WebCenter Suite, as the user provisioning repository for an OCMS deployment.

Account Security

Subscriber Data Services provides account security through the Account Lockout Service and Login Failure service MBean groups.

The Account Lockout group includes the following MBeans:

- AA Service

The AA Service MBean is used by the Login Failure Service to lock accounts. This MBean depends on the MathFunction Model MBeans, which enable the AA Service MBean to calculate the next lock duration for an account based on the current number of failed login attempts.

Note: Account locking persists when you restart OC4J.

[Table 3–5](#) describes the attributes of the AA Service MBean.

Table 3–5 *Attributes of the AA Service MBean*

Attribute	Value
MathFunction	The type of math function used to calculate the next lock duration for an account based on the number of current failed login attempts. The math functions, which include <code>Linear</code> , <code>Exponential</code> and <code>Constant</code> , are packaged as ModelMBeans. The default value is <code>Linear</code> .
DefaultLockDuration	The lock duration, in seconds, to use if no math functions are available. The default value is 600.
JNDIName	The JNDI Name of the AA Service. This value is read-only
SecurityServiceName	The JNDI Name bound to the service object that is used to unlock user accounts. This value is read-only.

Table 3–6 Subscriber Data Services MBeans

MBean	Tasks
ModelMBeanDeployer	The helper MBean used by the Subscriber Data Services application to deploy its Model MBeans.

- **Constant (read-only)**
The constant function used by the MathFunction MBean to calculate the next lock duration for an account based on the number of current failed login attempts.
- **Exponential**
The exponential math function used by the MathFunction MBean to calculate the next lock duration for an account based on the number of current failed login attempts.
- **Linear**
The linear math function used by the MathFunction MBean to calculate the next lock duration for an account based on the number of current failed login attempts.

Of the Subscriber Data Services MBeans, only AA Service and Command Service can be configured.

CommandService

The operations of the CommandService MBean enable you to execute the equivalent of Sapphire Shell (Sash) commands which are used to provision OCMS users to the Oracle database. For example, to view a list of commands for account management:

1. Click the *Operations* tab. A list of `get` commands appears.
2. Click **help** for a returning a help String for a partial command. The parameters for the `help` operation appear.
3. Enter *account* in the *Value* field.

Tip: Click **Use Multiline Editor** to expand the *Value* field to accommodate long command names.

4. Click **Invoke Operation**. The commands pertaining to account management appear (Figure 3–6). These commands match those retrieved by entering `help account` at the Sash prompt. For more information, see "Viewing Subcommands" in Chapter 11, "Provisioning Users with Sash".

Figure 3–6 Viewing Help for a Command

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: as10132 > OC4J: ocms > Application: subscriberdataservices > Application MBeans

Information
Operation executed successfully.

Operation: help

Return Invoke Operation

MBean Name: `subscriberdataservices:service=CommandService,name=CommandService,SIPApplication=subscriberdataservices,type=CommandService`
Description: Returns a help String for the partial command.
Return Type: `java.lang.String`

Parameters

Name	Description	Type	Value
command	The Partial command for which to display help.	java.lang.String	account

Return Value

*** Description ***
Contains commands for management of user accounts. In an account you can set if the account is active, locked or if it perhaps should be a temporarily account.
Aliases: [no aliases]
Syntax:
account

To view a list of all of the available commands (Figure 3–7), select `listAllCommands` from the *Operations* tab and then click **Invoke Operation** from the `listAllCommands` page that appears. For a description of Sash commands, see "Viewing Available Commands". For more information on user management through Sash, see "Creating a User".

Figure 3–7 Viewing Available Commands

ORACLE Enterprise Manager 10g
Application Server Control

OC4J: home > Application: sipcore > Application MBeans >

Information
Operation executed successfully.

Operation: listAllCommands

Return Invoke Operation

MBean Name: `sipcore:service=CommandService,name=CommandService,SIPApplication=sipcoreapplication`
Description: Operation exposed for management
Return Type: `java.lang.String`

Return Value

```

account add
account delete
account update
account info
credentials add
credentials delete
credentials deleteall
credentials update
credentials list
role system list
role system add
role system update
role system delete
role user add
role user delete
role user list
user add
user delete
    
```

The Command Service MBean is deployed within the presence application to enable user provisioning to the XDMS (XML Document Management Server). See [Command Service \(XDMS Provisioning\)](#).

Proxy Registrar

The Proxy Registrar is a user agent server (UAS) that implements the proxy and registrar functions described in RFC 3261. This SIP entity is a router of messages. The Proxy Registrar's registrar function processes the REGISTER requests from User Agent clients and uses a Location Service to store a binding (that is, an association) between a user's address of record (AOR) and the user's SIP or SIPS URIs that are located in a CONTACT field. Upon receiving requests to the AOR, the proxy function locates the mapped URIs through a Location Service lookup and then proxies the request using the location information retrieved by this lookup. [Table 3-7](#) describes the attributes of the Proxy Registrar.

Table 3-7 Attributes of the Proxy Registrar

Attributes	Description
CurrentRegDevices	A read-only attribute that displays the number of currently registered devices.
DefaultExpires	Sets the expiration value for the REGISTER request if the client has not indicated a preferred value itself. The default value for this attribute is 3600 seconds.
MaxExpires	Sets the maximum expiration value for the REGISTER request accepted by the server. Although a client can request any expiration value in the REGISTER request, the server can set a maximum amount of time that it accepts for expiration. If the client requests a time greater than the value set for <i>MaxExpires</i> , then the server sets the expiration time for that particular REGISTER request to the value set for <i>MaxExpires</i> . The default value for this attribute is 7200 seconds.
MinExpires	Specifies the minimum expiration value for a REGISTER request accepted by server. While clients can request any expiration time, they can also specify a very low value for the expiration of the REGISTER request. Such low values require clients to update registration information frequently, which creates traffic on the network. If a client requests a value that is below this minimum expiration time, then the server does not accept the REGISTER request and responds with a 423 (<i>Interval Too Brief</i>) error response per RFC 3261. This response message specifies the lowest expiration time allowed, which is set by the <i>MinExpires</i> attribute. The server is allowed to shorten an expiration time, but can never lengthen one. The default value for this attribute is 60 seconds.
SipRegAllowThirdParty	Specifies whether the Proxy Registrar allows third-party registrations. In a third-party registration, the entity issuing the request (in the <i>From</i> header) is different from the entity being registered (in the <i>To</i> header) to whom the provided Contact information applies. If set to true, the Proxy Registrar allows third party registrations. If set to false (the default value), then third-party registrations are rejected (the requestor receives a 403 <i>Forbidden</i> status code). This is a read-only attribute that is always set to <i>false</i> .
SipRegMaxUsers	A read-only attribute that specifies the maximum number of users supported by the Proxy Registrar.

Application Router

When the OCMS SIP servlet container receives incoming requests that do not contain the application ID (`appId`) parameter, the container invokes the Application Router which inserts routing information that includes this parameter into the request's `ROUTE`. As a result, the container can respond to incoming `INVITE`, `MESSAGE`, `PUBLISH`, `REGISTER`, and `SUBSCRIBE` requests by invoking the appropriate applications in the correct order. As described in *Oracle Communication and Mobility Server Developer's Guide*, the `appId` parameter is an OCMS extension to JSR-116 used to route requests to applications.

Because requests generated by SIP clients other than Oracle Communicator typically do not contain the `appId` parameter in either their `ROUTE` or `REQUEST URI` headers, the presence of the `appId` parameter in an incoming request's `ROUTE` header ensures that the OCMS SIP servlet container correctly handles requests. The Application Router MBean (`ocmsrouteloaderear`, illustrated in [Figure 3-8](#)) enables you to configure the destinations for incoming `INVITE`, `MESSAGE`, `PUBLISH`, `REGISTER`, and `SUBSCRIBE` requests by defining the application routing sequence and other route-related criteria.

Figure 3-8 Setting the Application Routing for an `INVITE` Request

The screenshot shows the Oracle Enterprise Manager 10g Application Server Control interface. The breadcrumb trail is: OC4J: home > Application: ocmsrouteloaderear > Application MBeans. The page title is "Application MBeans" and it includes a search bar and a "Find" button. The main content area displays the configuration for the MBean `ocmsrouteloaderear:name=RouteLoader_INVITE,type=RouteLoader`. The description is "Settings for handling INVITE requests". The persist policy is `OnUpdate` and the persist location is `routeloader`. A table titled "Attributes (4)" lists the following attributes:

Name	Description	Access	Value
<code>IncrementalMode</code>	Push routes in a similar way as a application dispatcher. This means pushing one application route and one app dispatcher route (route loader route in this case) until the request uri is changed. If set to false all configured routes will be pushed once an initial request arrives. WARNING: changing this setting in runtime may lead to unpredicted results for ongoing traffic.	RW	false
<code>RecordRoute</code>	Set Record-Route header field	RW	false
<code>RouteLoaderUri</code>	Back route of the RouteLoader. This URI is only used in incremental mode (route loader route).	RW	sip:127.0.0.1;lr;
<code>SipUriList</code>	Comma separated list of SIP URIs to push in Route header. First URI in list will be top route.	RW	sip:127.0.0.1:5060;transport=TCP;lr;

[Table 3-8](#) describes the attributes that you configure for the routing of `INVITE`, `MESSAGE`, `PUBLISH`, `REGISTER`, and, `SUBSCRIBE` requests.

Table 3–8 Attributes of the Application Router

Attribute	Description
IncrementalMode	<p>This boolean enables you to select the Application Router’s execution mode: <i>true</i> for incremental, <i>false</i> for standard.</p> <ul style="list-style-type: none"> ■ Select <i>true</i> to set the incremental execution mode for the Application Router. For the incremental mode, the Application Router inserts the first (or top-most) URI configured in the <i>SIPUriList</i> attribute as well as the URI of the return route to the Application Router itself, which is defined as the value to the <i>RouteLoaderUri</i> attribute. When the request returns to the Application Router, the Application Router then checks if the Request URI has changed. If so, it proxies the request to this new URI. If the Request URI remains unchanged, then the Application router inserts the next URI defined in the <i>SIPUriList</i> into the request’s ROUTE header and repeats the cycle. <p>You must configure the following Application Router attributes for the incremental mode:</p> <ul style="list-style-type: none"> ■ RecordRoute ■ RouteLoaderUri ■ SIPUriList <ul style="list-style-type: none"> ■ Select <i>false</i> to set the standard mode for the Application Router. In the standard mode, the Application Router inserts all of the routes defined in the <i>SipUriList</i> attribute in the request’s ROUTE header. The request then follows the routes in the sequence that they are defined in the <i>SipUriList</i> attribute. <p>You must configure the following Application Router attributes for the standard mode:</p> <ul style="list-style-type: none"> ■ RecordRoute ■ RouteLoaderUri ■ SipUriList
RecordRoute	Set to <i>true</i> to enable record routing.
RouteLoaderUri	The URI of the return route to the Application Router. This value must be the same URI as the SIP servlet container. In general, you set the value as the URI of the Proxy Registrar because the SIP servlet container recognizes the URI of the Proxy Registrar as its own.
SipUriList	<p>A comma-separated list of URIs, transport methods, and <code>appIDs</code> of applications that the Application Router inserts in the ROUTE header of an incoming request.</p> <p>The default list, which routes requests to the Proxy Registrar, is defined as</p> <pre> sip:<SIP Container IP Address>:<SIP port>;transport=TCP;lr;appId=proxyregistrar </pre> <p>For example:</p> <pre> sip:144.25.174.189:5060;transport=TCP;lr;appId=proxyregistrar </pre> <p>Request routing is set according to the order of the applications listed in this attribute. For example, to call an application called <i>dialin</i> as the first destination for an INVITE request, insert the information for <i>dialin</i> as the first item on the list as follows:</p> <pre> sip:144.25.174.189:5060;transport=TCP;lr;appId=dialin,sip:144.25.174.189:5060; transport=TCP;lr;appId=proxyregistrar </pre> <p>To ensure that request-related applications that you deploy to the OCMS SIP servlet container are invoked in response to incoming requests, you must add any application that you deploy to this list. The name of the application defined for the <code>appId</code> parameter (which is case-sensitive) can either be the full name of the deployed application, or an alias for the application. If the latter, the name must match the application alias set for the <i>ApplicationAliases</i> parameter of the SIP Servlet Container MBean. The Proxy Registrar must always be the last item listed in the <i>SIPUriList</i> attribute. [[See Deploying SIP Applications]]</p> <p>For more information on the <code>appID</code> parameter, refer to <i>Oracle Communication and Mobility Server Developer’s Guide</i>.</p>

Setting and Viewing the SIP Port

The SIP port is set in the opmn.xml file rather than through Enterprise Manager. You should specify a specific port number, such as 5060, rather than a range of ports.

You can view what SIP port you have set in the opmn.xml from the "Server Properties" page in Enterprise Manager. Do not change the SIP port through this page. If you attempt to change it the "protocol" part in the opmn.xml file will be modified from "sip" to "ajp" and the SIP container will start on the default port, 5060.

Configuring Security and Login Modules

This chapter describes how to set the security provider for an application. This chapter includes the following topics:

- ["Overview of Security"](#)
- ["Configuring Subscriber Data Services"](#)
- ["Configuring Applications to Use Login Modules"](#)
- ["Security in SIP Servlets"](#)
- ["Authentication Using the P-Asserted Identity Header"](#)
- ["Authentication of Web Service Calls and XCAP Traffic"](#)
- ["Configuring Oracle Internet Directory as the User Repository"](#)

Overview of Security

OCMS implements both basic (HTTP) authentication and digest authentication as described in RFCs 3261 and 2617. In OCMS, both SIP and HTTP applications can be configured to authenticate against a RADIUS authentication system or Oracle Internet Directory (OID) version 10.1.4.0.1.

The [Subscriber Data Services](#) application provides the infrastructure for authentication and authorization. This application, which is parent to applications requiring authentication, enables its child applications to access the authentication backend (RADIUS or Oracle Internet Directory) by means of native Enterprise JavaBeans (EJBs). Subscriber Data Services provides a framework that enables security but does not impose security constraints itself; this is done instead by its child applications, such as [Proxy Registrar](#), which have authority constraints defined by the `<security-constraint>` element in `sip.xml`. For more information on configuring security in the deployment descriptor file, see ["Security in SIP Servlets"](#).

Each of the child applications of Subscriber Data Services must be configured to use a JAAS (Java Authentication and Authorization Service) login module for authentication and authorization against the configured user repository.

The OCMS JAAS-Compliant Login Modules

OCMS leverages the pluggable architecture of JAAS by providing the RADIUS Login Module that checks user credentials stored in the RADIUS user repositories for SIP or HTTP applications.

The RADIUS Login Module provides digest authentication for SIP applications as described in RFC 4590 and basic authentication HTTP applications for users provisioned to a RADIUS database.

The RADIUS database only authenticates users; it does not authorize users. Therefore, if you use the RADIUS Login Module, you must create users and assign the appropriate roles in the Oracle database that correspond to each user provisioned to the RADIUS database for role-based authorization. When deleting a user's authentication information from a RADIUS database, you must also delete that user's authorization information (for example, an account and role-related information) manually from the Oracle database. This prevents a future user provisioned with the same user name in RADIUS from inheriting the authorization and account-specific information of a user who had previously been deleted.

Note: Due to licensing restrictions, open source RADIUS is not packaged with OCMS during installation. Prior to configuring the RADIUS Login Module with OCMS, the JRadius client library must be manually downloaded and installed. The necessary JRadius client library can be downloaded from the Sourceforge project at:

`http://jradius-client.sourceforge.net/`

Once the JRadius client library has been downloaded, manually copy the `jradius-client.jar` file to `$ORACLE_HOME/j2ee/home/lib/ext` on Oracle Containers for J2EE (OC4J) deployments. You must then restart the application server to deploy the RADIUS Login Module.

These pluggable JAAS login modules enable the SIP servlet container to perform authentication and authorization against external databases for User Agents sending SIP requests. The modules implement user authentication against the RADIUS user repositories by first invoking the `JAAS LoginModule` class and then by authorizing a previously authenticated user by verifying that the appropriate access permissions have been granted.

Note: Because these login modules authenticate users against external repositories, they are considered custom security providers in OC4J. See *Oracle Containers for J2EE Security Guide* for information on configuring a custom security provider within a J2EE application.

Application Type and Authentication Mode

The type of authentication depends on the protocol used by the login module. As noted in [Table 4-1](#), Digest authentication is used on the SIP side and Basic authentication is used on the HTTP side.

Note: Although the [Aggregation Proxy](#) is a Web application, it is not limited to basic authentication because it explicitly performs authentication using internal security APIs. As a result, the Aggregation Proxy's default security settings (which set the authentication method as digest) will work without further configuration.

Table 4–1 Authentication Based on Protocol

Protocol	Authentication Mode
SIP	Digest
HTTP	Basic

Note: This chapter describes how to specify the login module used by an application and how to configure the OCMS and RADIUS login modules themselves. See "[Configuring Oracle Internet Directory as the User Repository](#)" for information on using Oracle Internet Directory (OID), the LDAP data store used by Oracle WebCenter Suite, as the user provisioning repository for an OCMS deployment.

Configuring Subscriber Data Services

Subscriber Data Services (subscriberdataservices) is the parent application to all SIP applications that require authentication and security against the OCMS user repository. For example, the [Application Router](#), [Aggregation Proxy](#), and [Proxy Registrar](#) require Subscriber Data Services. In the case of the latter, the Location Service and the registrar component of the Proxy Registrar are dependent upon Subscriber Data Services. Subscriber Data Services also provides access to Oracle Internet Directory (OID). For more information, see "[Overview of Security](#)". See also "[Configuring Oracle Internet Directory as the User Repository](#)" for information on using Oracle Internet Directory (OID) as the user provisioning repository for an OCMS deployment.

Account Security

Subscriber Data Services provides account security through the Account Lockout Service and Login Failure service MBean groups.

The Account Lockout group includes the following MBeans:

- AA Service

The AA Service MBean is used by the Login Failure Service to lock accounts. This MBean depends on the MathFunction Model MBeans, which enable the AA Service MBean to calculate the next lock duration for an account based on the current number of failed login attempts.

Note: Account locking persists when you restart OC4J.

[Table 4–2](#) describes the attributes of the AA Service MBean.

Table 4–2 Attributes of the AA Service MBean

Attribute	Value
MathFunction	The type of math function used to calculate the next lock duration for an account based on the number of current failed login attempts. The math functions, which include <code>Linear</code> , <code>Exponential</code> and <code>Constant</code> , are packaged as ModelMBeans. The default value is <code>hotisp.math:service=MathFunction,name=Linear</code> .

Table 4–2 (Cont.) Attributes of the AA Service MBean

Attribute	Value
DefaultLockDuration	The lock duration, in seconds, to use if no math functions are available. The default value is 600.
JNDIName	The JNDI Name of the AA Service. This value is read-only
SecurityServiceName	The JNDI Name bound to the service object that is used to unlock user accounts. This value is read-only.

Table 4–3 Subscriber Data Services MBeans

MBean	Tasks
ModelMBeanDeployer	The helper MBean used by the Subscriber Data Services application to deploy its Model MBeans.

- Constant (read-only)
The constant function used by the MathFunction MBean to calculate the next lock duration for an account based on the number of current failed login attempts.
- Exponential
The exponential math function used by the MathFunction MBean to calculate the next lock duration for an account based on the number of current failed login attempts.
- Linear
The linear math function used by the MathFunction MBean to calculate the next lock duration for an account based on the number of current failed login attempts.

Of the Subscriber Data Services MBeans, only AA Service and Command Service can be configured.

CommandService

The operations of the CommandService MBean enable you to execute the equivalent of Sash commands which are used to provision OCMS users to the Oracle database. For example, to view a list of commands for account management:

1. Click the *Operations* tab. A list of `get` commands appears.
2. Click **help** for a returning a help String for a partial command. The parameters for the `help` operation appear.
3. Enter *account* in the *Value* field.

Tip: Click **Use Multiline Editor** to expand the *Value* field to accommodate long command names.

4. Click **Invoke Operation**. The commands pertaining to account management appear ([Figure 4–1](#)). These commands match those retrieved by entering `help account` at the Sash prompt. For more information, see "[Viewing Subcommands](#)" in [Chapter 11, "Provisioning Users with Sash"](#).

Figure 4–1 Viewing Help for a Command

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: as10132 > OC4J: ocms > Application: subscriberdataservices > Application MBeans

Information
Operation executed successfully.

Operation: help

Return Invoke Operation

MBeanName: `subscriberdataservices:service=CommandService,name=CommandService,SIPApplication=subscriberdataservices,type=CommandService`
Description: Returns a help String for the partial command.
Return Type: `java.lang.String`

Parameters

Name	Description	Type	Value
command	The Partial command for which to display help.	java.lang.String	account

Return Value

*** Description ***
Contains commands for management of user accounts. In an account you can set if the account is active, locked or if it perhaps should be a temporarily account.
Aliases: [no aliases]
Syntax:
account

To view a list of all of the available commands (Figure 4–2), select `listAllCommands` from the *Operations* tab and then click **Invoke Operation** from the `listAllCommands` page that appears. For a description of Sash commands, see "Viewing Available Commands". For more information on user management through Sash, see "Creating a User".

Figure 4–2 Viewing Available Commands

ORACLE Enterprise Manager 10g
Application Server Control

OC4J: home > Application: sipcore > Application MBeans >

Information
Operation executed successfully.

Operation: listAllCommands

Return Invoke Operation

MBean Name: `sipcore:service=CommandService,name=CommandService,SIPApplication=sipcoreapplication`
Description: Operation exposed for management
Return Type: `java.lang.String`

Return Value

```
account add
account delete
account update
account info
credentials add
credentials delete
credentials deleteall
credentials update
credentials list
role system list
role system add
role system update
role system delete
role user add
role user delete
role user list
user add
user delete
```

The Command Service MBean is deployed within the presence application to enable user provisioning to the XDMS (XML Document Management Server). See [Command Service \(XDMS Provisioning\)](#).

Configuring Applications to Use Login Modules

You configure security for SIP applications by first defining the `<security-constraint>` element in the deployment descriptor file, `sip.xml`, setting the security provider (login module) appropriate to the authenticating user repository used by the application and then by configuring the login module itself. For more information on setting security in `sip.xml`, see "Security in SIP Servlets".

You can configure a login module for each application that you deploy. During the deployment process, you can configure the login module for SIP or HTTP applications using the *Security Provider* task in the *Deployment Settings* page of the Application Server Control deployment wizard. Once you deploy an application, you can examine or edit its login module's configuration. From the *Edit Login Module* page, accessed from the *Security Providers* task in the *Administration* page ([Figure 4-3](#)).

Figure 4-3 *Modifying a Login Module.*

ORACLE Enterprise Manager 10g
Application Server Control

OC4J: home > Application: proxyregistrar > Security Provider >

Edit Login Module

Page Refreshed Jan 29, 2007 2:38:56 PM PST

JAAS Login Module Class **oracle.sdp.ocmsloginmodule.OCMSLoginModule**

Login Module Control Flag

Configuration Properties

Name	Value	Delete
failurePeriod	600	
maxFailure	3	
useUTF8	true	
authMethod	Digest	
LoginFailureServiceName	LoginFailureService	

For more information, see *Deploying with Application Server Control Console* in *Oracle Containers for J2EE Deployment Guide*

Configuring Login Modules through system-jazn-data.xml and orion-application.xml

Alternatively, login modules can also be configured through the `<jazn-loginconfig>` settings either in the `system-jazn-data.xml` file or in the `orion-application.xml` file located in an application's EAR (Enterprise Archive) file.

Configuring Login Modules in system-jazn-data.xml

The `system-jazn-data.xml` file is the repository for login module configuration. The settings in this file are updated when you administer login modules through Oracle Application Server Control ([Figure 4-3](#)) or through the OracleAS JAAS

Provider Admintool. For more information, see *Oracle Containers for J2EE Security Guide*.

[Table 4-4](#) describes the options supported by the OCMS Login Module and the RADIUS Login Module that you configure in `system-jazn-data.xml`.

Table 4-4 Login Module Options

Option	Description
useUTF8	If set to <code>true</code> , then the login module supports user names and passwords encoded in the UTF-8 character set.
authMethod	For SIP applications, select either <code>Basic</code> or <code>Digest</code> . For HTTP applications, select <code>Basic</code> .
failurePeriod	The time, in seconds, that an account is locked. The value entered in <code>orion-application.xml</code> takes precedence over the value entered in the <code>DefaultLockDuration</code> attribute of the AA Service MBean. See also " SIP Servlet Container ".
maxfailure	The number of failed login attempts before an account is blocked.

Declaring the OCMS Login Module in `orion-application.xml`

The `<jazn-loginconfig>` element in `orion-application.xml` defines the login modules used by an application. The configurations you set in this file are populated to `system-jazn-data.xml`. [Example 4-1](#) illustrates configuring login modules for authentication against a RADIUS authentication system.

Declaring the RADIUS Login Module in `orion-application.xml`

[Example 4-1](#) defines a login module for the Proxy Registrar application. This application, however, requires users to be authenticated against the RADIUS database. In addition to the options described in [Table 4-4](#), the RADIUS Login Module also supports the options described in [Table 4-5](#).

Table 4-5 RADIUS Login Module Options

Option	Description
hostname	The host name or IP address of the remote RADIUS server.
authPort	The destination port for authentication requests.
acctPort	The destination port for the accounting server.
sharedSecret	A <code>String</code> known only to the RADIUS server and the RADIUS client.

Example 4-1 Declaring the Radius Login Module in `orion-application.xml`

```
<jazn-loginconfig>
  <application Key="name">
    <name>proxyregistrar</name>
    <login-modules>
      <login-module Key="class control-flag">
        <class>
          oracle.sdp.radiusloginmodule.RadiusLoginModule
        </class>
      </login-module>
    </login-modules>
  </application>
</jazn-loginconfig>
```

```

</class>
<control-flag>required</control-flag>
<options>
  <option>
    <name>useUTF8</name>
    <value>true</value>
  </option>
  <option>
    <name>authMethod</name>
    <value>Digest</value>
  </option>
  <option>
    <name>failurePeriod</name>
    <value>600</value>
  </option>
  <option>
    <name>maxFailure</name>
    <value>5</value>
  </option>
  <option>
    <name>hostName</name>
    <value>127.0.0.1</value>
  </option>
  <option>
    <name>authPort</name>
    <value>1812</value>
  </option>
  <option>
    <name>acctPort</name>
    <value>1813</value>
  </option>
  <option>
    <name>sharedSecret</name>
    <value>secret</value>
  </option>
  <option>
    <name>radiusClientClass</name>
    <value>oracle.sdp.radiusloginmodule.JRadiusClient</value>
  </option>
</options>
</login-module>
</login-modules>
</application>
</jazzn-loginconfig>

```

Security in SIP Servlets

OCMS supports declarative and programmatic security for SIP servlets as described in the SIP Servlet API.

Declarative Security

The SIP Servlet API describes declarative security as expressing an application's security structure that includes roles, access control, and authentication requirements in a form that is external to the application¹. The deployment descriptor, `sip.xml`, is

¹ SIP Servlet API, Version 1.0

the vehicle for declarative security. Developers define how security should be effected in a deployed application by defining the `<security-constraint>` element. This element, which is described in detail in *Oracle Communication and Mobility Server Developer's Guide*, includes the following child elements (described in [Table 4-6](#)).

In OCMS, you first configure security for SIP applications by defining these elements. You can also set the login module used by the application in the `system-jazn-data.xml` file or in the `orion-application.xml` file as described in ["Configuring Applications to Use Login Modules"](#).

Note: You can prevent a SIP application from performing authentication by removing the constraints defined in `sip.xml` and then by redeploying the application.

Table 4-6 Child Elements of the `<security-constraint>` Element

Element	Description
<code><proxy-authentication></code>	If this element is present in <code>SIP.xml</code> , the container must challenge the user agent with a 407 (<i>Proxy Authentication Required</i>) response status code when authenticating an incoming request or return a 401 response (<i>Unauthorized</i>).
<code><resource-collection></code>	A set of servlets and SIP methods. This element describes the servlet that requires authentication and the SIP methods used for authentication.
<code><auth-constraint></code>	Indicates the user roles that are permitted access to this resource collection.
<code><role-name></code>	The name of a security role.

Programmatic Security

Programmatic security describes the security model from inside a servlet using the `SipServletMessage` methods `getRemoteUser`, `isUserInRole`, and `getUserPrincipal`.

Authentication Using the P-Asserted Identity Header

The SIP Servlet API 1.0 states that in addition to basic and digest authentication, a User Agent authenticates users through the P-Asserted Identity, a SIP header field that conveys the identity of an authenticated user between the nodes of a trusted domain. As described in RFC 3325, a proxy within a trusted domain can receive messages from both trusted and non-trusted nodes alike. In the case of the latter, the proxy authenticates the originator of the message using digest authentication. The proxy then creates the P-Identity Asserted header field from the identity that it derived from authentication and places this field into the message header which it passes to other entities. For example, an inbound proxy server authenticates a user and then inserts the P-Asserted Identity header field into the received SIP message. By inserting the P-Asserted Identity header field, other servers within the trusted domain (such as the Presence Server) do not have to perform authentication again.

OCMS supports the trusted domain identity assertion described in RFC 3325 through the `SipServletCommandInterceptors` attribute of its [SIP Servlet Container](#) Mbean.

Authentication of Web Service Calls and XCAP Traffic

OCMS supports Presence in Oracle Web Center 11g by allowing the Web Center system to manage their community memberships and presence authorization via APIs provided by OCMS. Web Center uses a community proxy user to represent a community to gain access to community members' presence information.

Web Services calls are authenticated by WS-Security. WS-Security is enabled by default to authenticate Web Service applications.

Default Role for All Users

OCMS 10.1.3.4 enables you to define a default role for all users in the JAAS (Java Authentication and Authorization Service) login modules. For example, you can define the role, PUBLIC, for all users by defining an `<option>` element in the `system-jazn-data.xml` file or in the `orion-application.xml` file located in an application's EAR (Enterprise Archive) file as illustrated in [Example 4-2](#):

Example 4-2 Defining a Default Role for All Users

```
<option>
  <name>defaultRole</name>
  <value>{{PUBLIC}}</value>
</option>
```

Alternatively, you can configure security for SIP applications by first defining the `<auth-constraint>` subelement of `<security-constraint>` in the deployment descriptor file, `sip.xml`, and then by setting the security provider (login module) appropriate to the authenticating user repository used by the application and then by configuring the login module itself through the MBean browser.

Configuring Oracle Internet Directory as the User Repository

This section describes how to configure Oracle Internet Directory (OID), as the user provisioning repository for an OCMS deployment. This appendix includes the following sections:

- ["Overview of Configuration for OID Support"](#)
- ["Configuring the OID LDAP Backend"](#)
- ["Repackaging Subscriber Data Services"](#)
- ["Provisioning OCMS Users to OID"](#)

Overview of Configuration for OID Support

For OCMS to support authentication and authorization services for users provisioned to OID requires the following configuration for both OCMS and OID:

- ["Configuring the OID LDAP Backend"](#)
- ["Installing OCMS Components into the OID LDAP Tree"](#)
- ["Repackaging Subscriber Data Services"](#)
- ["Provisioning OCMS Users to OID"](#)

Prerequisites for OID Support

Using the OID data store requires the following:

- A properly installed and configured instance of OCMS.
- An instance of OID, Version 10.1.4.0.1.
- For dynamic verifiers you must enable reversible password encryption for the LDAP realms employed for user authentication by selecting *Userpassword Reversible Encryption*. This configuration is not required for static verifiers. For more information, see *Oracle Internet Directory Administrator's Guide*.

Note: OID is only supported for standalone OCMS deployments and OCMS deployments on Version 10.1.3.2 or higher of Oracle Application Server. See *Oracle Communication and Mobility Server Installation Guide* for further hardware and software requirements and installation options.

Configuring the OID LDAP Backend

You must also configure the following OID LDAP attributes for the OID LDAP backend:

- *orclcommonnicknameattribute* (See "[Mapping JAAS Usernames to LDAP User Entries](#)".)
- *orclsubscriberricknameattribute* (See "[Mapping JAAS Realms to LDAP Subscribers](#)".)
- *orclcommonnamingattribute* (See "[Mapping JAAS Roles to LDAP Groups](#)".)

Mapping JAAS Usernames to LDAP User Entries

JAAS (Java Authentication and Authorization Service) user names are mapped to LDAP Users based on value of the *orclcommonnicknameattribute* under the node *cn=Common*, *cn=Products*, *cn=OracleContext* for each of the provisioned LDAP realms. For example, setting this attribute to *uid* for a given realm implies that SIP or Web users authenticating against OID must provide their corresponding LDAP UID as their username during authentication.

Mapping JAAS Realms to LDAP Subscribers

JAAS realms are mapped to LDAP Realm entries based on the value given to *orclsubscriberricknameattribute* under the root *cn=Common*, *cn=Products*, *cn=OracleContext* node for an OID deployment. For example, setting the value of *orclsubscriberricknameattribute* to *o* for an OID deployment implies that SIP or Web users authenticating against OID must belong to the JAAS realm identified by the value of the *o* attribute. As a result, user *sip.user@company.com* is challenged under the realm, *company*. The mapping of SIP domains to JAAS realms is exposed through the *SipServletContainer's DomainsAndRealms* attribute. In this example, the SIP domain, *company.com*, is mapped to the JAAS realm, *company*. The JAAS realm, *company*, is then mapped to the LDAP Subscriber for whom the value for the attribute in *orclsubscriberricknameattribute* (that is, the *o* attribute) is set to *company*. See also "[SIP Servlet Container](#)".

Mapping JAAS Roles to LDAP Groups

Group membership determines the JAAS roles for a specific user. Mapping LDAP groups to JAAS roles is based on the value given to *orclcommonnamingattribute* under the node *cn=Common*, *cn=Products*, *cn=OracleContext* for each of the provisioned LDAP Realms. For example, if a user belongs to an LDAP group with the distinguished name of *cn=Location Service, cn=groups, dc=example, dc=com* and the

orclcommonnamingattribute is set to *cn*, then that JAAS user is populated with the "Location Service" JAAS role.

Installing OCMS Components into the OID LDAP Tree

To allow the OCMS container to connect to an instance of OID, its product container must be installed into OID's LDAP tree under the node `cn=Products,cn=OracleContext` for the appropriate LDAP realm(s). Each instance of OCMS that will connect to a given instance of OID must add an application entry under the OCMS product container, granting it the appropriate privileges, as described in ["Associating an OCMS Instance with OID"](#). In addition, if static verifiers are required for an OID deployment, an OCMS verifier entry must also be added under the OCMS product container. Installation of static verifiers is described in ["Installing the OCMS Static Verifiers"](#).

To integrate an instance of OCMS into a deployment of OID, you must make entries and modifications to OID's LDAP tree under the appropriate LDAP realm(s) (the examples use sample realm under `dc=example,dc=com`).

These modifications can be made through a graphical tool such as `oidadmin` or through a command-line tool such as `ldapmodify` with the appropriate LDIF definitions. The following sections provide examples of the LDIF configuration data that could be used to install OCMS components into OID's LDAP tree.

Associating an OCMS Instance with OID

Perform the following steps to associate an OCMS instance with OID:

1. Add the OCMS product container entry under the `cn=Products,cn=OracleContext` node for the appropriate LDAP realm:

```
dn: cn=OCMS,cn=Products,cn=OracleContext,dc=example,dc=com
changetype: add
objectclass: orclContainer
```

2. Associate an instance of OCMS by adding an OCMS application entry under the OCMS product container created above.

```
dn:
orclApplicationCommonName=OCMSInstance1,cn=OCMS,cn=Products,cn=OracleContext,
dc=example,dc=com
changetype: add
objectclass: orclApplicationEntity
orclappfullname: OCMS Instance 1
userpassword: password1
description: OCMS Instance 1 of the OCMS SIP Container
```

3. Grant verifier services privileges to the OCMS application by adding the OCMS application object as a member of the verifier services group.

```
dn:cn=verifierServices,cn=Groups,cn=OracleContext,dc=example,dc=com
changetype: modify Grant verifier services privileges to the OCMS application
by adding the OCMS application object as a member of the verifier services
group.
add: uniquemember
uniquemember: orclApplicationCommonName=OCMSInstance1,
cn=OCMS,cn=Products,cn=OracleContext,dc=example,dc=com
```

4. Repeat steps 2 and 3 for each instance of OCMS wishing to associate with this OID LDAP server for its user data store.

Installing the OCMS Static Verifiers

Perform the following step to install OCMS static verifiers:

1. Add the OCMS verifier entry under the OCMS product container. This step is not required for deployments of OID configured to use dynamic verifiers.

```
dn: cn=OCMSVerifierProfileEntry,cn=OCMS,cn=Products,cn=OracleContext,
dc=example,dc=com
objectclass:top
objectclass:orclpwdverifierprofile
cn:OCMSVerifierProfileEntry
orclappid:ocms
orclpwdverifierparams;authpassword: crypto:SASL/MD5 $ realm:example $
usernameattribute:mail $usernamecase:lower
```

Set the `usernameattribute` above to the value of the `orclcommonnicknameattribute` under the node `cn=Common,cn=Products,cn=OracleContext` for the given LDAP realm where the verifier is to be installed.

Set the `realm` verifier parameter to match one of the JAAS realms configured in the `SipServletContainer`'s `DomainsAndRealms` JMX attribute, as described in ["Mapping JAAS Realms to LDAP Subscribers"](#).

Set the `orclappid` attribute to a unique component name assigned to the OCMS component. The value of this attribute determines the value of the `StaticVerifierAttribute` used for the Security Service EJB configuration in ["Configuring User Service and Security Service"](#). For example, if `orclappid` is set to "ocms", then the `StaticVerifierAttribute` should be set to "authpassword;ocms", following the general static verifier pattern of `authpassword;orclappid` (where `authpassword` is a static string).

Refer to *Oracle Internet Directory Administrator's Guide 10g* for more information on static verifiers.

Repackaging Subscriber Data Services

Configuring OCMS to support OID requires that Subscriber Data Services (`subscriberdataservices.ear`) and its child applications be undeployed from the OCMS OC4J instance. Before the application and its child applications can be re-deployed, the user service and security service EJB configuration must be altered by adding the following LDAP configuration parameters to the `ejb-jar.xml` files for `securityservice.jar` and `userservice.jar`:

- `java.naming.security.principal`
- `java.naming.provider.url`
- `java.naming.security.protocol` (an optional parameter)

The user service EJB configuration also exposes the `SipUriLdapAttribute`, which defines the LDAP user attribute where the SIP URI is stored. This attribute defaults to `mail` if no value is defined.

Configuring User Service and Security Service

To configure the Subscriber Data Services application with OID as the user provisioning store:

1. Copy the EAR file of the Subscriber Data Services application (`subscriberdataservices.ear`) as well as its child applications to a temporary directory.

2. Undeploy the Subscriber Data Services application and its child applications from the OC4J instance. See ["Deploying, Undeploying, and Redeploying SIP Servlet Applications with Application Server Control"](#).
3. Expand the Subscriber Data Services application in the temporary directory.
4. Expand `securityservice.jar`
5. Edit `ejb-jar.xml` (located under META-INF) by replacing the following entry with the entry listed in [Example 4-3](#) that includes the `java.naming.security.principal`, `java.naming.provider.url`, and the `java.naming.security.protocol` parameters.

```

    <env-entry>
      <description><![CDATA[Datasource for Service activation
facades]]></description>
      <env-entry-name>SecurityServiceDSN</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value><![CDATA[java:jdbc/OcmsSsDs]]></env-entry-value>
    </env-entry>
    <env-entry>
      <env-entry-name>SecurityDAOImpl</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[com.hotsip.securityservice.dao.timesten.SecurityDAOIm
pl]]>
      </env-entry-value>
    </env-entry>
    <env-entry>
      <env-entry-name>StoreHashedCredentials</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value><![CDATA[True]]></env-entry-value>
    </env-entry>
    <env-entry>
      <description><![CDATA[Datasource for Service activation
facades]]></description>
      <env-entry-name>UserServiceDSN</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value><![CDATA[java:jdbc/OcmsUsDs]]></env-entry-value>
    </env-entry>
    <env-entry>
      <env-entry-name>UserDAOImpl</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[oracle.sdp.userservice.dao.timesten.UserDAOImpl]]>
      </env-entry-value>
    </env-entry>

```

Example 4-3 `ejb-jar.xml` Entries

```

    <env-entry>
      <description><![CDATA[DN of the OCMS LDAP application entry for this
instance of OCMS.]]></description>
      <env-entry-name>java.naming.security.principal</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[orclApplicationCommonName=OCMSInstance1,cn=OCMS,cn=Produ
cts,cn=OracleContext,dc=example,dc=com]]></env-entry-value>
    </env-entry>
    <env-entry>
      <description><![CDATA[Password for the OCMS LDAP application entry for

```

```

this instance of OCMS.]]></description>
    <env-entry-name>java.naming.security.credentials</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[{903}FOO48C7YXgS6EMBZ4I47/Xs0JsJXuHOU1VCyJzWHXo=]]></env-
v-entry-value>
    </env-entry>
    <env-entry>
        <description><![CDATA[LDAP Provider URL]]></description>
        <env-entry-name>java.naming.provider.url</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[ldap://ldapusers.example.com:636]]></env-entry-value>
    </env-entry>
    <env-entry>
        <description><![CDATA[Security Protocol (e.g. ssl)]]></description>
        <env-entry-name>java.naming.security.protocol</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value><![CDATA[ssl]]></env-entry-value>
    </env-entry>
    <env-entry>
        <description><![CDATA[The LDAP user attribute containing the user's
static verifier hash.]]></description>
        <env-entry-name>StaticVerifierAttribute</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value><![CDATA[authpassword;ocms]]>
        </env-entry-value>
    </env-entry>
    <env-entry>
        <description><![CDATA[The optional LDAP filter that should be applied
when searching for users.]]></description>
        <env-entry-name>UserSearchFilter</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[(&(mail=*@example.com)(orclisvisible=true)]]>
    </env-entry-value>
    </env-entry>
    <env-entry>
        <env-entry-name>SecurityDAOImpl</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[oracle.sdp.securityservice.dao.ldap.SecurityDAOImpl]]>
    </env-entry-value>
    </env-entry>
    <env-entry>
        <env-entry-name>UserDAOImpl</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[oracle.sdp.userservice.dao.ldap.UserDAOImpl]]>
    </env-entry-value>
    </env-entry>

```

Note: The `java.naming.security.principal`, `java.naming.security.credentials` and `java.naming.provider.url` environment entries must be updated with the LDAP server's configuration. Typically, the entry for `java.naming.security.principal` will be set to the dn for the `orclApplicationEntity` object associated with the current instance of OCMS. The optional entry, `java.naming.security.protocol`, must be set to `"ssl"` for SSL-based connections to the OID LDAP server.

The value of `java.naming.security.credentials` can be provided in cleartext by prepending the `"!"` character to the cleartext password (for example, `!"password"`). For additional security, use the obfuscated form of the password.

In OID deployments configured to use dynamic verifiers, the `StaticVerifierAttribute` entry is not needed and should be removed. In OID deployments configured to use static verifiers, the `StaticVerifierAttribute` entry should be set to the user attribute where the OCMS static verifier is stored. Typically, the value of this attribute is `"authpassword;orclappid"`, where `authpassword` is a static string and `orclappid` is replaced by the value of the `orclappid` attribute uniquely identifying the OCMS static verifier installed into OID.

The optional `UserSearchFilter` attribute sets the value of the LDAP filter to apply when searching for users in the LDAP repository. If omitted, no user search filter will be applied.

6. Repackage `securityservice.jar`.
7. Expand the `userservice.jar`.
8. Edit the `ejb-jar.xml` under META-INF by replacing the following entry with the entry described in [Example 4-4](#) that includes the `java.naming.security.principal`, `java.naming.provider.url`, `java.naming.security.protocol`, and `SipUriLdapAttribute` parameters. Do not remove the `UserServiceDSN` entry.

```

    <env-entry>
      <description><![CDATA[Datasource for Service activation
facades]]></description>
      ...
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value><![CDATA[java:jdbc/OcmsUsDs]]></env-entry-value>
    </env-entry>
    <env-entry>
      <env-entry-name>UserDAOImpl</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>

    <env-entry-value><![CDATA[oracle.sdp.userservice.dao.timesten.UserDAOImpl]]>
      </env-entry-value>
    </env-entry>

```

Example 4-4 userservice.jar Entries

```

    <env-entry>
      <description><![CDATA[DN of the OCMS LDAP application entry for this
instance of OCMS.]]></description>
      <env-entry-name>java.naming.security.principal</env-entry-name>

```

```

    <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[orclApplicationCommonName=OCMSInstance1,cn=OCMS,cn=Products,cn=OracleContext,dc=example,dc=com]]></env-entry-value>
</env-entry>
<env-entry>
  <description><![CDATA[Password for the OCMS LDAP application entry for this instance of OCMS.]]></description>
  <env-entry-name>java.naming.security.credentials</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[{903}FOO48C7YXgS6EMBZ4I47/Xs0JsJXUHOULVCyJzWHXo=]]></env-entry-value>
</env-entry>
<env-entry>
  <description><![CDATA[LDAP Provider URL]]></description>
  <env-entry-name>java.naming.provider.url</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[ldap://ldap.example.com:636]]></env-entry-value>
</env-entry>
<env-entry>
  <description><![CDATA[The optional LDAP user attribute containing the user's SIP URI.]]></description>
  <env-entry-name>SipUriLdapAttribute</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value><![CDATA[mail]]></env-entry-value>
</env-entry>
<env-entry>
  <description><![CDATA[The optional LDAP filter that should be applied when searching for users.]]></description>
  <env-entry-name>UserSearchFilter</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[(&(mail=*@example.com)(orclisvisible=true)]]>
</env-entry-value>
</env-entry>
<env-entry>
  <description><![CDATA[Security Protocol (e.g. ssl)]]></description>
  <env-entry-name>java.naming.security.protocol</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value><![CDATA[ssl]]></env-entry-value>
</env-entry>
<env-entry>
  <env-entry-name>UserDAOImpl</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value><![CDATA[oracle.sdp.userservice.dao.ldap.UserDAOImpl]]>
</env-entry-value>
</env-entry>

```

Note: The `java.naming.security.principal`, `java.naming.security.credentials` and `java.naming.provider.url` environment entries must be updated with the LDAP server's configuration. In addition, the optional entry, `java.naming.security.protocol`, must be set to "ssl" for SSL-based connections to the OID LDAP server. The `userservice` exposes an optional environment parameter, `SipUriLdapAttribute`. The value set for this entry is the LDAP user attribute where the SIP URI is stored. If no value is set for this entry, then the attribute defaults to `mail`.

9. Repackage `userservice.jar`.
10. Repackage the EAR file for Subscriber Data Services.
11. Redeploy the repackaged Subscriber Data Services application and its child applications to the OCMS OC4J instance.

Provisioning OCMS Users to OID

You must provision user accounts to OID using the Oracle Identity Management (OIM) Web-based Oracle Delegated Administration Services application (OIDDAS), described in *Oracle Identity Management Guide to Delegated Administration*.

You access this tool from a browser by entering `http://<host>:<port>/oiddas`, where the host and port are the hostname and HTTP port for the Oracle Application Server instance of the OID deployment.

Adding Users to LDAP Groups

Proxy registrar can be configured to have users with any roles by assigning a default PUBLIC role.

Using Oracle Delegated Administration Services, you can add a user to an LDAP group by adding the distinguished name of the user to the list of values in the LDAP group's *uniquemember* attribute. For more information, refer to Oracle Identity Management Guide to Delegated Administration.

You can also configure the `sip.xml` authentication constraints in `proxyregistrar` to require user membership in a particular role (for example, Location Service), based on the SIP message type.

Configuring High Availability

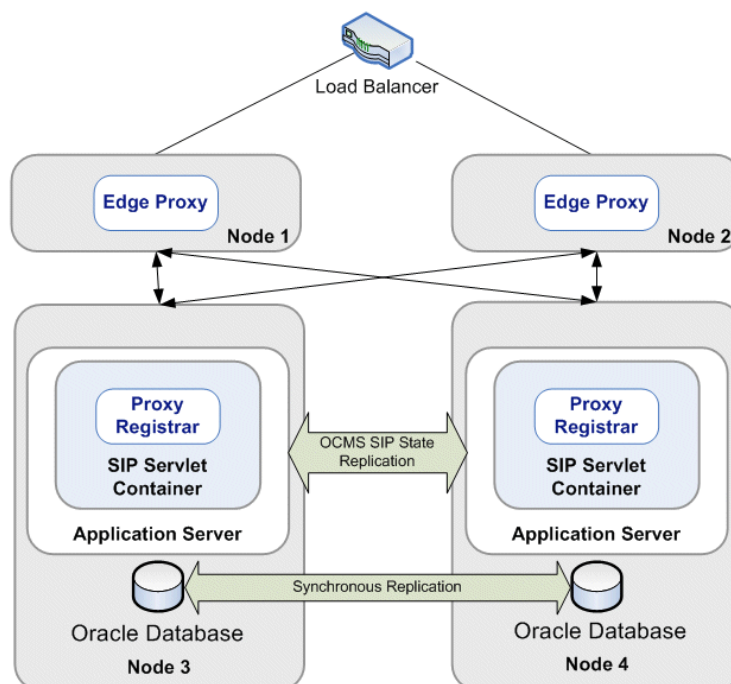
This chapter discusses configuring high availability through the following sections:

- "About Configuring High Availability"
- "Setting Up a Highly Available Cluster of OCMS Nodes"
- "Configuring the OCMS SIP Containers for High Availability"
- "Configuring the Edge Proxy Nodes for High Availability"
- "Configuring Highly Available SIP Servlet Applications"
- "Configuring an Overload Policy"

About Configuring High Availability

OCMS provides high availability through redundancy, application state replication, and clustering. Highly available OCMS topologies are active-active, meaning that any redundant nodes actively function in the context of the topology. This makes OCMS scalable as well.

Figure 5–1 Highly Available OCMS Topology



A highly available OCMS topology (Figure 5–1) provides the following:

- **Process death detection and automatic restart**—Processes may die unexpectedly due to configuration or software problems. A proper process monitoring and restart system monitors all system processes constantly and restarts them if necessary.
- **Clustering**—Clustering components of a system together allows the components to be viewed functionally as a single entity from the perspective of a client for runtime processing and manageability. A cluster is a set of processes running on single or multiple computers that share the same workload. There is a close correlation between clustering and redundancy. A cluster provides redundancy for a system.
- **Configuration management**—A clustered group of similar components often need to share common configuration. Proper configuration management ensures that components provide the same reply to the same incoming request, allows these components to synchronize their configurations, and provides highly available configuration management for less administration downtime.
- **State replication and routing**—For stateful applications, client state can be replicated to enable stateful failover of requests in the event that processes servicing these requests fail.
- **Server load balancing and failover**—When multiple instances of identical server components are available, client requests to these components can be load balanced to ensure that the instances have roughly the same workload. With a load balancing mechanism in place, the instances are redundant. If any of the instances fails, then requests to the failed instance can be sent to the surviving instances.

Configuring a highly available OCMS environment involves the following main steps, depending on the OCMS topology you have chosen to deploy:

- [Setting Up a Highly Available Cluster of OCMS Nodes](#)—This involves associating each OCMS node with OPMN to form a manageable cluster.
- [Configuring the OCMS SIP Containers for High Availability](#)—Use the Application Server Control Console MBean browser to configure parameters affecting high availability in the SIP Servlet container.
- [Configuring the Edge Proxy Nodes for High Availability](#)—Optional. Required only if using Edge Proxy nodes in the OCMS topology.
- [Configuring Highly Available SIP Servlet Applications](#)—Edit the SIP Servlet application descriptor files to enable support for high availability. Configure state replication for each application.
- ["Configuring an Overload Policy"](#)—Optional. Required only if using the Proxy Registrar in the OCMS topology.

Table 5–1 Additional Information

For more information on...	See:
OCMS deployment topologies	Chapter 2, "Deployment Topologies" in this guide
OCMS installation	<i>Oracle Communication and Mobility Server Installation Guide</i>
Operating systems supported by highly available OCMS clusters	<i>Oracle Communication and Mobility Server Certification Guide</i>

Table 5–1 (Cont.) Additional Information

For more information on...	See:
Configuring a highly available clustered Oracle Application Server environment	<ul style="list-style-type: none"> ▪ The "Application Clustering" chapter in <i>Containers for J2EE Configuration and Administration Guide</i>. ▪ The "Active-Active Topologies" chapter in <i>Oracle Application Server High Availability Guide</i>.

Setting Up a Highly Available Cluster of OCMS Nodes

Note: If using UDP, place all servers on the same subnet or switch to avoid the defragmentation of large UDP packages.

Each OCMS node—including the Edge Proxy nodes—must be configured to support high availability.

Following are the main steps in setting up a cluster of OCMS servers:

1. [Associating Nodes with OPMN](#)—Oracle Process Manager and Notification Server provides a command-line interface for process control and monitoring for single or multiple Oracle Application Server components and instances. Using OPMN, you can start and stop each OCMS node and its sub-components.
2. [Starting the Cluster](#)—Starting the cluster with OPMN indicates that all OCMS nodes have been correctly associated with OPMN and are recognized as a cluster.
3. [Verifying the Status of the Cluster](#)—Using OPMN or Enterprise Manager, you can verify that each node in the cluster is up and running.
4. [Stopping the Cluster](#)—Having set up and verified the cluster of OCMS nodes, be sure to stop the cluster before configuring each SIP container for high availability (see "[Configuring the OCMS SIP Containers for High Availability](#)").

Associating Nodes with OPMN

Setting up a cluster of OCMS nodes requires associating the nodes with OPMN. There are three ways to do this:

- Configuring the cluster during Oracle Application Server installation. For more information, refer to the *Oracle Application Server Installation Guide*.
- [Associating Nodes with OPMN Using the Dynamic Discovery Method](#)
- [Associating Nodes with OPMN Using the Discovery Server Method](#)

See also: For more information regarding configuring and managing clusters using OPMN, see *Oracle Process Manager and Notification Server Administrator's Guide*.

Associating Nodes with OPMN Using the Dynamic Discovery Method

In this method -- one that is recommended by Oracle -- you define the same multicast address and port for each Oracle Application Server instance in the cluster. An advantage in using this method is that you do not have to specify the name of each

Oracle Application Server instance in the cluster. Instead, you can dynamically add or remove instances from the cluster by editing the multicast address and port.

1. For each Oracle Application Server instance that you want to group in the same cluster, run the following command:

```
$ORACLE_HOME/opmn/bin/opmnctl config topology update
discover="*<multicastAddress>:<multicastPort>"
```

For example:

```
$ORACLE_HOME/opmn/bin/opmnctl config topology update
discover="*225.0.0.20:6200"
```

where:

- `multicastAddress` specifies the multicast address that you want to use for the cluster. The multicast address must be within the valid address range, which is 224.0.1.0 to 239.255.255.255. Note that the multicast address is preceded by an asterisk (*).
- `multicastPort` can be any unused port number.

Use the same multicast IP and port for all the instances.

2. On each Oracle Application Server instance where you ran the command in Step 1, run `opmnctl reload` so that OPMN reads the updated `opmn.xml` file.

```
$ORACLE_HOME/opmn/bin/opmnctl reload
```

Associating Nodes with OPMN Using the Discovery Server Method

Although Oracle recommends associating nodes with OPMN using the dynamic discovery method, you can also define a cluster by specifying the names of the nodes running the Oracle Application Server instances in the `opmn.xml` file for each instance. For example, to cluster four instances (*node1.example.com*, *node2.example.com*, *node3.example.com*, *node4.example.com*), associate these nodes with OPMN using the discovery server method as follows:

1. Run Oracle Application Server on all nodes.
2. Designate one instance as the discovery server, which maintains the topology for the cluster. (In this example, *node1.example.com* acts as the discovery server for the cluster.)
3. In the `opmn.xml` file for all instances in the cluster, specify the node that is running the discovery server (*node1.example.com* in [Example 5-1](#)). As illustrated in [Example 5-1](#), the `opmn.xml` file includes the `<discover>` element. The 6200 value specifies the port number on which the notification server listens. Use the remote port number designated in the `<port>` sub-element of the `<notification-server>` element.

Example 5-1 Designating an Instance as the Discovery Server

```
<?xml version="1.0" encoding="UTF-8"?>
<opmn xmlns="http://www.oracle.com/ias-instance">
...
<notification-server interface="ipv4">
  <port local="6100" remote="6200" request="6003"/>
  ...
</notification-server>
<discover list="node1.example.com:6200"/>
```

```

        </topology>
        ...
    </notification-server>
    <process-manager>
    ...
    </process-manager>
</opmn>

```

4. On all server instances, run `opmnctl reload` so that OPMN loads the updated `opmn.xml` file:

```
$ORACLE_HOME/opmn/bin/opmnctl reload
```

Starting the Cluster

To start the cluster using OPMN run the following command on each instance in the cluster:

```
cd ORACLE_HOME/opmn/bin/
$ORACLE_HOME/opmn/bin/opmnctl startall
```

Verifying the Status of the Cluster

To verify the status of the OCMS nodes in the cluster:

1. In a Web browser, enter the URI of Enterprise Manager running on any SIP container in the cluster:

```
http://<SIP container URI>:<port number>/em
```

2. Enter the administrator user name and password at the prompt.
Enterprise Manager displays the status of the cluster topology.

Stopping the Cluster

After verifying the status of the cluster, stop the nodes in the cluster using OPMN so that you can continue configuring the SIP containers (see "[Configuring the OCMS SIP Containers for High Availability](#)").

To stop OCMS, execute the following command on each node in the cluster:

```
cd ORACLE_HOME/opmn/bin/
$ORACLE_HOME/opmn/bin/opmnctl stopall
```

Configuring the OCMS SIP Containers for High Availability

In the Application Server Control Console MBean browser, configure the following parameters under the [SIP Servlet Container](#) MBean for each SIP Application Server node:

1. Configure the *EdgeProxy* parameter in the SIPContainer Mbean to point to the SIP URI of the Edge Proxy or to a third-party load balancer if more than one Edge Proxy is used.

Use the following format:

```
SIP:<Edge Proxy or Load Balancer IP address>:<port>;lr
```

2. Configure the *DistributableRecordRoute* parameter in the following format:

SIP:<SIP Container IP address>:<port>

Remove any appended transport methods (such as `transport=tcp`) to enable any type of transport to be used between the Edge Proxy and OCMS.

3. Configure the *RecordRoute* parameter using the following format:

SIP:<SIP Container IP address>:<port>

Remove any appended transport methods (such as `transport=tcp`) to enable any type of transport to be used between the Edge Proxy and OCMS.

Configuring the Edge Proxy Nodes for High Availability

Note: In the load balancer, you must disable stickiness for UDP datagrams sent to the Edge Proxy servers. Refer to the load balancer documentation for more information on disabling stickiness when sending datagrams over UDP.

To configure the Edge Proxy nodes for high availability:

- Configure each OCMS node running an Edge Proxy for high availability as described in ["Setting Up a Highly Available Cluster of OCMS Nodes"](#) and ["Configuring the OCMS SIP Containers for High Availability"](#).
- Point the *edgeproxy* parameter in the SIPContainer Mbean to one of the following:
 - The IP address of the Edge Proxy node.
 - For more than one Edge Proxy node -- The virtual IP address or host name of the third-party load balancer or DNS server if clients connect using DNS lookup.

Note: When setting up OCMS in a high-availability environment with multiple Edge Proxy nodes and a load balancer, the SIP port for the Edge Proxy and the Virtual Server on the load balancer must be the same.

- Configure the interval at which SIP Container nodes ping the Edge Proxy nodes, as well as the number of missed ping intervals before the Edge Proxy nodes remove the unresponsive SIP Container from the routing table. These parameters enable the Edge Proxy node(s) to monitor the health of the SIP Container nodes.

For each Edge Proxy node in the topology, configure the following:

1. In the Application Server Control Console Mbean Browser, click the *edgeproxy* Mbean.
2. Configure the *RecordRoute* parameter to point to one of the following:
 - **For a single Edge Proxy without a load balancer**—Set the parameter to the IP address of the Edge Proxy node
 - **For more than one Edge Proxy with a load balancer or DNS server**—Set the parameter to the virtual IP address or host name of the third-party load balancer or DNS server (if clients connect using a DNS lookup)

3. Modify the `edgeproxy.xml` file (`sdp/edgeproxy/conf/edgeproxy.xml`, illustrated in [Example 5-2](#)) to include the `oc4j-ping` element:

```
<oc4j-ping interval="1" allowed-miss-count="16"/>
```

The `oc4j-ping` element configures the interval, in seconds, at which the Oracle Application Servers in the cluster ping the Edge Proxy. The `allowed-miss-count` attribute specifies the number of missed ping intervals allowed before the Edge Proxy removes an unresponsive Oracle Application Server from the routing table.

Example 5-2 `edgeproxy.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<edge-proxy xmlns:xsi="http://www.oracle.com/sdp">
  <record-route sip-uri="sip:%IPADDRESS%:%SIPPORT%" />
  <jmx-rmi-connector port="%EPRMIIPORT%" />
  <oc4j-ping interval="1" allowed-miss-count="16"/>
  <nat-traverse enabled="true" />
  <sip-stack ip="%IPADDRESS%">
    <listening-point transport="tcp" port="%SIPPORT%" />
    <listening-point transport="udp" port="%SIPPORT%" />
  </sip-stack>
</edge-proxy>
```

4. In the `edgeproxy.xml` file, modify the `nat-traverse` element if necessary.
 - If the Edge Proxy enables SIP clients to traverse of NATs (Network Address Translators), then set the value to `true` (the default). The corresponding default value must be set in Oracle Communicator.
 - If NAT traversal is not used, then this attribute must be set to `false`. For more information disabling NAT traversal, see ["Disabling NAT Traversal Enabled by the Edge Proxy"](#).
5. Verify the status of the Edge Proxy node or nodes in the cluster by performing the following:
 - [Starting the Cluster](#)
 - [Verifying the Status of the Cluster](#)

For more information, refer to "Configuring OCMS in a Clustered Environment with Edge Proxy" in *Oracle Communication and Mobility Server Installation Guide*

The NAT Traversal Option Enabled for the Edge Proxy

NAT traversal enables access to SIP User Agents even when they are located behind firewalls or NATs. To support SIP clients residing behind firewalls or NATs, proxy servers use the Path extension header mechanism (described in RFC 3327), which ensures that SIP clients follow specific paths that enable the traversal of NATs and firewalls throughout the network. When you enable the NAT traversal function in the Edge Proxy, an OCMS cluster supports the Path extension header mechanism by inserting a Path header field into REGISTER requests.

Disabling NAT Traversal Enabled by the Edge Proxy

By default, NAT traversal is enabled in `edgeproxy.xml` (`nat-traverse enabled=true`, as noted in [Example 5-2](#)). To disable this function:

1. If the Edge Proxy is running, stop it by entering the following command:

```
$ORACLE_HOME/opmn/bin/opmnctl stopproc process-type=EdgeProxy
```

2. Edit the `nat-traverse` element of `edgeproxy.xml` (located at `ORACLE_HOME/sdp/edgeproxy/conf/edgeproxy.xml`) as follows:

```
<nat-traverse enabled="false"/>
```

3. Start the Edge Proxy using the following command:

```
$ORACLE_HOME/opmn/bin/opmnctl startproc process-type=EdgeProxy
```

4. Repeat these steps for each Edge Proxy node in the OCMS cluster.

Caution: When NAT traversal is enabled, the Edge Proxy nodes insert their local IP addresses into the *RecordRoute* headers of SIP requests. Therefore, the Edge Proxy nodes must be globally routable. This may not be the case if the cluster has been configured according to the white paper, *Oracle Communication and Mobility Server in a High-Availability Environment Running with F5 BigIP* (available at the Oracle Technology Network).

Configuring Highly Available SIP Servlet Applications

This section describes how to configure high availability for SIP Servlet applications deployed to a cluster of OCMS nodes.

- [Enabling High Availability in SIP Servlet Applications](#)—Prior to deployment, each application’s descriptor files (`web.xml` and `sip.xml`) must be configured for high availability. The `orion-application.xml` file for each application must also be configured for high availability.
- [Configuring Application Session Data Replication](#)—The session data of each SIP Servlet application can be replicated to other nodes in the cluster, in the event of node failure.
- [Configuring High Availability for a Deployed SIP Servlet Application](#)—You can also configure high availability for an application that you have already deployed.
- [Disabling High Availability at the Application Level](#)—You can remove support for high availability from your SIP Servlet applications.
- [Upgrading SIP Servlet Applications in OCMS](#)—This section explains how to perform a rolling upgrade on deployed SIP Servlet applications.

Notes:

- When configuring high availability for SIP Servlet applications that depend upon the Proxy Registrar, you must also configure the Proxy Registrar for high availability.
 - High availability is not currently supported for converged applications (meaning applications comprised of both SIP and HTTP servlets).
-
-

Enabling High Availability in SIP Servlet Applications

To configure a highly available SIP Servlet application:

1. Modify the `sip.xml` file (located at `ORACLE_HOME/j2ee/ocms/applications/<application name>/<web module name>/WEB-INF/sip.xml`) to include the `<distributable>` element.

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
  <sip-app>
    <display-name>proxyregistrarssr</display-name>
    <b><distributable/></b>
  <!--Servlets-->
  <servlet>
    <servlet-name>Registrar</servlet-name>
    <servlet-class>oracle.sdp.registrar.VoipRegistrarServlet</servlet-class>
    <init-param>
      <param-name>LocationService</param-name>
      <param-value>oracle.sdp.locationdbservice.LocationDbServiceBD
    </param-value>
    </init-param>
  </servlet>
</sip-app>
```

2. Modify the `web.xml` file (located at `ORACLE_HOME/j2ee/ocms/applications/<application name>/<web module name>/WEB-INF/web.xml`) to include the `<distributable>` element.

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <display-name>proxyregistrarssr</display-name>
  <b><distributable/></b>
</web-app>
```

3. Modify the `orion-application.xml` file (located at `ORACLE_HOME/j2ee/ocms/application-deployments/<application name>/orion-application.xml`) to include the `<cluster>` element which is used to configure clustering both for Oracle Application Server nodes as well as specific SIP servlet applications.

For example:

```
<orion-application ... >
  <cluster allow-colocation="false">
    ...
  </cluster>
</orion-application>
```

The `<cluster>` element, which is used in both `orion-application.xml` and `application.xml` files, includes the following sub-elements that control application replication:

- `enabled`—Specifies whether clustering is enabled. The default value is `true`.
- `group-name`—The name to use when establishing replication group channels. If not supplied, the application name as defined in `server.xml` (the Oracle Application Server configuration file) is used by default. New group channels are created for each enterprise application. If a value is specified, the application and all child applications use the channels associated with this group name.

- `allow-colocation`—Specifies whether to allow application state to be replicated to a node residing on the same host machine. The default value is `true`.

Note: Although the default value is `true`, set `allow-colocation` to `false` if multiple hosts are available. If multiple Oracle Application Server instances are instantiated on the same machine, specify different listener ports for each instance in the `default-web-site.xml`, `jms.xml`, and `rmi.xml` configuration files.

- `write-quota`—The number of other group members to which the application state should replicate to. This attribute enables reducing overhead by limiting the number of nodes to which state is written. The default value is 1.

For additional information regarding the `<cluster>` element and its sub-elements, refer to the chapter "Configuring Application Clustering" in *Containers for J2EE Configuration and Administration Guide*.

Note: Ensure that you use the correct spelling for attributes in the `orion-application.xml` file, as misspellings will not result in error messages. For example, if you misspell `start-port` in the cluster configuration section as `start-prt`, replication will appear to have started even though session replication does not work.

4. Repeat these steps for each application deployed on each OCMS instance.

Important: Deploy the application symmetrically to all SIP application server nodes.

For information about developing highly available SIP Servlet applications, refer to *Oracle Communication and Mobility Server Developer's Guide*.

Configuring Application Session Data Replication

OCMS supports multicast replication by defining the `orion-application.xml` file's `<cluster>` and `<property-config>` elements. The `<property-config>` element contains data required to use the JGroups group communication protocol to replicate session state across nodes in the cluster.

To set the replication policy, edit the `ORACLE_HOME/j2ee/ocms/application-deployments/<application name>/orion-application.xml` file as follows:

- Set the `<cluster>` element's `allow-colocation` attribute to `false`.
- Set the `<property-config>` element's `<url>` element to the path to the JGroup's XML file describing the application-related high-availability configuration (illustrated in [Example 5-4](#)).

Example 5-3 Editing `orion-application.xml` File for Replication

```
<orion-application ... >
```

```

...
<cluster allow-colocation="false">
...
  <property-config>
    <url>file:///ORACLE_HOME/j2ee/ocms/application-deployments/
      <application name>/jgroups-tcp.xml</url>
  </property-config>
</cluster>

```

[Example 5-4](#) illustrates the JGroups XML file (referred to as `jgroups-tcp.xml`) in [Example 5-3](#).

Example 5-4 A Sample JGroups Application High Availability Configuration File

```

<config>
  <TCP/>
  <MPING mcast_addr="230.0.0.130" mcast_port="8500" ip_ttl="1"/>
  <MERGE2 min_interval="5000" max_interval="10000"/>
  <FD timeout="1000" max_tries="3" shun="false"/>
  <FD_SOCKET/>
  <VERIFY_SUSPECT timeout="1000"/>
  <pbcast.NAKACK gc_lag="100" retransmit_timeout="3000"/>
  <pbcast.STABLE desired_avg_gossip="20000"/>
  <pbcast.GMS join_timeout="3000" join_retry_timeout="2000" shun="false"
    print_local_addr="true"/>
</config>

```

In [Example 5-4](#), failure detection (set by `<FD timeout>`) is set in milliseconds. In the sample JGroups file, it is set at 1000 milliseconds with three retries (`max_tries="3"`).

Configuring High Availability for a Deployed SIP Servlet Application

Perform the following if the SIP Servlet application has already been developed and deployed, but not configured for high availability.

1. Undeploy the SIP Servlet application.
2. Unpack the application EAR.
3. Modify the `sip.xml` and `web.xml` files to include the `<distributable>` element, as described in ["Enabling High Availability in SIP Servlet Applications"](#). To edit the `sip.xml` and `web.xml` files, do the following:
 - a. Create a new folder and name it `<Your SIP application>`.
 - b. Unpack the EAR file to the new folder.
 - c. Unpack the WAR file inside the EAR file to the folder `<Your SIP application>/<WAR module name>`.
 - d. Edit the following files:


```

<Your SIP application>/<WAR module name>/WEB-INF/sip.xml
<Your SIP application>/<WAR module name>/WEB-INF/web.xml

```
 - e. Under `<Your SIP application>/<WAR module name>`, re-package the contents of the WAR file using the original WAR file name. Replace the original WAR file with in the `<Your SIP application>` folder with the new one that you just created.
 - f. Delete the `<Your SIP application>/<WAR module name>` folder.

Configuring an Overload Policy

The Overload Policy enables OCMS to execute overload protection when capacity reaches high threshold levels. To configure this MBean, you first set the maximum value for number of SIP sessions (*SipSessionTableMaxSize*) and the maximum size for the Application (*AppQueueMaxSize*) and Network queues (*StackQueueMaxSize*) as described in [Table 5–2](#). You then set the threshold values for each of these, as described in [Configuring High and Low Thresholds](#).

In addition to the maximum values settings, the *AllowedTrafficDuring503* attribute enables you to set the percentage of traffic allowed to pass through the system when it becomes overloaded and issues the 503 (*Service Unavailable*) error response to clients. The system does not process any new incoming requests if you set this attribute to zero (0), the default value.

Note: Refer to the load figures displayed for the *ApplicationPeakQueue*, *NetworkPeakQueue*, *Sessions*, *503ResponseSent* and *MessagesDropped* attributes of the [SIP Servlet Container Monitor](#) when setting these values.

Table 5–2 Default Values for SIP Sessions, Application and Network Queues

Attribute	Default Value
AppQueueMaxSize	200
StackQueueMaxSize	100
SipSessionTableMaxSize	70000

Note: Because of the internal use of the queues by OCMS, the actual peak values for the queues in an overload situation may exceed the maximum values configured in the Overload Policy.

Overview of Overload Policy Architecture

The Overload Policy receives collectors and actions from the Policy Manager (the lookup service for collectors, actions and policies). The Policy Manager sends notifications to all observers when available collectors, actions, or policies have changed.

Collectors

Collectors notify policies when states change. The Overload Policy subscribes to state changes for the following collectors:

- [Memory Usage](#)
- [Application Queue](#)
- [Network Queue](#)
- [SIP Session Table Usage](#)

Overview of Overload Policy

The Overload Policy implements the following default actions, for which it enacts an overload action when high threshold values are reached and then reverts this action when usage sinks to the low threshold value:

- Do not accept new connections: This overload action occurs when a high warning threshold value has been met.
- Send 503 (*Service Unavailable*) response on initial requests: This overload action occurs when a high alarm threshold value has been met.
- Stop reading from all connections: This overload action occurs when a high critical threshold has been met.

Configuring High and Low Thresholds

The Overload Policy MBean enables you to configure the high and low values for the Warning, Alarm and Critical levels for [Memory Usage](#), [Application Queue](#), [Network Queue](#), and [SIP Session Table Usage](#). At each level, there may be one or even several actions to execute if usage exceeds the specified threshold. When the high value set for a threshold is met, the Overload Policy calls overload actions. This value is the threshold at which these actions start. If usage drops to the low value set for a threshold, then the Overload Policy stops the overload action. For example, if a high level is set to 90 and low level is set to 80, overload actions that start when usage reaches 90% and are then stopped when usage drops back to 80%.

Starting and Stopping the Overload Policy

The start and stop operations for the Overload Policy MBean enable you manually start and stop the Overload Policy. To automatically start the Overload Policy at the startup of OCMS, set the *Autostart* attribute to *true*.

Memory Usage

The [Memory Monitor](#) reports memory usage to the Overload Policy. The usage is reported as percent of total memory. For example, if the Memory Monitor reports a value of 85, it means that 85% of total memory is currently in use and that 15% of it is free.

Using the Overload Policy MBean, you configure the Warning, Alarm and Critical threshold levels for the Overload Policy's memory usage. At each level, there may be one or several actions to execute if memory usage exceeds the specified threshold. The MBean enables you to set high and low threshold values for each of these levels. The high value is the threshold at which to start executing overload actions. The low level marks the threshold at which to stop executing the action. For example, if high level is set to 90 and low level is set to 80, the actions start when memory usage reaches 90% and then stop when memory usage drops to 80% again.

Delaying a Memory Overload Action

High and low levels keep overload actions from starting and stopping repeatedly for small changes in memory usage. In addition to the threshold level that sets the start of the executing actions, you can also configure a delay time (in seconds) for memory overload actions using the *MemoryActionDelay* attribute. When the high threshold is exceeded, a scheduled timer fires after the specified delay time (in seconds). Memory overload actions are not be executed before the delay time has passed, and if memory usage drops below high threshold during the delay time, the timer is canceled and no actions executes.

Configure the *MemoryActionDelay* attribute by entering the delay (in seconds) from the instance when the memory threshold value has been exceeded to the instant the actions are executed. The execution stops if the memory drops below the threshold during the delay. The value set for the delay must be greater than 0. The default value is 60 seconds.

[Table 5–3](#) lists the attributes that you configure to set the values for the high and low warning thresholds for memory usage.

Table 5–3 Warning High and Low Thresholds for Memory Usage

Attribute	Description
MemoryWarningHigh	The high value for a warning threshold that triggers an overload action to decline new connections. This is the default overload action. The range of values is 0-100. 0 disables the action. The default value is 95.
MemoryWarningLow	The low value for a warning threshold that triggers an end to the overload action. The range of values is 0-100. The default value is 90.
MemoryWarningActions	A comma-separated list of actions performed when a memory warning level has been reached. The default value is <i>oracle.sdp.networklayer.NetworkServiceImpl\$StopAcceptAction</i> .

[Table 5–4](#) lists the attributes that you configure to set the high and low alarm thresholds for memory usage.

Table 5–4 The Alarm High and Low Thresholds for Memory Usage

Attribute	Description
MemoryAlarmHigh	The high value for an alarm threshold for memory usage that triggers an action to send a 503 response (<i>Service Unavailable</i>) on initial requests. This is the default overload action. The range of values is 0-100. 0 disables the action. The default value is 95.
MemoryAlarmLow	The low value for the alarm threshold for memory usage that triggers an end to the overload action. The range of values is 0-100. The default value is 90.
MemoryAlarmActions	A comma-separated list of actions performed when a memory alarm threshold level has been reached. The default value is <i>com.hotsip.jainsipimpl.javax.sip.context.SipContextImpl\$Send503Action</i> .

[Table 5–5](#) lists the attributes that you configure to set the high and low critical thresholds for memory usage.

Table 5–5 Critical High and Low Thresholds for Memory Usage

Attribute	Description
MemoryCriticalHigh	The high value for a critical threshold for memory usage that triggers an overload action to stop reading connections. This is the default overload action. The range of values is 0-100; 0 disables the action. The default value is 98.
MemoryCriticalLow	The value that triggers an end to the overload action when the low threshold for memory usage has been reached. The range of values is 0-100. The default value is 90.
MemoryCriticalActions	A comma-separated list of action performed when the critical threshold of memory usage has been reached. The default value is <i>oracle.sdp.networklayer.NetworkServiceImpl\$StopReadAction</i> .

Application Queue

The Application Queue is the communication link between the network layer and the applications. An event is added to the Application Queue when network packages are framed and ready for application consumption after a session timer fires or other network related events occur. The network layer's EventNotifier reports Application Queue usage to the Overload Policy. This usage is reported as a percent of the total queue capacity. For example, if the EventNotifier reports a value of 85, it means that 85% of total queue capacity is currently used and that 15% of the queue is empty. The *AppQueueMaxSize* attribute sets the capacity of the Application Queue. The default value is 200. This value must always be greater than zero.

The EventNotifier does not report every change in Application Queue usage to the Overload Policy: below 95%, every 5% change is reported (that is, 0, 5, 10, 15 and so on). From 95% and above, every 1% change is reported (that is, 95, 96, 97 and so on). The Overload Policy MBean enables you to configure the Warning (Table 5-6), Alarm (Table 5-7), and Critical (Table 5-8) thresholds of the Application Queue.

Table 5-6 lists the attributes of the Overload Policy that enable you to set the high and low warning thresholds for the Application Queue usage.

Table 5-6 Warning High and Low Threshold and Actions for the Application Queue

Attribute	Value
AppQueueWarningHigh	The high threshold warning value for Application Queue usage that triggers an action to decline new connections. This is the default overload action. The range of values is 0-100. Selecting 0 disables the action. The default value is 70.
AppQueueWarningLow	The low threshold warning value for Application Queue usage that triggers an end to the overload action. The range of values is 0-100. The default value is 50.
AppQueueWarningActions	A comma-separated list of actions performed when a warning threshold for Application Queue usage has been reached. The default value is <i>oracle.sdp.networklayer.NetworkServiceImpl\$StopAcceptAction</i> .

Table 5-7 lists the attributes of the Overload Policy that enable you to set the high and low alarm thresholds for the Application Queue usage.

Table 5-7 Alarm High and Low Threshold Actions for the Application Queue

Attribute	Description
AppQueueAlarmHigh	The high alarm threshold value for the Application Queue usage that triggers an overload action to send a 503 Response (<i>Service Unavailable</i>) to initial requests. (This is the default overload action.) The range of values is 0-100. 0 disables the action. The default value is 80.
AppQueueAlarmLow	The low alarm threshold value for Application Queue usage that triggers an end to the overload action. The range of values is 0-100. The default value is 60.
AppQueueAlarmActions	A comma-separated list of action performed when an alarm threshold is reached. The default value is <i>com.hotsip.jainsipimpl.javax.sip.context.SipContextImpl\$Send503Action</i> .

Table 5-8 lists the attributes of the Overload Policy MBean that enable you to set the high and low critical thresholds for the Application Queue usage.

Table 5–8 Critical High and Low Threshold Actions for the Application Queue

Attribute	Description
AppQueueCriticalHigh	The high threshold value for Application Queue usage that triggers an overload action to stop reading connections. This is the default overload action. The range of values is 0-100; 0 disables the action. The default value is 90.
AppQueueCriticalLow	The low threshold value for Application Queue usage that triggers an end to the overload action. The range of values is 0-100. The default value is 70.
AppQueueCriticalActions	A comma-separated list of actions performed when a critical level is reached. The default value is <i>oracle.sdp.networklayer.NetworkServiceImpl\$StopReadAction</i> .

Network Queue

The network layer deposits the incoming unframed data from the network into the Network Queue. The network layer's EventQueue reports the Network Queue usage to the Overload Policy. The usage is reported as a percent of the total queue capacity. For example, if the EventQueue reports a value of 85, it means that 85% of total queue capacity is currently in use and 15% of the queue is empty. The *StackQueueMaxSize* attribute sets the Network Queue capacity. The default value for this attribute is 100. The value must always be greater than zero.

The EventQueue does not report every change in queue usage to the Overload Policy. Below 95%, every 5% change is reported (that is, 0, 5, 10, 15 and so on). From 95% and above, every 1% change is reported (that is, 95, 96, 97 and so on). The Overload Policy MBean enables you to configure the Warning (Table 5–9), Alarm (Table 5–10), and Critical (Table 5–11) thresholds of the Network Queue.

Table 5–9 lists the attributes that you configure to set the high and low threshold values that trigger Warning actions.

Table 5–9 Warning High and Low Threshold Actions for the Network Queue

Attribute	Description
StackQueueWarningHigh	The high warning threshold value for Network Queue usage that triggers the overload action to decline new connections. This is the default overload action. The range of values is 0-100; 0 disables the action. The default value is 70.
StackQueueWarningLow	The low warning threshold value that trigger end to the overload action. The range of values is 0-100. The default value is 50.
StackQueueWarningActions	A comma-separated list of actions performed when a warning threshold is reached. The default value is <i>oracle.sdp.networklayer.NetworkServiceImpl\$StopAcceptAction</i> .

Table 5–10 lists the attributes that you configure to set the high and low threshold values that trigger Alarm actions.

Table 5–10 Alarm High and Low Threshold Actions for the Network Queue

Attribute	Description
StackQueueAlarmHigh	The high alarm threshold for Network Queue usage that triggers an overload action to send a 503 (<i>Service Unavailable</i>) response to initial requests. This is the default action. The range of values is 0-100; 0 disables the action. The default value is 80.
StackQueueAlarmLow	The low alarm threshold for Network Queue usage that triggers an end to the overload action. The range of values is 0-100. The default value is 60.
StackAlarmQueueActions	A comma-separated list of actions performed when a threshold is reached. The default value is <code>com.hotsip.jainsipimpl.javax.sip.context.SipContextImpl\$Send503Action</code> .

Table 5–11 lists the attributes that you configure to set the high and low values that trigger Critical actions.

Table 5–11 Critical High and Low Threshold Actions for the Network Queue

Attribute	Description
StackQueueCriticalHigh	The high critical threshold value for Network Queue usage that triggers an overload action to stop reading all connections. (This is the default overload action.) The range of values is 0-100. 0 disables the action. The default value is 90.
StackQueueCriticalLow	The low critical threshold value that triggers an end to the overload action. The range of values is 0-100. The default value is 70.
StackQueueCriticalActions	A comma-separated list of actions performed when a critical level is reached. The default value is <code>oracle.sdp.networklayer.NetworkServiceImpl\$StopReadAction</code> .

SIP Session Table Usage

The SIP servlet engine's Application Manager reports SIP session table usage to the Overload Policy. The usage is reported as a percent of total table capacity. For example, if the Application Manager reports a value of 85, it means that 85% of total table capacity is currently in use and 15% of it is free. The `SessionTableMaxSizeSip` attribute sets the SIP session table capacity. The default value of this attribute is 70000. The value must be greater than zero.

Not every change in table usage is reported back to overload policy. Below 95%, every 5% change is reported (that is, 0, 5, 10, 15 and so on). From 95% and above, every 1% change is reported (that is, 95, 96, 97 and so on). The Overload Policy MBean enables you to configure the Warning (Table 5–12), Alarm (Table 5–13), and Critical (Table 5–14) thresholds that trigger overload actions.

Table 5–12 lists the attributes that you configure to set the warning thresholds.

Table 5–12 Warning High and Low Threshold Actions for SIP Session Table Usage

Attribute	Description
SipSessionWarningHigh	The high warning threshold value for SIP session table usage that triggers an overload action to decline new connections. This is the default overload action. The range of values is 0-100; 0 disables the action. The default value is 90.
SipSessionWarningLow	The low warning threshold that triggers an end to the overload action. The range of values is 0-100. The default value is 85.
SipSessionWarningActions	A comma-separated list of actions performed when a critical level is reached. The default value is <code>oracle.sdp.networklayer.NetworkServiceImpl\$StopAcceptAction</code> .

Table 5–13 lists the attributes that you configure to set the alarm level thresholds.

Table 5–13 Alarm High and Low Threshold Actions for SIP Session Table Usage

Attribute	Description
SipSessionAlarmHigh	The high alarm threshold of SIP session table usage that triggers an overload action to send a 503 Response (<i>Service Unavailable</i>) to initial requests. This is the default overload action. The range of values is 0-100; 0 disables the action. The default value is 98.
SipSessionAlarmLow	The low alarm threshold of SIP session table usage that triggers an end to the overload action. The range of values is 0-100. The default value is 97.
SipSessionAlarmActions	A comma-separated list of actions performed when an alarm threshold is reached. The default value is <code>com.hotsip.jainsipimpl.javax.sip.context.SipContextImpl\$Send503Action</code> .

Table 5–14 lists the attributes that you configure to set the critical thresholds.

Table 5–14 Critical Threshold Actions for SIP Session Table Usage

Attribute	Description
SipSessionCriticalHigh	The high critical threshold value for SIP session table usage that triggers an overload action to stop reading all connections. This is the default overload action. The range of values is 0-100; 0 disables the action. The default value is 100.
SipSessionCriticalLow	The low critical threshold for SIP session table usage that triggers an end to the overload action. The range of values is 0-100. The default value is 99.
SipSessionCriticalActions	A comma-separated list of actions performed when a critical level is reached. The default value is <code>oracle.sdp.networklayer.NetworkServiceImpl\$StopReadAction</code> .

Deactivating the Overload Protection for System Tuning

Technically, overload protection is always activated. When you tune the system, you can prevent overload actions from executing by doing either of the following:

- Disabling all of the rules by setting all of the trigger values to zero (0).
- Setting the `AppQueueMaxSize`, `StackQueueMaxSize`, and `SipSessionTableMaxSize` attributes to high numbers.

Viewing Statistics and Metrics

This chapter includes the following sections:

- ["Viewing Statistics and Metrics"](#)

Viewing Statistics and Metrics

This section contains the following topics:

- ["SIP Servlet Container Monitor"](#)
- ["Application Counters"](#)
- ["Memory Monitor"](#)
- ["SIP Cluster"](#)

SIP Servlet Container Monitor

The SIP Servlet Container Monitor MBean (`SipServletContainerMonitor`) displays read-only values for system queues and SIP transactions that enable you to assess the performance of the SIP servlet container when it enters an abnormal state. These values can also serve as a reference for system tuning.

The following sections describe SIP Servlet Container Monitor's attributes:

- ["Viewing System Status"](#)
- ["Viewing Transactions"](#)
- ["Using the Current, Peak, and Total Usage Statistics to Tune the System"](#)

Viewing System Status

[Table 6–1](#) lists the attributes that display the current status of the SIP servlet container.

Table 6–1 *Current Status of the SIP Servlet Container*

SIP Servlet Container Status	Attribute
The time that the SIP container was last started.	<code>SipServiceStartTime</code>
The time that the SIP container entered its current state.	<code>SipServiceLastChanged</code>
The number of events dropped by OCMS.	<code>SipMessagesDropped</code>

Viewing Transactions

The `SipServletContainerMonitor` MBean's `SipSummaryTotalTransactions` attribute displays the total number of transactions (both current and completed) as a read-only value. The MBean further delineates transactions by displaying the total number of requests received by the SIP servlet container as well as the total number of subsequent responses that it receives and sends.

Total Requests and Responses

The `SipSummaryInRequests` attribute displays the total number of requests received by the SIP servlet container. In addition to this attribute, the `SipServletContainerMonitor` provides read-only values for the responses that comprise each transaction, both in terms of the total of SIP response messages received and sent by the SIP servlet container (the `SipSummaryInResponses` and `SipSummaryOutResponses` attributes, respectively) and also in terms of the total number of messages from provisional responses (Status Code 1xx) to the final responses (Status Codes 200 - 600). The `SipServletContainerMonitor` displays the total of SIP responses for each message category supported by OCMS (Table 6–2). See RFC 3261 and RFC 3265 for more information on response codes.

Table 6–2 SIP Response Messages Supported by OCMS

Status Code	Reason Phrase	Related Attributes
1xx	Provisional Response Messages	1xx-related attributes: <ul style="list-style-type: none"> <code>SipStatsInfoClassIns</code> <code>SipStatsInfoClassOuts</code>
2xx	Success Messages	2xx-related attributes: <ul style="list-style-type: none"> <code>SipStatsSuccessClassOuts</code> <code>SipStatsSuccessClassIns</code>
3xx	Redirection Response Messages	3xx-related attributes: <ul style="list-style-type: none"> <code>SipStatsRedirClassIns</code> <code>SipStatsRedirClassOuts</code>
4xx	Client Error Responses	4xx-related attributes: <ul style="list-style-type: none"> <code>SipStatsReqFailClassIns</code> <code>SipStatsReqFailClassOuts</code>
5xx	Server Failure Responses	5xx-related attributes: <ul style="list-style-type: none"> <code>SipStatsServerFailClassIns</code> <code>SipStatsServerFailClassOuts</code> <code>503ResponseSent</code>
6xx	Global Failure Messages	6xx-related attributes: <ul style="list-style-type: none"> <code>SipStatsGlobalFailClassIns</code> <code>SipStatsGlobalFailClassOuts</code>
NA	Non-SIP Response Codes	Non-SIP response-related attributes: <ul style="list-style-type: none"> <code>SipStatsOtherClassesIns</code> <code>SipStatsOtherClassesOuts</code>

Provisional Response Messages

As described in RFC 3261, the 1xx SIP response messages indicate informational or provisional responses and are sent when servers expect that obtaining a final response will exceed 200 milliseconds. The *SipStatsInfoClassIns* indicates the total number of 1xx responses received by the SIP servlet container, including transmission. The *SipStatsInfoClassOuts* attribute represents the number of messages sent, relayed, or re-transmitted by the SIP servlet container.

Success Messages

The *SipStatsSuccessClassOuts* and *SipStatsSuccessClassIns* represent the total number of 200 (OK) or 202 (Accepted) response messages sent and received by the SIP servlet container, respectively.

Redirection Response Messages

The 3xx responses provide information about the user's new location, or about alternative services that might be able to satisfy the call. The *SipStatsRedirClassIns* attribute represents the total number of 3xx responses received by (and re-transmitted to) the SIP servlet container. The *SipStatsRedirClassOuts* attribute represents the total number of 3xx responses sent (or re-transmitted) by the SIP servlet container.

Client Error Responses

The 4xx response messages are failure responses issued from server. When a client receives a 4xx response message, it should not attempt to send the request again without modifying it. The *SipStatsReqFailClassIns* attribute represents the total number of client error received by (or re-transmitted to) the SIP Servlet Container. The *SipStatsReqFailClassOuts* attribute represents the total number of client error messages sent (or retransmitted) by the SIP servlet container.

Server Failure Responses

The SIP Servlet Container sends the following error messages:

- 500 Server Internal Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Server Time-out
- 505 Version Not Supported
- 513 Message Too Large

The *SipStatsServerFailClassIns* attribute represents the total number of 5xx response messages received by (or re-transmitted to) the SIP servlet container. The *SipStatsServerFailClassOuts* attribute represents the total number of 5xx response messages sent (or re-transmitted) by the SIP servlet container.

While the *SipStatsServerFailClassOuts* attribute represents all of the 5xx response messages sent by the SIP servlet container, the *503ResponseSent* attribute represents the total number of 503 (*Service Unavailable*) responses sent by the SIP servlet container.

Global Failure Messages

The 6xx responses provide information specific to a user (as opposed to information specific to the instance indicated in the Request-URI). The *SipStatsGlobalFailClassIns* represents the total number of 6xx response messages received by (or re-transmitted)

to the SIP servlet container. The *SipStatsGlobalFailClassOuts* attribute represents the total number of 6xx responses sent (or relayed) by the SIP servlet container.

Non-SIP Response Codes

The *SipStatsOtherClassesIns* and *SipStatsOtherClassesOuts* represent non-SIP response messages (that is, response codes other than the 1xx, 2xx, 3xx, 4xx, 5xx, and 6xx that OCMS supports). The *SipStatsOtherClassesIns* attribute represents the total number of non-SIP responses messages received by (or re-transmitted to) the SIP servlet container. The *SipStatsClassesOuts* attribute represents the total number of non-SIP response messages sent (or relayed) by the SIP servlet container.

Using the Current, Peak, and Total Usage Statistics to Tune the System

This *SipServletContainerMonitor*'s attributes enable you to view the current, peak and total usage for the application and network queues as well as the current and total number of SIP sessions. These read-only values can serve as a reference when you tune overload protection using the [Overload Policy](#) MBean. In particular, you can use the numbers for *ApplicationPeakQueue*, *NetworkPeakQueue*, and *Sessions* to gauge the values for the Overload Policy's *AppQueueMaxSize*, *StackQueueMaxSize*, and *SipSessionTableMaxSize* attributes.

In general, overload protection is not invoked for normal load situations. Use this Mbean to find out the numbers for a normal load by monitoring the system's responses to various test scenarios without executing the overload protection. You can then set the overload protection to start above these figures. See also "[Deactivating the Overload Protection for System Tuning](#)".

Note: The number of 503 responses sent and messages dropped (indicated by the *503ResponseSent* and *MessagesDropped* attributes, respectively) indicate how often overload protection should execute to reduce incoming traffic.

Application Counters

In addition to the counters available to the SIP servlet container (described in "[SIP Servlet Container Monitor](#)"), OCMS enables you to assess application performance through counters (listed in [Table 6–3](#)) that display as read-only values for each deployed SIP application. These counters provide metrics both counter values and range values (*current*, *high*, *low*) are implementations of the following JSR-77 interfaces:

- `javax.management.j2ee.statistics.RangeStatistic`
- `javax.management.j2ee.statistics.CountStatistic`

Table 6–3 Application Counters

Attribute	Description
<code>SipSessions</code>	The current number of SIP sessions. This is a <i>range</i> statistic.
<code>SipApplicationSessions</code>	The total number of SIP sessions. This is a <i>range</i> statistic.
<code>OutResponse</code>	The total number of responses sent by the application.
<code>OutRequest</code>	The total number of SIP application sessions.
<code>TotalSipApplicationSessions</code>	The total number of SIP applications that have been created.
<code>TotalSipSessions</code>	The total number of SIP sessions that have been created.

Table 6–3 (Cont.) Application Counters

Attribute	Description
InResponse	The number of responses received by the application.
InRequest	The number of requests received by the application.

Memory Monitor

The Memory Monitor reports memory usage to the [Overload Policy](#). The Memory Monitor polls the memory status from the runtime environment at either specified or random polling intervals. The Memory Monitor MBean includes attributes that enable you to select the type of polling interval and also the duration of the polling interval. [Table 6–4](#) lists the attributes of the Memory Monitor MBean.

Table 6–4 Attributes of the Memory Monitor MBean

Attribute	Description
AutoStart	Select <i>true</i> to activate the Memory Monitor on startup of the SIP container.
PollingInterval	Enter the time, in seconds, between each interval that the Memory Monitor polls the memory status of the runtime environment. This attribute sets a fixed interval between polls. This value must be greater than five (5) seconds. The default value is 5.
RandomInterval	Select <i>true</i> to set the Memory Monitor to poll at a random intervals. The average length of these intervals will be the same as the value set for the <i>PollingInterval</i> attribute, but individual polls may differ by 50% from the fixed interval. For example, if the polling interval is set to 20 seconds, then a random interval may be pending between 10 and 30 seconds. The default setting is <i>false</i> .
MemoryMonitorStatus	The current status of the Memory Monitor. This value is read-only.
MemoryUsage	The current memory usage. This value is read-only.

Starting and Stopping the Memory Monitor

The start and stop operations enable you to manually start and stop the Memory Monitor.

SIP Cluster

The SIP Cluster MBean (SipCluster) enables you to override the default timeouts set for OC4J clusters. High Availability for OCMS is based on OC4J clusters. Because this framework is based on HTTP, the intervals set for replication are inappropriate for SIP, as they are on the order of seconds rather than milliseconds.

Caution: Because the values set for these timers can significantly impact high availability for OCMS, contact Oracle Support (<http://www.oracle.com/>) support before you change the values for this MBean. The seeded values will suffice for most systems.

The SIPCluster MBean enables you to configure the following timers for state replication between peer nodes:

- *RestoreTimeout* -- If a node cannot recognize the session ID of an incoming request, it requests a state replica from a peer. The interval set by the attribute reflects the time that the node waits for the requested state replica from another peer.

Note: The default value for the restore timeout for OC4J clusters is 0, meaning that the peer would wait indefinitely for the state replica. Without the override provided by this attribute, the peer might block all of its active threads under some circumstances.

- *OwnedByTimeout* -- The time needed to pass ownership of the replicated data from one peer to another. This value sets the interval in which the node owning the state replica drops ownership and the requesting node confirms ownership. Once the requesting peer attains the data, its session replica is promoted to a "live" state.

Note: The default value to pass ownership for OC4J clusters is -1, which instructs the peer not to wait for the replicated data. Because there is no interval between ownership, two peers can concurrently host "live" session replicas for a short period of time.

Configuring Presence and Presence Web Services

This chapter provides an introduction to the Oracle Communication and Mobility Server (OCMS) in the following sections:

- ["Overview of Presence"](#)
- ["Configuring Presence"](#)
- ["Configuring Presence Web Services"](#)
- ["Configuring Scalable Presence Deployments with the User Dispatcher"](#)

Overview of Presence

Presence represents the end-user's willingness and ability to receive calls. Client presence is often represented as a contact management list, which displays user availability as icons. These icons, which not only represent a user's availability, but also a user's location, means of contact, or current activity, enable efficient communications between users.

The Presence application enables a service provider to extend presence service to end users. The application also enables service providers to base other services on presence information. The MBeans registered to the Presence application enable you to configure the presence service, which accepts, stores, and distributes presence information. See also ["Presence Server"](#) in [Chapter 1, "An Overview of Oracle Communication and Mobility Server"](#).

The Presence application MBeans enable you to manage the following:

- [Presence Status Publication](#)
- [Presence Status Subscriptions](#)
- [Watcher-Info Support](#)
- [Presence XDMS Authorization of Subscriptions](#)
- [Privacy Filtering](#)
- [Presence Hard State](#)
- [Composition of Multiple Presence Sources](#)

Presence Status Publication

A presentity can publish a PIDF (Presence Information Data Format) document containing presence state to the Presence Server.

Presence Status Subscriptions

The Presence server supports subscriptions to a user's status. The Presence Server notifies the user when the watcher (subscriber) is authorized to view the user's status. The Presence server also notifies all of the active, authorized watchers of the publication of a new presence document.

Watcher-Info Support

The Presence Server enables the user who is publishing presence information to subscribe to watcher-info events to receive information on all watchers currently subscribing to the user's presence information. The Presence Server also notifies users of changes in the watcher subscriptions, such as new or terminated subscriptions.

Presence XDMS Authorization of Subscriptions

Whenever a watcher subscribes to a user's presence, the Presence Server checks the authorization policy that the publisher has set to see if the subscriber has the required authorization.

If no matching rule can be found, the subscriber is put in a pending state and a watcher info notification is sent to the publisher. Usually, the publisher's client (User Agent) presents a pop-up box asking whether to accept or reject a new pending subscriber. The answer is added to the publisher's authorization policy document in the form of a rule for this subscriber. The document is then updated by the client on the XDMS using HTTP. When the document is updated, the Presence Server reads the new policy document and acts on the new rule, changing the subscription state accordingly.

Privacy Filtering

A user can create privacy filtering rules to allow or block a user.

Presence Hard State

The hard state feature enables a user to leave a document in the XDMS that notifies watchers when there are no other documents. In general, this feature is used for leaving an off-line note, such as "On Vacation".

Composition of Multiple Presence Sources

If a user has two or more clients (such as a PC and a mobile phone) both publishing presence documents, the Presence Server combines two or more documents into a unified document as dictated by a composition policy. The Presence server supports two different composition policies: a default policy and a policy that performs composition according to the OMA (Open Mobile Alliance) Presence enabler.

The default composition policy is a simple, but robust, algorithm. It adds `<dm:timestamp>` elements to the `<dm:person>` and `<dm:device>` elements if they are missing, and `<pidf:timestamp>` elements to the `<pidf:tuple>` elements if they are missing.

When the Presence Server creates the candidate document, it includes all `<pidf:tuple>` and `<dm:device>` elements from the source documents. It includes only one `<dm:person>` element in the candidate document, and uses the latest published element based on the `<dm:timestamp>` element. All other `<dm:person>` elements are ignored.

Configuring Presence

Configuring the following MBeans enables Presence:

- [Bus](#)
- [PackageManager](#)
- [Presence](#)
- [PresenceApplicationDeployer](#)
- [PresenceEventPackage](#)
- [PresenceWInfoEventPackage](#)
- [UA-ProfileEventPackage](#)
- [UserAgentFactoryService](#)

Configuring XDMS

The following MBeans enables you to configure the XDMS (XML Document Management Server):

- [Command Service \(XDMS Provisioning\)](#)
- [XCapConfig](#)

Note: If you change any attributes of the following MBeans, you must restart OCMS for these changes to take effect.

- [Presence](#)
 - [PresenceEventPackage](#)
 - [PresenceWInfoEventPackage](#)
 - [UAProfileEventPackage](#)
 - [XCAPConfig](#)
-
-

Bus

The Bus MBean supports presence by setting the thread pool, the high and low watermarks for the job queues, and the duration that a job remains in the queue before notifications are dispatched. [Table 7-1](#) describes the attributes of the Bus MBean.

Table 7-1 *Attributes of the Bus MBean*

Attribute	Value Type	Description
HighWatermark	int	The number of pending jobs reached before the bus's exhausted threshold level is reached. The default value is 20.
KeepAlive	long	The number of seconds to keep an idle thread alive before dropping it (if the current number of threads exceeds the value specified for <i>MinThreads</i>). The default value is 60.
LogDuration	long	The duration, in seconds, that an event remains in the queue. A warning is logged to the system log for events that remain in the queue for a period exceeding the specified duration before they are broadcast to the bus. This warning indicates that server is about to be overloaded, since an old job has been sent to the bus. The default value is 60.

Table 7-1 (Cont.) Attributes of the Bus MBean

Attribute	Value Type	Description
LowWatermark	int	Specifies the low threshold level for the number of pending jobs. When this threshold is reached from below, the Bus logs a warning that it is about to be choked. At this point, no more warnings are logged until the high watermark level is reached. The default value is 15.
MinThreads	int	The minimum number of threads held in the thread pool. If no threads are used, then the specified number of threads remains in an idle state, ready for upcoming jobs. The default value is 15.
MaxThreads	int	The maximum number of threads held in the thread pool. When the specified number of threads are occupied with jobs, subsequent jobs are placed in a queue and are dealt with as the threads become available. The default value is 10.

PackageManager

The [PresenceEventPackage](#), [PresenceWInfoEventPackage](#), and [UA-ProfileEventPackage](#) MBeans enable you to configure the event packages, which define the state information to be reported by a notifier to a watcher (subscriber). These packages form the core of the Presence Server, as most requests flow through them.

A notifier is a User Agent (UA) that generates NOTIFY requests that alert subscribers to the state of a resource (the entity about which watchers request state information). Notifiers typically accept SUBSCRIBE requests to create subscriptions. A watcher is another type of UA, one that receives the NOTIFY requests issued by a notifier. Such requests contain information about the state of a resource of interest to the watcher. Watchers typically also generate SUBSCRIBE requests and send them to notifiers to create subscriptions.

The PackageManager MBean sets the configuration for the PresenceEventPackage, WatcherinfoPackage, and UA-ProfileEventPackage MBeans. [Table 7-2](#) describes the attributes of the PackageManager MBean.

Table 7-2 Attributes of the EventPackages MBean

Attribute	Description
CaseSensitiveUserPart	Setting this attribute to <i>true</i> enables case-sensitive handling of the user part of the SIP URI. If this parameter is set to <i>false</i> , then the user part of the URI is not a case-sensitive match. For example, <i>foo</i> is considered the same as <i>FoO</i> . The domain part of the URI is always case-insensitive.
EventPackageNames	A comma-separated list of event package names. For example: <i>presence,presence.winfo,ua-profile</i> .
WaitingSubsCleanupInterval	The interval, in seconds, in which the subscription cleanup check runs. The thread sleeps for this period and then awakens to check for any waiting subscriptions with a timestamp older than the <i>MaxWaitingSubsTimeHours</i> parameter. All old subscriptions are then removed from the subscribed resource.

Table 7–2 (Cont.) Attributes of the EventPackages MBean

Attribute	Description
Max WaitingSubsTimeHours	The maximum time, in hours, that a subscription can be in a waiting state before the server removes it. This parameter is used by the subscription cleanup check thread (<code>waitingsubscleanupinterval</code>) to decide if a waiting subscription is old enough to be removed from the subscribed resource.

Presence

The Presence MBean controls how the Presence Server interacts with presentities, Publish User Agents (PUAs) that provide presence information to presence services. The attributes (described in [Table 7–3](#)) include those for setting the composition policy for creating a unified document when a user publishes presence documents from two or more clients, as well as setting the blocking, filtering, and presence hard state.

Table 7–3 Attributes of the Presence MBean

Attribute	Description/Value
CompositionPolicyFilename	The filename of the composition policy document. Values include <code>compose.xslt</code> , for the OCMS composition policy, and <code>compose_OMA.xslt</code> , for the OMA composition policy.
DefaultSubHandling	The default subscription authorization decision that the server makes when no presence rule is found for an authenticated user. The defined values are: <ul style="list-style-type: none"> ▪ block ▪ confirm ▪ polite-block Unauthenticated users will always be blocked if no rule is found. For more information about this, see <i>Chapter 3.2.1: Subscription Handling</i> in the IETF SIMPLE draft for presence rules (http://www.ietf.org/internet-drafts/draft-ietf-simple-presence-rules-04.txt).
DocumentStorageFactory	The name of the DocumentStorage Factory Class. The default value is <code>oracle.sdp.presenceeventpackage.document.XMLDocumentStorageFactoryImpl</code> .
DocumentStorageRootUrl	The system identifier for the document storage. In the file storage case, this is the root file URL path where documents are stored. The content of this directory should be deleted when the server is restarted. The default value is <code>file:/tmp/presencestorage/</code> .
DocumentStorageType	The type of storage to be used for presence documents. If the number of users is large, Oracle recommends that you store the presence documents on file. Valid values: <ul style="list-style-type: none"> ▪ file ▪ memory The default value is <i>memory</i> .
HttpAssertedIdentityHeader	The type of asserted identity header used in all HTTP requests from the Presence Server to the XDMS. Set the value of this attribute to one expected by the XDMS. Valid values: <ul style="list-style-type: none"> ▪ X_3GPP_ASSERTED_IDENTITY ▪ X_3GPP_INTENDED_IDENTITY ▪ X_XCAP_ASSERTED_IDENTITY (The default value.)

Table 7–3 (Cont.) Attributes of the Presence MBean

Attribute	Description/Value
PidfManipulationAuid	The ID of the application usage for PIDF (Presence Information Data Format) manipulation. The default value is <i>pidf-manipulation</i> .
PidfManipulationDocumentName	The document name for pidf manipulation application usage. For example: <i>hardstate</i> . Unauthenticated users are blocked when no rule is found. If the URI contains a domain name instead of an IP address, then you must configure the DNS Server. The default value is <i>hardstate</i> .
PidfManipulationEnabled	Set to <i>true</i> (the default value) to enable PIDF manipulation.
PidfManipulationXcapUri	The SIP URI of the XDMS for the pidf manipulation application usage. The default value is: <i>sip:127.0.0.1;transport=TCP;lr</i> . The loose route (<i>lr</i>) parameter must be included in the SIP URI for the server to function properly.
PoliteBlockPendingSubscription	Set to <i>true</i> if pending subscriptions should be polite-blocked. This feature is used to hide the presentity from the presence watcher with a pending subscription and instead send them fake presence documents. If set to <i>false</i> the subscriptions will remain as pending.
PresRulesAuid	The ID of the application usage for presrules. The default is <i>pres-rules</i> .
PresRulesDocumentName	The document name for presrules application usage. The default value is <i>presrules</i> .
PresRulesXcapUri	The SIP URI of the XDMS for the presence rules application usage. The default value is: <i>sip:127.0.0.1;transport=TCP;lr</i> . The loose route (<i>lr</i>) parameter must be included in the SIP URI for the server to function properly.
PrivacyFilteringEnabled	Set to <i>true</i> to enable privacy filtering. Set to <i>false</i> to disable filtering. If privacy filtering is disabled, then all subscriptions that are allowed to see a user's presence will always see everything that has been published for the presentity.
TransformerFactory	The name of the TransformerFactory class. The default value is <code>oracle.xml.jaxp.JXSAXTransformerFactory</code> .

PresenceEventPackage

Table 7–4 describes the attributes of the PresenceEventPackage MBean. The presence event package has two subgroups: publish and subscribe. Each subgroup has a `minexpires` and a `maxexpires` parameter to set the interval of the expiry of a publication or a subscription that is accepted by the Presence Server. A client states when its publication or subscription expires. If a client sends an expiry time that is lower than the configured `minexpires` time, the server returns a 423 (*Subscription Too Brief*) response. If a client sends an expires time that is higher than the configured `maxexpires` time, the server returns the `maxexpires` time in the response. To keep a publication or subscription alive, the client sends `republish` or `resubscribe` to the server within the expiry time. The client must perform this task repeatedly through the lifetime of the publication or subscription.

Table 7–4 Attributes of the PresenceEventPackage

Attribute	Value/Description
Description	A description of the PresenceEventPackage. For example: <i>The event package that enables presence.</i>
DocumentFactory	The DocumentFactory class name. The default value is <code>oracle.sdp.presenceeventpackage.document.PresenceDocumentFactoryImpl</code> .
EscMaxDocumentSize	The maximum size, in bytes, for the contents of a publication. If a client attempts to publish a document that is larger than the specified size, the server sends the 413 response, <i>Request entity too long</i> . The default value is 10000.
ESCMaxExpires	The maximum time, in seconds, for a publication to expire. The default value is 3600.
ESCMaxPubPerRes	The maximum number of publications allowed per resource. If the maximum number has been reached for a resource when a new publish is received, the server sends the 503 Response (<i>Service Unavailable</i>).
ESCMinExpires	The minimum time, in seconds, for a publication to expire. The default is 60.
EventStateCompositor	The class name of the EventStateCompositor. The default value is <code>oracle.sdp.presenceeventpackage.PublishControl</code> .
Name	The name of this event package. The default value is <i>Presence</i> .
Notifier	The name of the Notifier class. The default value is <code>oracle.sdp.presenceeventpackage.PresenceSubscriptionControl</code> .
NotifierMaxDocumentSize	The maximum size for a SUBSCRIBE.
NotifierMaxExpires	The maximum time, in seconds, for a SUBSCRIBE to expire. The default is 3600.
NotifierMaxNoOfSubsPerRes	The maximum number of subscriptions allowed per resource. If the maximum number has been reached for a resource, then a new presence subscribe is received and the server sends the 503 Response (<i>Service Unavailable</i>).
NotifierMinExpires	The minimum time, in seconds, for a SUBSCRIBE to expire.
ResourceManagerClassName	The name of the ResourceManager class. The default is <code>oracle.sdp.presenceeventpackage.PresenceEventManagerImpl</code> .

PresenceWInfoEventPackage

As described in RFC 3857, a Watcher Information Event Package monitors the resources in another event package to ascertain the state of all of the subscriptions to that resource. This information is then sent to the subscriptions of the Watcher Information Event Package. As a result, the subscriber learns of changes in the monitored resources subscriptions.

The PresenceWInfoEventPackage MBean (described in [Table 7–5](#)) sets the subscription state information for the Watcher Information Event Package.

Table 7–5 Attributes of the WatcherinfoEventPackage

Attribute	Description/Value
Description	A description of the PresenceWInfoEventPackage. For example: <i>The event package that enables watcherinfo.</i>
DocumentFactory	The name of the DocumentFactory class. The default is <code>oracle.sdp.eventnotificationsservice.DocumentFactoryImpl</code> .
Name	The name of the event package. The default value is <i>presence.wininfo</i> .
Notifier	The Notifier class name. The default value is <code>oracle.sdp.presenceeventpackage.PresenceSubscriptionControl</code> .

Table 7–5 (Cont.) Attributes of the WatcherinfoEventPackage

Attribute	Description/Value
NotifierMaxDocumentSize	The maximum document size for SUBSCRIBE.
NotifierMaxExpires	The maximum time, in seconds, for a SUBSCRIBE to expire. The default is 3600.
NotifierMaxNoSubsPerRes	The maximum number of subscriptions allowed per resource. If the maximum number has been reached for a resource when a new presence subscribe is received, the server will send a 503 Response (<i>Service Unavailable</i>). The default value is 100.
NotifierMinExpires	The minimum time, in seconds, for a SUBSCRIBE to expire.
ResourceManagerClassName	The name of the ResourceManager class. The default is <code>oracle.sdp.winforeventpackage.WatcherinfoResourceManager</code> .

UA-ProfileEventPackage

Table 7–6 describes the attributes of the UA-ProfileEventPackage MBean.

Table 7–6 Attributes of the UA-Profile Event Package

Attributes	Description/Value
Description	A description of the UA-ProfileEventPackage. The default value is <i>The event package that enables the ua-profile</i> .
Document Factory	The Document Factory class name. The default value is: <code>oracle.sdp.eventnotificationsservice.DocumentFactoryImpl</code>
Name	The name of the event package. The default value is <i>ua-profile</i> .
Notifier	The name of the Notifier class. The default value is: <code>oracle.sdp.presenceeventpackage.PresenceSubscriptionControl</code>
NotifierMaxDocumentSize	The maximum document size for a SUBSCRIBE.
NotifierMaxExpires	The maximum time, in seconds, for a SUBSCRIBE to expire. The default is 6000.
NotifierMaxNoOfSubsPerRes	The maximum number of subscriptions allowed per resource. If the maximum number has been reached for a resource when a new presence subscribe is received, the server will send a 503 Response (<i>Service Unavailable</i>). The default value is 100.
NotifierMinExpires	The minimum time, in seconds, for a SUBSCRIBE to expire. The default value is 60.
ResourceManager	The name of the Resource Manager class. The default value is: <code>oracle.sdp.winforeventpackage.WatcherinfoResourceManager</code>

UserAgentFactoryService

The UserAgentFactoryService MBean sets the commands for user agent factory service. The Presence Server uses the user agent factory to subscribe to changes in XML documents stored in the XDMS for presence.

Table 7–7 Attributes of the UserAgentFactoryService MBean

Attribute Name	Description/Value
DNSNames	A comma-separated list of DNS (Domain Name System) IP addresses used by the user agent.
IpAddress	The IP address for the user agent client; use the empty string (the default setting) for the default network interface on the current system.
PreferredTransport	The preferred transport protocol that enables communication between the Presence Server and the XDMS. The default value is TCP. Valid values are TCP and UDP.
Port	The IP port for the user agent client. The default value is 5070.

Command Service (XDMS Provisioning)

The Command Service MBean enables user provisioning to the XDMS. For more information see "[CommandService](#)".

XCapConfig

The XCapConfig MBean controls the configuration of the XDMS, the repository of the XCAP (Extensible Markup Language Configuration Access Protocol) documents containing user presence rules (pres-rules) and hard state information. The XCapConfig MBean settings can be ignored if the XDMS is external to OCMS.

Table 7–8 Attributes of the XCapConfig MBean

Attribute Name	Description/Value
CreateNonExistingUserstore	Set to <i>true</i> to create a user store if one does not exist when storing a document; otherwise, set to <i>false</i> . If the parameter is set to <i>false</i> and a client tries to store a document for a user that does not exist, then the store fails. If the parameter is set to <i>true</i> , then the user will first be created in the XDMS and then the document will be stored. The default value is <i>true</i> .
MaxContentLength	The maximum size, in bytes, for an XDMS document. Although Oracle recommends a default maximum size per XDMS document of 1 MB (1000 contacts at about 1 KB each), you can increase or decrease the size of the document. If you increase the document size, then you must be sure to that there is sufficient disk space to accommodate the XDMS document * the number of users * the number of applications. If you set a smaller per-document size, then this calculation is reduced to the sum of (max_doc_size_n * number of users) where each max_doc_size_n is specific to application n. The default size for the resource-lists document is also 1 MB.
PersistenceRootUrl	The persistent storage location. Use the default value <i>jpa:oc4j</i> if you are running a single node instance. This provides for default caching. Use the value <i>jpa:multinode</i> if you are running a multinode presence topology that includes a presence server running on a single instance.
PidfManipulationAuid	The ID of the application usage for PIDF (Presence Information Data Format) manipulation. The default value is <i>pidf-manipulation</i> .
PidfManipulationDocname	The document name for pidf manipulation application usage. For example: <i>hardstate</i> . Unauthenticated users are blocked when no rule is found. If the URI contains a domain name instead of an IP address, then you must configure the DNS Server. The default value is <i>hardstate</i> .
PresRulesAU	The name of the pres-rules application usage. The default value is <i>pres-rules</i> .

Table 7–8 (Cont.) Attributes of the XCapConfig MBean

Attribute Name	Description/Value
PresRulesDocName	The name of the pres-rules document. The default value is <i>presrules</i> .
PublicContentServerRootUrl	The URL to the public content server root. The URL must be set to the public URL of the content server (that is, the URL of the authentication HTTP proxy server).
PublicXCAPRootUrl	<p>The URL to the public XDMS root, entered as <i>http://<your.xdms.domain.com>/services/</i>. For example, enter <i>http://127.0.0.1:8080/services</i>. The URL defined in this parameter gives clients the location of the content server (which can be on a separate server from the XDMS). The XDMS places this URL in the <i>Content-Type</i> header of its outgoing NOTIFY messages. For example, the <i>Content-Type</i> header in the following NOTIFY message from the XDMS to the Presence Server notes that the body of the pres-rules document is stored externally and also includes instructions within the URL for retrieving the document.</p> <pre> CSeq: 1 NOTIFY From: <sip:bob_0@144.22.3.45>;tag=66910936-0e31-41b2-abac-10d7616d04ef To: <sip:bob_0@144.22.3.45>;tag=ffa3e97bd77f91e6ca727fbf48a5678b Content-Type: message/external-body;URL="http://127.0.0.1:8888/contentserver/pres-rules/users/bob_0@144.22.3.45/presrules";access-type="URL" ... Event: ua-profile;document="pres-rules/users/sip:bob_0@144.22.3.45/presrules";profile -type=application;aid="pres-rules" </pre>
RequireAssertedIdentity	Set to <i>true</i> if all HTTP/XDMS requests require an asserted identity header; otherwise, set this parameter to <i>false</i> . Setting this attribute to <i>true</i> requires all XCAP traffic to be authenticated by the Aggregation Proxy . If this attribute is set to <i>true</i> , then any incoming XCAP request that lacks an asserted identity is denied access.

Configuring Presence Web Services

OCMS enables Web Service clients to access presence services through its support of the Parlay X Presence Web Service as defined in *Open Service Access, Parlay X Web Services, Part 14, Presence ETSI ES 202 391-14*. A Parlay X Web Service enables an HTTP Web Service client to access such presence services as publishing and subscribing to presence information. The Parlay X Presence Web Service does not require developers to be familiar with the SIP protocol to build such a Web-based client; instead, Parlay X enables Web developers can build this client using their knowledge of Web Services.

The Presence Web Services application, which is deployed as a child application of the Presence application, contains the following MBeans that enable you to configure a Web Services deployment server:

- [Bus](#)
- [PackageManager](#)
 - [PresenceEventPackage](#)
 - [PresenceWInfoEventPackage](#)
 - [UA-ProfileEventPackage](#)
- [PresenceWebServiceDeployer](#)
- [PresenceSupplierWebService](#)
- [PresenceConsumerWebService](#)

- [UserAgentFactoryService](#)
- [XCapConfig](#)

The Presence Web Services application also includes the [PresenceSupplierWebService](#) and [PresenceConsumerWebService](#) MBeans, which contain attributes for managing presence publication and watcher subscriptions enabled through the OCMS implementation of Presence Consumer and Presence Supplier interfaces.

PresenceWebServiceDeployer

Starts the JMX framework for the Presence Web Services application and deploys all of its Model MBeans. The operations of the PresenceWebServiceDeployer MBean enable you to retrieve information of the objects exposed by the Presence Web Service to this MBean.

Table 7–9 Operations of the PresenceWebServiceDeployer MBean

Operation	Description
getManagedObjectNames	Returns a String array containing the object names of the deployed application.
getMBeanInfo	Returns the meta-data for the deployed MBean.
getMBeanInfo (locale)	Returns the localized meta-data for the deployed Mbean.

PresenceSupplierWebService

The PresenceSupplierWebService MBean (described in [Table 7–10](#)) enables you to manage the presence data published to watchers.

Table 7–10 Attributes of the PresenceSupplierWebService MBean

Attributes	Description
Expires	The default expiry time, in seconds, for the PUBLISH of a presence status. The value entered for this attribute should be optimized to match that entered for the <i>SessionTimeout</i> attribute.
PIDFManipulationAU	The name of the application usage for PIDF (Presence Information Data Format) manipulation. The default value is <i>pidf-manipulation</i> .
PidfManipulationDocname	The document name for pidf manipulation application usage. For example: <i>hardstate</i> . Unauthenticated users are blocked when no rule is found. If the URI contains a domain name instead of an IP address, then you must configure the DNS Server. The default value is <i>hardstate</i> .
PresRulesAU	The name of the pres-rules application usage. The default value is <i>pres-rules</i> .
PresRulesDocname	The name of the pres-rules document. The default value is <i>presrules</i> .
PublicXCAPRootUrl	The URL to the public XDMS root, entered as <i>http://<your.xdms.domain.com>/services/</i> . For example, enter <i>http://127.0.0.1:8080/services</i> .
SessionTimeout	The timeout of the HTTP session, in seconds. The value entered for this attribute should be optimized to match the value entered for the <i>Expires</i> attribute. This timeout takes effect for new sessions only.

Table 7–10 (Cont.) Attributes of the PresenceSupplierWebService MBean

Attributes	Description
SIPOutboundProxy	<p>The IP address of the outbound proxy server where all requests are sent on the first hop. Enter this address in the following format:</p> <pre>sip:<IP address>;lr;transport=TCP</pre> <p>You can also enter the default port (5060) in this address. For example, enter <code>sip:127.0.0.1:5060;lr;transport=TCP</code>. The shortest format for entering this address is <code>sip:127.0.0.1;lr</code>.</p> <p>If you do not define this attribute, then no outbound proxy will be used.</p>

PresenceConsumerWebService

The PresenceConsumerWebService MBean (described in [Table 7–11](#)) enables you to set the duration of watcher subscriptions.

Table 7–11 Attributes of the PresenceConsumerWebService MBean

Attribute	Value
Expires	The default expiry time, in seconds, for watcher subscriptions. The value entered for this attribute should be optimized to match the value entered for the <i>SessionTimeout</i> attribute.
SessionTimeout	The timeout of the HTTP session, in seconds. The value entered for this attribute should be optimized to match the value entered for the <i>Expires</i> attribute. This timeout takes effect for new sessions only.
SIPOutboundProxy	<p>The IP address of the outbound proxy server where all requests are sent on the first hop. Enter this address in the following format:</p> <pre>sip:<IP address>;lr;transport=TCP</pre> <p>You can also enter the default port (5060) in this address. For example, enter <code>sip:127.0.0.1:5060;lr;transport=TCP</code>. The shortest format for entering this address is <code>sip:127.0.0.1;lr</code>.</p> <p>If you do not define this attribute, then no outbound proxy will be used.</p>

Aggregation Proxy

The Aggregation Proxy is a server-side entry point for OMA clients that authenticates any XCAP traffic and Web Service calls (which are conducted through HTTP, not SIP) by providing identity assertion. This component acts as the gatekeeper for the trusted domain that houses the Presence Server and the XDMS.

The Parlay X Web Service operates within a trusted domain where the Aggregation Proxy authorizes the user of the Web Service. It authenticates XCAP traffic and Web Service calls emanating from a Parlay X client by inserting identity headers that identify the user of the Web Services. The Aggregation Proxy then proxies this traffic (which is sent over HTTP) to the Parlay X Web Service and XDMS.

The attributes of the Aggregation Proxy MBean ([Table 7–12](#)) enable you to set the type of identity assertion that is appropriate to the XDMS. In addition, you set the host and port of the Web Server and XDMS that receive the proxied traffic from the Aggregation Proxy.

Table 7–12 Attributes of the Aggregation Proxy

Attribute	Description
AssertedIdentityType	Enter the number corresponding to the identity header inserted into proxied HTTP requests that is appropriate to the XDMS: <ol style="list-style-type: none"> 1. X_3GPP_ASSERTED_IDENTITY (the default) 2. X_3GPP_INTENDED_IDENTITY 3. X_XCAP_ASSERTED_IDENTITY
ContentHost	Hostname of the Content Server where the Aggregation Proxy sends proxied requests.
ContentPort	The port number of the Content Server where the Aggregation Proxy sends proxied requests.
ContentRoot	The root URL of the Content Server.
IgnoreUserpartCase	Set to <i>true</i> if case-sensitive handling of the user name is not required.
JAASLoggingContext	The name for the JAAS (Java Authentication and Authorization Service) <code>javax.security.auth.login.LoginContext</code> .
JAASRoles	A comma-separated list of JAAS roles for authentication. If the value is <code>"*"</code> , it will allow all JAAS roles.
PresenceConsumerEndpoint	<i>Note:</i> this attribute is deprecated and is only here for backward compatibility. The path to the endpoint of the Presence Consumer Web Service. The methods of the Presence Consumer interface enable watchers to obtain presence data.
PresenceSupplierEndpoint	<i>Note:</i> this attribute is deprecated and is only here for backward compatibility. The path to the endpoint of the PresenceSupplier Web Service. The methods of the Presence Supplier Interface enable presentities to provide presence manage the data accessed by watchers.
TrustedHosts	A comma-separated list of IP addresses of trusted hosts. Asserted identity headers are removed from requests with addresses that are not included in this list.
WebServiceHost	<i>Note:</i> this attribute is deprecated and is only here for backward compatibility. The host name of the Web Services deployment server to which the Aggregation proxies requests.
WebServicePort	<i>Note:</i> this attribute is deprecated and is only here for backward compatibility. The port of the Web Services deployment server to which the Aggregation proxies requests.
XCAPHost	The host name of the XDMS to which the Aggregation Proxy proxies requests.
XCAPPort	The port of the XDMS to which the Aggregation Proxy proxies requests.
XCAPRoot	The root URL of the XDMS.

Configuring the Aggregation Proxy to Work with Realms

You can configure the Aggregation Proxy to work with one or more realms.

Perform the following steps:

1. Select **aggregationproxy > Administration > Security Provider > OCMSLoginModule > Edit**.
Five attributes are displayed, the most important of which is the *realm*.
2. Configure the realm or realms as a comma-separated list in the following format:
`<domain>=<realm>,<domain>=<realm>,...`

Securing the XDMS with the Aggregation Proxy

Secure the XDMS by deploying it behind the [Aggregation Proxy](#). Access to the XDMS should be restricted only to the Aggregation Proxy and the Presence Server. In addition, securing the XDMS requires that you configure the Presence Server application's [XCAPConfig](#) MBean, the Aggregation Proxy and the Oracle Communicator as follows:

- Deny access to any incoming XCAP request that lacks an asserted identity header by setting the value of the *RequiredAssertedIdentity* attribute of Presence Server's [XCAPConfig](#) MBean to *true*. Setting this attribute to *true* requires authentication of all XCAP by the Aggregation Proxy.
- Set the appropriate XDMS-related values for the *XCAPIHost*, *XCAPPort*, *XCAPRoot*, *ContentHost*, *ContentPort* and *ContentRoot* attributes of the Aggregation Proxy MBean.
- Configure the Oracle Communicator's XDMS settings in `customize.xml` to point to the Aggregation Proxy -- not to the XDMS -- by defining the `<RootContext>` element as `aggregationproxy`, the context root of the Aggregation Proxy and by setting the `<host>` and `<port>` elements to the host of the Aggregation Proxy and the HTTPS port on that host, such as 443.

The Aggregation Proxy must be deployed as a child application of [Subscriber Data Services](#). You can bind to the `default-web-site` for HTTP. To enable HTTP over SSL, you must configure the OC4J Container on which the Aggregation Proxy executes to provide HTTPS. Refer to *Oracle Containers for J2EE Security Guide* for instructions on configuring HTTPS. To enable access to the Aggregation Proxy over HTTPS, bind the Aggregation Proxy with the `secure-web-site`. Ensure that the Presence Server binds with the `default-web-site` if it resides on the same server with the Aggregation Proxy. Because the Presence Server resides in the `presence.ear` file, all of the HTTP servlets in that EAR file must bind to `default-web-site`.

Configuring Scalable Presence Deployments with the User Dispatcher

In non-distributed environments, stateful applications function properly because they receive requests from a single node. In distributed environments where applications must be scaled over multiple nodes to accommodate traffic, stateful applications may fail because any node can serve a request, not just to the one running the application that maintains the session state for the request. The User Dispatcher guarantees that SIP and HTTP user requests are dispatched to the node that maintains the session state needed to successfully process that request; once user requests are directed to the User Dispatcher, they are consistently sent to the same destination.

Failover

Fail-over is a technique that can be used by the User Dispatcher to assert a higher level of availability of the Presence Server. Since the Presence server does not replicate any state (such as established subscriptions) the state has to be recreated by the clients on the new server node by setting up new subscriptions. Also, since a subscription is a

SIP dialog and the User Dispatcher is not record routing, it cannot fail-over a subscription from one node to another. All subsequent requests will follow the route set and end up on the old node.

This is not a problem when failing over from a failing server since that node is not processing the traffic anyway and any request within a dialog will eventually get a fail response or timeout and the dialog will be terminated. However, when migrating back a user from the backup node to the original node (when it has been repaired), which has to be done to maintain an even distribution after the failure, this is a problem that can lead to broken presence functionality. The only way to migrate a subscription from one running server to another is to either restart the client or the server.

However, the server that holds the subscription can actively terminate it by sending out a terminating NOTIFY and discarding the subscription state. This will force the client to issue a new initial SUBSCRIBE to establish a new dialog. For a subscription to migrate from one live node to another the User Dispatcher must fail-over the traffic (which is only affecting initial requests) and instruct the current server to terminate the subscriptions.

Presentity Migration

Presentities must be migrated when the set of nodes have changed. This involves having the Presence application to terminate some or all subscriptions to make the migration happen.

Stateless User Dispatcher and Even Distribution The most basic approach is to contact the Presence application on all nodes to terminate all its subscriptions. The problem with this is that a burst of traffic will be generated although spread out over a period of time. This time period results in incorrect presence states since the longer the termination period is the longer it will take until all users get a correct presence state.

To optimize this you could terminate only those subscriptions that actually need to be terminated (the ones that has been migrated). The problem is that the User Dispatcher does not know which users these are (since it does stateless distribution based on an algorithm) and the Presence application does not either (since it only knows what users it has). However, if the Presence application could iterate over all its subscriptions and for each of them ask the User Dispatcher if this user would go to this Presence node, then the Presence server could terminate only those that will not come back to itself. This may be a heavy operation, but under the constraint that each Presence server is collocated with a User Dispatcher each such callback would be within the same JVM.

Presence Application Broadcast Another solution is to have the Presence servers guarantee that a user only exists on one Presence node at any given time. This can be done by having the Presence application broadcast a message to all its neighbors when it receives a PUBLISH or SUBSCRIBE for a new presentity (a presentity that it does not already have a state for). If any other Presence node that receives this broadcast message already has active subscriptions for this presentity, that server must terminate that subscription so that the client can establish a new subscription with the new server.

With this functionality in the Presence application, the User Dispatcher would not have to perform additional steps to migrate a user from one live node to another.

Standby Server Pool

Another approach is to have a standby pool of servers that are idling ready to take over traffic from a failing node. When an active node fails the User Dispatcher will

redistribute all its traffic to one server from the standby pool. This node will now become active and when the failing node eventually is repaired it will be added to the standby pool. This will eliminate the need for migrating users back from a live node when a failing node resumes.

This approach requires more hardware and the utilization of hardware resources will not be optimal.

Failure Types

There are several types of failures that can occur in a Presence server and different types of failures may require different actions from the User Dispatcher.

Fatal Failures If the failure is fatal all state information is lost and established sessions will fail. However, depending on the failure response, subscriptions (presence subscribe sessions) can survive using a new SIP dialog. If the response code is a 481 the presence client must according to RFC 3265 establish a new SUBSCRIBE dialog and this is not considered to be a failure from a presence perspective. All other failure responses may (depending on the client implementation) be handled as an error by the client and should therefore be considered a failure.

After a fatal failure the server does not have any dialog states from the time before the failure, which means that all subsequent requests that arrive at this point will receive a 481 response back. During the failure period all transactions (both initial and subsequent) will be terminated with a non-481 error code, most likely a 500 or an internal 503 or 408 (depending on if there is a proxy in the route path or not, and what the nature of the failure is).

Typically a fatal failure will result in the server process or the entire machine being restarted.

Temporary Failures A temporary failure is one where none or little data is lost so that after the failure session states will remain in the server. This means that a subsequent request that arrives after the server has recovered from the failure will be processed with the same result, as it would have been before the failure.

All requests that arrive during the failure period will be responded with a non-481 failure response, such as 503.

In general a temporary failure has a shorter duration, and a typical example is an overload situation in which case the server will respond 503 on some or all requests.

Failover Actions

The User Dispatcher can take several actions when it has detected a failure in a Presence server node. The goal with the action is to minimize the impact of the failure in terms of number of failed subscriptions and publications and the time it takes to recover. In addition to this the User Dispatcher needs to keep the distribution as even as possible over the active servers.

The fail-over action to be used in this version of the User Dispatcher is to disable the node in the pool. This approach is better than removing the node because when the `ResizableBucketServerPool` is used since the add and remove operations are not deterministic. This means that the result of adding a node depends on the sequence of earlier add and delete operations, whether as the disable operation will always result in the same change in distribution given the set of active and disabled nodes.

Overload Policy

An activated overload policy can indicate several types of failures but its main purpose is to protect from a traffic load that is too big for the system to handle. If such a situation is detected as a failure, fail-over actions can lead to bringing down the whole cluster since if the distribution of traffic is fairly even all the nodes will be in or near an overloaded situation. If the dispatchers remove one node from the cluster and redistribute that node's traffic over the remaining nodes they will certainly enter an overload situation that causes a chain reaction.

Since it is difficult to distinguish this overload situation from a software failure that triggers the overload policy to be activated even though the system is not under load, it might still be better to take the fail-over action unless Overload Policy is disabled. If the system is really in an overload situation it is probably under dimensioned and then the fail-over should be disabled.

The User Dispatcher will not fail over when it has detected a 503 response (which indicates overload policy activated). However, if a server is in the highest overload policy state where it drops messages instead of responding 503 the User Dispatcher monitor will receive an internal 408, which can never be distinguished from a dead server and failover will occur.

Synchronization of Failover Events

Depending on the failure detection mechanism there may be a need to synchronize the fail-over events (or the resulting state) between the different dispatcher instances. This is required if the detection mechanism is not guaranteed to be consistent across the cluster, such as an Error Response. For instance one server node sends a 503 response on one request but after that works just fine (this can be due to a glitch in the overload policy). If there was only one 503 sent then only one dispatcher instance will receive it and if that event triggers a fail-over then that dispatcher instance will be out of sync with the rest of the cluster. Further, even if the grace period is implemented so that it takes several 503 responses over a time period to trigger the fail-over there is still a risk for a race condition if the failure duration is the same as the grace period.

The following methods can be used to assure that the state after fail-over is synchronized across the cluster of dispatcher instances:

Broadcasting Fail-Over Events In this approach each dispatcher instance has to send a notification to all other instances (typically using JGroups or some other multicast technique) when it has decided to take a fail-over action and change the set of servers. This method can still lead to race conditions since two instances may fail-over and send a notification at the same time for two different server nodes.

Shared State If all dispatcher nodes in the cluster share the same state from a single source of truth then when the state is changed (due to a fail-over action) by any instance all other instances will see the change.

Expanding the Cluster

Since the Presence application can generate an exponentially increasing load due to the fact that every user subscribes to multiple (potentially a growing number of) other users, there is a need for a way to dynamically expand the cluster without too much disturbance. Compared to for instance a classic telecom application where it may be acceptable to bring all servers down for an upgrade of the cluster during low traffic hours, a Presence system may have higher availability requirements than that.

Expanding the cluster may involve both adding Presence nodes and User Dispatcher nodes.

When a new Presence server is added to a cluster, some presentities must be migrated from old nodes to the new node in order to keep a fairly even distribution. This migration needs to be minimized to avoid a too big flood of traffic on the system upon changing the cluster.

When a new User Dispatcher is added to the cluster that User Dispatcher node must achieve the same dispatching state as the other dispatcher nodes. This may depending on the pool implementation require a state being synchronized with the other dispatcher nodes (for instance when using the bucket pool implementation with persistence).

Updating the Node Set

Depending on the algorithm used to find the server node for a given presentity, different number of presentity will be migrated to another node when a new node is added or removed. An optimal Pool implementation will minimize this number.

Migrating Presentities

When the node set has been updated some Presentities may have to be migrated to maintain an even distribution. The different ways to do this are described in "[Presentity Migration](#)".

Failover Use Cases

These use cases illustrates how the User Dispatcher reacts in different failure situations in one or several Presence server nodes.

One Presence Server Overloaded for 60 Seconds

The cluster consists of four Presence servers, each node consisting of one OCMS instance with a User Dispatcher and a Presence application deployed. 100.000 users are distributed over the four servers evenly (25.000 on each node). Due to an abnormally long GC pause on one of the servers, the processing of messages is blocked by the Garbage Collector, which leads to the SIP queues getting filled up and the overload policy is activated. 60s later the processing resumes and the server continues to process messages.

The User Dispatcher will not do any fail-over but keep sending traffic to the failing node. In this case no sessions will be migrated to another node since all PUBLISH and initial SUBSCRIBE requests will be sent to the failing node. The initial SUBSCRIBES that arrives during the failure period will fail with a non-481 error (likely 503). It is up to the client to try and setup a new subscription when the failing one expires or report a failure. All PUBLISH requests and initial SUBSCRIBE request will generate a failure.

When the failing node resumes to normal operation all traffic will be processed again and no requests should fail. The time it takes until all presence states are correct again will be minimal since no sessions were failed-over.

If the monitoring feature is implemented in a way that detects the node as down in this case, then some users will be migrated to another node and when this node comes back they will be migrated back again. This will generate some increased load for a duration of time. If the overload policy was activated because of a too high traffic load this migration is bad, since it will most likely happen again and since the other servers will most likely also be close to overload. This could lead to a chain reaction resulting in the whole cluster going down and a complete loss of service.

One Presence Server Overloaded Multiple Times for Five Seconds

This use case describes a Presence server that is going in and out from overload with short time periods such as 5 seconds. This is common if the system is under dimensioned and can barely cope with the traffic load, but it could also be caused by some other disturbance only on that particular node. The User Dispatcher will behave exactly as in ["One Presence Server Overloaded for 60 Seconds"](#) and the result will be the same except that the number of failed sessions and failed-over sessions will be smaller due to the shorter failure period.

Overload Policy Triggered by an OCMS Software Failure

A failure in the OCMS software or an application deployed on top of it causes all threads to be locked (deadlock). This will eventually lead to that the in queue is filled up and the overload policy is activated even though the system is not actually overloaded. This is a permanent error that can only be solved by restarting the server.

Depending on if and how the monitor function is implemented the number of affected users can be minimized. However this cannot be distinguished from a real overload situation in which case a fail-over may not be the best thing to do.

A Presence Server Hardware Failure

The cluster consists of four Presence servers, each node consisting of one OCMS instance with a User Dispatcher and a Presence application deployed. 100.000 users are distributed over the four servers evenly (25.000 on each node). One of the presence servers crashes due to a hardware failure. A manual operation is required to replace broken server with a new one and only after two hours is the server up and running again. Depending on the type of the failure the response code sent back on transactions proxied to the failed node will be 408 or 503.

In this case all sessions on this node will fail since the failure duration is (most likely) more than the expiration time for the subscriptions. If a monitor server is implemented with fail-over then the failure time will be minimized to the detection time (seconds). The users will be migrated by the migration feature, which will create an increased load for a duration of time.

Because the User Dispatcher was also running on the failed node, all the persisted data for the user dispatcher will be lost when replacing the server with a new machine.

Expanding the Cluster with One Presence Node

The cluster consists of 3 Presence servers, each node consisting of one OCMS instance with a User Dispatcher and a Presence application deployed. 100.000 users are distributed over the four servers evenly (33.000 on each node). A new node is installed and added to the cluster. The following sequence of operations are performed to add the new node:

1. The User Dispatcher and the Presence application on the new node are configured with the same settings as the rest of the cluster. This includes synchronizing the distribution state to the new User Dispatcher in case of a pool implementation with persistence.
2. The `addServer JMX` operation is invoked with the new node on the cluster User Dispatcher MBean. This will invoke the `addServer` operation on all User Dispatcher nodes (including the new node).
3. The Load Balancer is reconfigured with the new node so that initial requests are sent to the new User Dispatcher node.

4. Depending on the migration approach an additional JMX operation may be invoked on the Presence application (using the cluster MBean server).

The result of this is that the new distribution of users is 25.000 on each node after 8.000 users have been migrated. Depending on the migration method this will generate an increased load of traffic on the system over a period of time.

Removing a Node from the Cluster

The cluster consists of four Presence servers, each node consisting of one OCMS instance with a User Dispatcher and a Presence application deployed. 100.000 users are distributed over the four servers evenly (25.000 on each node). One Presence node is removed from the cluster. The following sequence of operations are performed to remove the node:

1. The Load Balance is reconfigured to not include the node to be removed.
2. The removeNode JMX operation is invoked to remove the node from all the User Dispatcher's in the cluster. The cluster MBean is used to delegate the operation.
3. Depending on the migration approach an additional JMX operation may be invoked on the node to be removed.
4. When all users have been migrated from the node to be removed (the duration of this depends on the migration method) the node is finally stopped and removed from the cluster.

The result of this is that the new distribution of users is 33.000 on each node after 8.000 have been migrated.

OPMN Restart After a Presence Server Crash

Consider a four-node cluster with a User Dispatcher and a Presence application deployed on each node. The Presence server JVM on one of the nodes crashes and OPMN restarts the process. The restart takes one minute.

503 Responses from an Application

Due to a software bug or misbehavior in the application, 503 responses are sent for all incoming traffic. The SIP server itself is not under a significant load and the Overload Policy has not been activated. This may or may not be a permanent error condition.

OCMS Parlay X Web Services Architecture

This chapter describes the architecture, security, and installation for the OCMS Parlay X Web Services. This chapter contains the following sections:

- ["Architecture of Web Service Client Applications"](#)
- ["Web Service Security"](#)
- ["Installing the Web Services"](#)

Architecture of Web Service Client Applications

The architecture of client applications is such that one client of a Web service will be acting on behalf of many end users of the system (Figure 8-1).

Multiple users can simultaneously connect to the same Web service client, which will act on behalf of those users when invoking the Web Service. Note the following usage guidelines:

- Security – the OCMS Web server on which the Web services are running authenticates the client and not the end users. The client is a trusted entity and once the client is authenticated it is assumed that all end users of the client are authenticated. This places the task of authenticating end users on the Web service client. For the client to be correctly authenticated, they must connect with pre-determined authentication credentials.
- The Web client need not invoke all the Web services – that is, a web client might invoke methods on the SendMessage Web service only, and therefore has no need to concern itself with the other Web services. However, there are instances where the needs of the client application dictate that it invokes specific methods on the different web services in a specific order to achieve its goals. For example, a client applications for sending messages that also wants to receive message notifications for all the messages sent must first call the startMessageNotification method of the MessageNotificationManager interface before it can receive notifications for messages sent.

Web Service Security

The default deployments of all Web services on the OCMS server require that the clients authenticate themselves using DIGEST authentication (requires username and password). To support this, the parlayxclient-10.1.3.4.jar includes runtime xml descriptors (<interface-name>Binding_Stub.xml) that are configured to support inclusion of the required headers for DIGEST authentication from the clients.

Therefore, clients using the OCMS `parlayxclient-10.1.3.4.jar` need only set the username and password for the specific Web service client they are using. For instance:

```
SendMessageClient smc = new SendMessageClient();
...
smc.setUsername("oracle-ws-client");
smc.setPassword("secret");
```

The username and password used for authentication must be pre-determined and configured on the OCMS web server in order for authentication to succeed. Configuring the correct username and password on the server depends on the choice of security provider to be configured. For instance, consider configuring the username and password of the multimedia messaging web services (this includes `SendMessage`, `ReceiveMessage` and `MessageNotificationManager` services) to use a file based security provider (the simplest). Perform the following steps on the OCMS server:

1. Log onto Application Server Control Enterprise Manager.
2. Select **Administration > Security > Security Providers** and edit the `messagingwebservice` security provider.
3. Verify that the default security provider is a File-Based Security Provider. To modify it, select **Change Security Provider > File-Based Security Provider**, select "Use the OC4J instance default file based security provider" and click **OK**.
4. Select the **Realms** and click the **Users** link by the default realm (`jazn.com`).
5. Choose **Create** to add a new user.
6. Enter the user name (for example, *oracle-ws-client*), password (for example, *secret*) and add the *users* user role. Select **OK**. You can now connect to any of the multimedia messaging web services as user "oracle-ws-client" with password "secret", using the following code on the client side:

```
smc.setUsername("oracle-ws-client");
smc.setPassword("secret");
```

Typically, client application developers do not need to know the details of setting up authentication credentials on the server side, since that would be handled by the server administrator.

If the server administrator chooses to change the authentication requirements on the Web services (for instance, to use a PLAINTEXT username and password instead of DIGEST), they will provide instructions on new requirements. This typically requires unpacking the `parlayxclient-10.1.3.4.jar`, modifying `<interface-name>Binding_Stub.xml`, and re-packaging the updated `parlayxclient-10.1.3.4.jar` for use when running your application. The instructions provided by your server administrator should include the relevant details.

Web Service Security on Notification

The different Web services include corresponding notification Web services (`MessageNotification`, `PresenceNotification`) that run on the client side and receive notifications (message delivery status, message receipt, presence status change) when the appropriate event occurs. This implementation does not provide for the use of Web Service security (WS-Security) by default during notification of the clients. That is, the server assumes that the notification Web services running on the client side do not use WS-Security, and makes no attempt to authenticate itself when sending notifications. If you do enable WS-Security on the client side, the notification from the server will fail because the notification SOAP request will be missing the required headers.

Installing the Web Services

The Web services are packaged as a standard .ear file and can be deployed the same as any other Web services through Enterprise Manager. The .ear file contains two .war files that implement the two interfaces. If the Web services are deployed on the same server as the presence server, they must be a child application of the presence server.

Your client applications need to import (and be compiled against) the parlayx libraries that are provided with OCMS. This consists of importing the following jars into your projects:

- parlayx-10.1.3.4.jar – this jar contains all the 'unmodified' classes – classes that are generated by the oracle web services assembler and are not to be modified/customized. This includes the types (common types as well as the types for different web services), runtime classes, and local classes.
- parlayxclient-10.1.3.4.jar – this jar contains all the modified <interface-name>Client classes (for instance, PresenceConsumerClient, and SendMessageClient) that have been specialized to work with the corresponding Web services deployed with Oracle Communication and Mobility Server.

In addition to compiling against these jar files, they should be also included in the runtime configuration of the client (this might mean, for instance, setting the classpath in case of a console client, or including the jars in the deployed war/ear in a client application deployed into a J2EE container).

In addition to the jars above, the OCMS installation contains war files for the notification Web services. These war files contain all the necessary jar files that developers need to import to enable notification for the different Web services:

- messagingwsnotification-10.1.3.4.war – deployable war file that contains jars that should be imported when building a client intends to receive notifications for message delivery status and message reception. This war should also be deployed along with the client application in order for the OCMS server to be able to invoke the messaging notification Web service.
- presencewsnotification-10.1.3.4.war – deployable war file that contains jars that should be imported when building a client intends to receive notifications for presence status changes. This war should also be deployed along with the client application so that the OCMS can invoke the presence notification Web service.

OCMS Parlay X Presence Web Services

This chapter describes OCMS support for the Parlay X 2.1 Presence Web Services interfaces for developing applications. The Web service functions as a Presence Network Agent which can publish, subscribe, and listen to notifies on behalf of the users of the Web service. This chapter contains the following sections:

- ["Introduction"](#)
- ["Presence Web Services Interface Descriptions"](#)
- ["Using the Presence Web Services Interfaces"](#)
- ["OCMS Parlay X Presence Custom Error Codes"](#)

Introduction

OCMS provides support for Part 14 of the Parlay X Presence Web Service as defined in the *Open Service Access, Parlay X Presence Web Services, Part 14, Presence ETSI ES 202 391-14* specification. The OCMS Parlay X Web service maps the Parlay X Web service to a SIP/IMS network according to the Open Service Access, Mapping of Parlay X Presence Web Services to Parlay/OSA APIs, Part 14, Presence Mapping, Subpart 2, Mapping to SIP/IMS Networks, ETSI TR 102 397-14-2 specification.

Note: Due to the synchronous nature of the Web Service, to receive a callback from the Web service the client must implement the Web Service callback interface. For Presence, the required interface is the `PresenceNotification` interface described in *Open Service Access, Parlay X Presence Web Services, Part 14, Presence ETSI ES 202 391-14*.

The Presence Web Service communicates directly with IMS presence network elements using the SIP/SIMPLE protocol interface, and uses the JSR-32 UAC framework to communicate with the SIP network.

The HTTP server that hosts the Presence Web Service is a Presence Network Agent or a Parlay X to SIP gateway.

Presence Web Services Interface Descriptions

The Presence Web Services consist of the following interfaces:

- `PresenceConsumer`: The watchers use these methods to obtain presence data ([Table 9-1](#)).

- PresenceNotification: The presence consumer interface uses the client callback defined in this interface to send notifications. (Table 9–2).
- PresenceNotificationListener: This is a thin Java wrapper layer on top of the Parlay X PresenceNotification interface. An Oracle extension provides the user context if available. The client application should implement this interface and register it using the addPresenceNotificationListener() method from PresenceNotificationListenerManager. Once registered with the manager, it will be called if there is incoming notification. The end client can register multiple listeners. All listeners will be called on every incoming notification. The user context in the form of a SIP address is passed in the context parameter. This class is included in the presencewsnotification-10.1.3.4.war.
- PresenceSupplier: The presentity uses these methods to supply presence data and manage access to the data by its watchers (Table 9–3).

Table 9–1 PresenceConsumer Interface

Operation	Description
subscribePresence	The Web Services send a SUBSCRIBE to the presence server.
getUserPresence	Returns the cached presence status because the status changes of the presentity are asynchronously sent to the Web services through a SIP NOTIFY. The Web services actually have the subscription, not the Web services client.
startPresenceNotification	Indicates that the watcher want to receive notifications for a user presence status.
endPresenceNotification	Indicates that the watcher does not want further notifications for a specific notification request (identified by the correlator).

Table 9–2 PresenceNotification Interface (used by PresenceNotificationListener)

Operation	Description
statusChanged	The asynchronous operation is called by the Web Service when an attribute for which notifications were requested changes.
statusEnd	The notifications have ended. This message will not be delivered in the case of an error ending the notifications or deliberate ending of the notifications (using endPresenceNotification operation).
notifySubscription	This asynchronous operation is called by the Web Service to notify the watcher (application) that the subscription has terminated. Typical reasons are a timeout of the underlying SIP soft state subscription or the decision of the presentity to block further presence information to that watcher.
subscriptionEnded	This asynchronous method notifies the watcher that the server or the presentity handled the pending subscription.

Table 9–3 PresenceSupplier Interface

Operation	Description
publish	Maps directly to a SIP PUBLISH.

Table 9–3 (Cont.) PresenceSupplier Interface

Operation	Description
getOpenSubscriptions	Called by the presentity (supplier) to check if any watcher wants to subscribe to its presence data. No SIP message maps to this method. Returns pending subscriptions currently in the Web services server.
updateSubscriptionAuthorization	The supplier uses this method to answer any open pending subscriptions. An XCAP PUT message is sent to the XDMS server to update the presence-rule document.
getMyWatchers	Retrieves the local list of watchers from the Web services server.
getSubscribedAttributes	Retrieves the local list of subscribed attributes from the Web services server. Currently, only returns <i>Activity</i> .
blockSubscription	Causes the Web services server to end a watcher subscription by modifying the XCAP document on the XDMS server (i.e., putting the watcher on the block list).

Using the Presence Web Services Interfaces

This section describes how to use each of the operations in the interfaces, and includes code examples.

Interface: PresenceConsumer, Operation: subscribePresence

This is the first operation the application must call before using another operation in this interface. It serves two purposes:

- It allows the Web services to associate the current HTTP session with a user.
- It provides a context for all the other operations in this interface by subscribing to at least one presentity (SUBSCRIBE presence event).

Code Example

```
// Setting the attribute to activity
PresenceAttributeType pa = PresenceAttributeType.Activity;
PresenceAttributeType[] pat = new PresenceAttributeType[]{pa};

// These inputs are required but not used.
SimpleReference sr = new SimpleReference();
sr.setCorrelator("unique_correlator");
sr.setInterfaceName("PresenceNotification");
sr.setEndpoint(new URI
("http://127.0.0.1:8088/presencenotification/PresenceNotification"));

// Calling the web service
consumer.subscribePresence (new URI
("sip.presentity@test.example.com") , pat, "webcenter", sr);
```

Interface: PresenceConsumer, Operation: getUserPresence

Call this operation to retrieve a subscribed presentity presence. If the person is offline, it returns *ActivityNone* and the hardstate note will be written to *PresenceAttribute.note*. If it returns *ActivityOther*, the description of the activity is returned in the *OtherValue* field.

If the Name field is equal to "ServiceAndDeviceNote", OtherValue is a combination of the service note and the device note. Note that there can be more than one "ServiceAndDeviceNote" when the presentity is logged into multiple clients.

Code Example

```
PresenceAttributeType pat = new
    PresenceAttributeType(){PresenceAttributeType.Activity};
PresenceAttribute[] resultPA =
    consumer.getUserPresence(new URI(presentity),pat);
for (int i = 0; i < resultPA.length; i++){
    PresenceAttribute pa = resultPA[i];
    // Check to see if it is an activity type.
    if (pa.getTypeAndValue().getUnionElement() ==
        PresenceAttributeType.Activity){
        // Get the presence status.
        System.out.println("Activity: " +
            pa.getTypeAndValue().getActivity().toString());
        // Get the customized presence note.
        if (pa.getNote().length() > 0){
            System.out.println("Note: " + pa.getNote());
        }
    }
    // If this is of type Other, then we need to extract
    // different type of information.
    if (pa.getTypeAndValue().getUnionElement() ==
        PresenceAttributeType.Other){
        // This is "ActivityOther", a custom presence status.
        if (pa.getTypeAndValue().getOther()
            .getName().compareToIgnoreCase("ActivityOther") == 0){
            System.out.println("Other Activity->" +
                pa.getTypeAndValue().getOther().getValue() + "\n");
        } else {
            // Currently, the only other value beside ActivityOther is
            // "ServiceAndDeviceNote" which is the service note +
            // device note.
            System.out.println("Combined Note->" +
                pa.getTypeAndValue().getOther().getValue() + "\n");
        }
    }
}
}
```

Interface: PresenceConsumer, Operation: startPresenceNotification

This operation indicates that the watcher want to receive notifications for a user presence status.

Code Example

```
SimpleReference sr = getNotificationReference(presentity);

TimeMetric freq = new TimeMetric();
freq.setMetric(TimeMetrics.Minute);
TimeMetric duration = new TimeMetric();
duration.setMetric(TimeMetrics.Minute);

PresenceAttributeType pa = PresenceAttributeType.Activity;
PresenceAttributeType[] pat = new PresenceAttributeType[] { pa };
mConsumer.startPresenceNotification(new URI(presentity), pat, sr, freq, duration,
0, false);
```

Interface: PresenceConsumer, Operation: endPresenceNotification

This operation indicates that the watcher does not want further notifications for a specific notification request (identified by the correlator).

Code Example

```
// Pass in the correlator used in startPresenceNotification.
mConsumer.endPresenceNotification(correlator);
```

Interface PresenceSupplier, Operation: publish and Oracle Specific Remove Presence

This is the first operation the application must call before using another operation in this interface. It serves three purposes:

- It allows the Web services to associate the current HTTP session with a user.
- It publishes the user's presence status.
- It subscribes to watcher-info so that the Web services can keep track of any watcher requests.

There are three attributes that are of interest when performing a PUBLISH. These attributes can be set in a PresenceAttribute structure and passed into the PUBLISH method.

- Presence status with a customized note: this is the customized note configured in the My Presence text box in Oracle Communicator. The <note> element is contained in the <person> element of the Presence Information Data Format (PIDF) XML file.
- Device note: implicitly inserted by Oracle Communicator, or inserted from a Web service. The <note> element is contained in the <device> element of the Presence Information Data Format (PIDF) XML file.
- Service note: configured in the Presence tab in the Oracle Communicator preferences. The <note> element is contained in the <tuple> element of the Presence Information Data Format (PIDF) XML file.

Code Example

```
// PresenceAttribute contains presence status and note.
typeValue.setUnionElement(PresenceAttributeType.Activity);
typeValue.setActivity(activity);
paActivity.setTypeAndValue(typeValue);
// Setting the customized note here.
paActivity.setNote(activityNote);
paActivity.setLastChange(dateTime);

// Create the PresenceAttribute containing device note.
AttributeTypeAndValue typeValueOther = createATV();
PresenceAttribute paOther = new PresenceAttribute();
// Device note is carried in a PresenceAttributeType.Other
typeValueOther.setUnionElement(PresenceAttributeType.Other);
// Set the name to "DeviceNote" to indicate the value
// should be used as device note.
other.setName("DeviceNote");
other.setValue(deviceName);
typeValueOther.setOther(other);
```

```
paOther.setTypeAndValue(typeValueOther);

// Create the PresenceAttribute containing service note.
AttributeTypeAndValue typeValueOther1 = createATV();
PresenceAttribute paOther1 = new PresenceAttribute();
// Service note is carried in another
// PresenceAttributeType.Other
typeValueOther1.setUnionElement(PresenceAttributeType.Other);
OtherValue other1 = new OtherValue();
// Set the name to "ServiceNote" to indicate the value
// should be used as device note.
other1.setName("ServiceNote");
other1.setValue(serviceName);
typeValueOther1.setOther(other1);
paOther1.setTypeAndValue(typeValueOther1);
// The note is not used. Can be anything.
paOther1.setNote("OracleExtension");
paOther1.setLastChange(dateTime);

//Unpublish Functionality Implemented by OCMS
//To perform an "Unpublish", set OtherValue to (Expires, 0)
//OtherValue other = new OtherValue();
//other.setName("Expires");
//other.setValue(0);
//typeValue.setOther (other);
//typeValue.setUnionElement(PresenceAttributeTypeOther);

paArray = new PresenceAttribute[]{paActivity,paOther,paOther1};

// Calling the publish method by passing the PresenceAttribute
// array containing the presence status, device note and service
// note.
publish(paArray);
```

Interface: PresenceSupplier, Operation: getOpenSubscriptions

This operation retrieves a list of new requests to be on your watcher list.

Code Example

```
SubscriptionRequest[] srArray = getOpenSubscriptions();
for (SubscriptionRequest sr:srArray) {
    System.out.println(sr.getWatcher().toString());
}
```

Interface: PresenceSupplier, Operation: updateSubscriptionAuthorization

This operation allows you to place a watcher on either the block or allow list.

Code Example

```
//You always pass in Activity
pp.set.PresenceAttribute(PresenceAttributeType.Activity);
updateSubscriptionAuthorization(new URI("sip:allow@test.example.com"),
new PresencePermission[]{pp});
PresencePermission pp = new PresencePermission();
pp.setDecision(true);
//Put the user on the allow list
```


Interface: PresenceSupplier, Operation: getMyWatchers

This operation retrieves the list of watchers in your allow list.

Code Example

```
URI[] uris;
uris = getMyWatchers();
for (URI uri:uris)
    System.out.println(uri.toString());
```

Interface: PresenceSupplier, Operation: getSubscribedAttributes

This operation returns only a single item of PresenceTypeAttribute.Activity. An exception will be thrown if there is no existing subscription.

Code Example

```
PresenceAttributeType[] pat =
getSubscriberdAttributes("sip:watcher@test.example.com");
```

Interface: PresenceSupplier, Operation: blockSubscription

This operation places a watcher into the block list.

Code Example

```
blockSubscription(new URI("sip:block.this.watcher@test.example.com"));
```

OCMS Parlay X Presence Custom Error Codes

OCMS introduces two extensions to the Parlay X standard exceptions:

- *PresencePolicyException* extends *PolicyException*, and
- *PresenceServiceException* extends *ServiceException*

[Table 9–4](#) and [Table 9–5](#) describe the error codes and their associated error message.

Table 9–4 OCMS Parlay X Presence Custom Error Codes: PresencePolicyException

Error Code	Error Message
SDP20201	Watcher is on the block, polite-block or pending list.
SDP20202	Subscription is pending.

Table 9–5 OCMS Parlay X Presence Custom Error Codes: PresenceServiceException

Error Code	Error Message
SDP20101	Invalid result from XDMS server.
SDP20102	Invalid HTTP session data.
SDP20103	Invalid URI.
SDP20104	Peer unavailable.
SDP20105	Unknownhost.
SDP20106	Service not available.
SDP20107	Internal error.

Table 9–5 (Cont.) OCMS Parlay X Presence Custom Error Codes:

Error Code	Error Message
SDP20108	User unauthenticated.

OCMS Parlay X Multimedia Messaging Web Services

This chapter describes OCMS support for the Parlay X Multimedia Messaging Web Services interfaces for developing applications. This chapter contains the following sections:

- "Introduction"
- "Multimedia Messaging Web Services Interface Descriptions"
- "Using the Multimedia Messaging Web Services Interfaces"

Introduction

OCMS implements support for a subset of the operations in the `SendMessage`, `ReceiveMessage`, and `MessageNotificationManager` interfaces, as they are defined in *ETSI ES 202 391-5 V1.2.1 (2006-12), Open Service Access (OSA), Parlay X Web Services Part 5: Multimedia Messaging (Parlay X 2)*.

Multimedia Messaging Web Services Interface Descriptions

The Multimedia Messaging Web Services consist of the following interfaces:

- `SendMessage`: Provides operations to send messages and check status of sent messages ([Table 10-1](#)).
- `ReceiveMessage`: Provides operations to retrieve messages that have been received ([Table 10-2](#)).
- `MessageNotificationManager`: Provides an application side notification interface where notifications about multimedia messages are delivered. ([Table 10-3](#)).
- `MessageNotification`: Provides notifications about multimedia messages ([Table 10-4](#)).
- `MessagingNotificationListener`: This is a thin Java wrapper layer on top of the Parlay X `MessagingNotification` interface. An Oracle extension provides the user context if available. The client application should implement this interface and register it using the `addMessagingNotificationListener()` method from `MessagingNotificationListenerManager`. Once registered with the manager, it will be called if there is incoming notification. The end client can register multiple listeners. All listeners will be called on every incoming notification. The user context in the form of a SIP address is passed in the context parameter. This class is included in the `messagingwsnotification-10.1.3.4.war`.

Table 10–1 SendMessage Interface

Operation	Description
sendMessage	Request to send a Message to a set of destination addresses, returning a requestIdentifier to identify the message. The requestIdentifier can subsequently be used by the application to poll for the message status.
getMessageDeliveryStatus	This method is not supported in this implementation, and will always throw a ServiceException with code SVC0001. To get the delivery status on a sent message, call the sendMessage method with a valid SimpleReference pointing to the notification endpoint which will be invoked for each of the target URIs.

Table 10–2 ReceiveMessage Interface

Operation	Description
getReceivedMessages	This method is not supported in this implementation. Clients are not allowed to poll for received messages; instead, they must use the startMessageNotification method of the MessageNotificationManager interface to register the notification endpoint that will be invoked whenever a new message is available for the endpoint user.
getMessageURIs	This method is not supported in this implementation – calling this method will always result in a ServiceException with code SVC001.
getMessage	This method will read the whole message. The data is returned as an attachment in the return message.

Table 10–3 MessageNotificationManager Interface

Operation	Description
startMessageNotification	Start notifications to the application for a given Message Service activation number and criteria. Implemented according to the Parlay X 2.1 specification, section 8.4.1, with the clarifications described in: " Interface: MessageNotificationManager, Operation: startMessageNotification ".
stopMessageNotification	This operations allows an application to end a multimedia message notification. Implemented according to the Parlay X 2.1 specification, section 8.4.2, with the clarifications described in: " Interface: MessageNotificationManager, Operation: stopMessageNotification ".

Table 10–4 MessageNotification Interface

Operation	Description
notifyMessageReception	This notification is sent when a message is received by the server from the SIP network. The MessageReference's Subject will contain the text message if the content-type is plain-text.
notifyMessageDeliveryReceipt	This notification is sent if either the message has been delivered or if delivery is impossible.

Using the Multimedia Messaging Web Services Interfaces

This section provides guidelines for using each of the operations in the interfaces.

Interface: `SendMessage`, Operation: `sendMessage`

This method is always invoked on behalf of an actual end user. The end user is identified by a SIP Address-Of-Record (AOR). We determine the AOR by parsing the *from* part of the message request:

The `requestIdentifier` returned should not be used to poll for message status using the `getMessageDeliveryStatus` method because this implementation does not implement the `getMessageDeliveryStatus` method. Instead, pass in a valid `SimpleReference` object to the `sendMessage` method in order to get a notification of the delivery status once it is available for each of the target URIs.

The `senderAddress`, `priority` and `charging` parameters are presently ignored.

Group URLs are not supported.

If the caller of this method includes an attachment in the SOAP context, then that attachment is presumed to contain the body of the message, in which case the `subject` parameter is ignored; the raw contents of the attachment (bytes) are sent to the target URIs. If, however, the caller does not include a SOAP attachment, then it is presumed that the `subject` parameter consists of the whole message – therefore a SIP message with content type `text/plain` is sent to the target recipients.

Interface: `sendMessage`, Operation: `getMessageDeliveryStatus`

This method is not supported in this implementation, and will always throw a `ServiceException` with code `SVC0001`. To get the delivery status on a sent message, call the `sendMessage` method with a valid `SimpleReference` pointing to the notification endpoint which will be invoked for each of the target URIs.

Interface: `ReceiveMessage`, Operation: `getReceivedMessages`

This method is not supported in this implementation, and will always throw a `ServiceException` with code `SVC001`. Clients are not allowed to poll for received messages; instead, they must use the `startMessageNotification` method of the `MessageNotificationManager` interface to register the notification endpoint that will be invoked whenever a new message is available for the endpoint user.

Interface: `ReceiveMessage`, Operation: `getMessageURIs`

This method is not supported in this implementation – calling this method will always result in a `ServiceException` with code `SVC001`.

Interface `ReceiveMessage`, Operation: `getMessage`

Implemented according to the Parlay X 2.1 specification, section 8.2.3, with the following clarification:

Whenever this method is called with a `messageRefIdentifier` that does not exist on the server, a `ServiceException` with error code `SVC002` is thrown.

Interface: MessageNotificationManager, Operation: startMessageNotification

Implemented according to the Parlay X 2.1 specification, section 8.4.1, with the following clarifications:

- The `messageServiceActivationNumber` translates to the SIP Address-Of-Record of the user for whom message notifications will be delivered.
- The `criteria` parameter is currently ignored.
- The `correlator` parameter in the `SimpleReference` object used in this method must be globally unique for every instance of an end user on any client. See discussion under *stopMessageNotification* below.

Interface: MessageNotificationManager, Operation: stopMessageNotification

Implemented according to the Parlay X 2.1 specification, section 8.4.2, with the following clarification:

- Since the `messageServiceActivationNumber` provided in the call to *startMessageNotification* is the SIP AOR of the end user, then the `correlator` passed to *stopMessageNotification* must be able to be uniquely mapped to a particular end-user instance from a particular client. Therefore, we require that the notification correlator be globally unique for each end user instance on any client, so that we can correctly map the stop notification request to the correct user instance.

Provisioning Users with Sash

This chapter describes using the Sash utility. This chapter includes the following sections:

- ["Overview of Sash"](#)
- ["Launching Sash"](#)
- ["Using Sash"](#)
- ["Creating a User"](#)
- ["Provisioning the XDMS Using Sash"](#)
- ["Scripting with Sash"](#)
- ["Error Logging in Sash"](#)

Overview of Sash

The Sapphire Shell (Sash) is a command-line utility to provision OCMS users to the Oracle database, the XDMS (XML Document Management Server) and the RADIUS server. You can provision users from the Sash command line prompt (`sash#`) or by using the [CommandService](#) MBean.

See ["Configuring Oracle Internet Directory as the User Repository"](#) for information on using Oracle Internet Database (OID) as the user provisioning repository for an OCMS deployment.

Launching Sash

On Linux systems, the Sash launcher script (`launch_sash.sh`) is located in the same folder that contains the start and stop scripts for OCMS.

Launching Sash from the Command Line

OCMS provides the following shortcuts for launching Sash from the command line:

`launch_sash.sh` (UNIX)

`launch_sash.bat` (Windows)

This shortcut is located at `ORACLE_HOME/sdp/bin` on OC4J OCMS installations.

Connecting Sash to an External OCMS Instance

By default, Sash connects to the local instance of OCMS. If needed, you can override this default behavior and connect Sash to external instances of OC4J or to another instance of Oracle Application Server.

Connecting to an External Instance of OC4J

Sash connects to the OCMS server through RMI. [Example 11-1](#) illustrates how to connect Sash to a server with the host IP address 10.0.0.234.

Example 11-1 Connecting Sash to OCMS

```
sash --host 10.0.0.234
```

When you connect to OC4J, Sash prompts you for a username and a password. The user name is the same as that for OC4J administrator (oc4jadmin). The password is the same as the password associated with the OC4J administrator. Once you log in, the Sash command prompt (*sash*) appears. An error message displays if the login is unsuccessful.

Connecting Sash to an External Oracle Application Server Instance

To connect Sash to an external instance of the Oracle Application Server, use the `--ias` option and then add `--port` followed by the port number for OPMN (Oracle Process and Management and Notification) server. [Example 11-2](#) illustrates how to connect to an external instance of the Oracle Application Server.

Example 11-2 Connecting to an External Instance of the Oracle Application Server

```
sash --ias --port 6003
```

The default port is 6003 if `--port` is not specified.

When you connect to the server, enter the administrator user name and password.

Tip: The OPMN request port is configured in the `opmn.xml` configuration file under `ORACLE_HOME/opmn/conf/opmn.xml`.

Using Sash

There are two groups of Sash commands: commands that create, delete and update system objects and commands that query the system for information.

Viewing Available Commands

Entering `help` displays a list of all available commands in the server (described in [Table 11-1](#)). The list of commands varies depending on the components deployed to the server.

Table 11–1 Sash Commands

Command	Description	Aliases	Subcommands
<code>privateIdentity</code>	Commands for adding and removing private communication identities used for authentication.	None	<p>Subcommands include:</p> <ul style="list-style-type: none"> ■ <code>add</code> – Adds a new user to the system. For example: <code>privateIdentity add privateId=alice</code> ■ <code>delete</code> – Removes a user from the system. For example: <code>privateIdentity delete privateId=alice</code>
<code>publicIdentity</code>	Commands for adding and removing public identities associated with a private identity.	<code>pubid</code>	<p>Subcommands include:</p> <ul style="list-style-type: none"> ■ <code>add</code> – Adds a public identity to the system which is associated with a particular user. For example: <code>publicIdentity add publicId=sip:alice@test.company.com privateId=alice</code> ■ <code>delete</code> – Deletes a communication identity from the system. For example: <code>publicIdentity delete publicId=sip:alice@test.company.com privateId=alice</code>
<code>account</code>	Contains commands for managing user accounts. This command enables you to set the account as active, locked, or as a temporary account.	None	<p>Subcommands include:</p> <ul style="list-style-type: none"> ■ <code>add</code> – adds a new account to the system. The syntax is as follows: <code>account add uid=<string> [active=<true false> [locked=<true false> [accountExpiresAt=<accountExpiresAt> [tempAccount=<true false> [description=<string> [lockExpiresAt=<lockExpiresAt> [currentFailedLogins=<integer></code> For example: <code>account add uid=alice active=true</code> ■ <code>delete</code> – Deletes an account from the system. For example: <code>account delete uid=<string></code> ■ <code>update</code> – Updates an account. For example: <code>account update uid=<string> [active=<true false> [locked=<true false> [accountExpiresAt=<accountExpiresAt> [tempAccount=<true false> [description=<string> [lockExpiresAt=<lockExpiresAt> [currentFailedLogins=<integer></code> ■ <code>info</code> – Retrieves information for a specific account. For example: <code>account info uid=<string></code>

Table 11-1 (Cont.) Sash Commands

Command	Description	Aliases	Subcommands
role	Manages role types and user roles in the system. <code>role</code> is an additional security and authorization mechanism that is defined within the <code><auth-constraint></code> element of <code>sip.xml</code> . This command authorizes a group of users access to applications. The applications in turn check for a specific role. OCMS defines one role for the Proxy Registrar application, "Location Services".	None	Subcommands include <code>role system</code> and <code>role user</code> .
role system (subcommand of role)	Manages the roles types.	None	Subcommands include: <ul style="list-style-type: none"> ■ <code>list</code> – Lists the roles in the system. For example: <code>role system list</code> ■ <code>add</code> – Adds a new role to the system. For example: <code>role system add name=<string></code> <code>[description=<string>]</code> ■ <code>update</code> – Updates a role in the system. For example: <code>role system update name=<string></code> <code>[description=<string>]</code> ■ <code>delete</code> – Deletes a role from the system. For example: <code>role system delete name=<string></code> <code>[description=<string>]</code>

Table 11–1 (Cont.) Sash Commands

Command	Description	Aliases	Subcommands
role user (subcommand of role)	Manages the user roles	None	Subcommands include: <ul style="list-style-type: none"> ■ add – Adds a role to a user. For example: role user add uid=<string> name=<string> ■ delete – Deletes a role from a user. For example: role user delete uid=<string> name=<string> ■ list – Lists roles for a user. For example: role user list uid=<string>
credentials	Command for managing credentials.	None	Subcommands include: <ul style="list-style-type: none"> ■ add – Adds credentials to a user. For example: credentials add password=<string> realm=<string> uid=<string> ■ addAll – Adds credentials for all of the configured realms in the system to a user. For example: credentials addAll password=<string> uid=<string> ■ delete – Deletes realm credentials for a user. For example: credentials delete realm=<string> uid=<string> ■ deleteAll – Deletes all credentials for a user. For example: credentials deleteall uid=<string> ■ update – Updates the credentials for a user. For example: credentials update password=<string> realm=<string> uid=<string> ■ updateAll – Updates a user’s credentials for all provisioned realms in the system. For example: credentials updateAll password=<string> uid=<string> ■ list – Lists all of the realms for which credentials exist for a given user. For example: credentials list uid=<string>
identity add	Enables you to create a basic user account.	None	None. See " Creating a User with the identity add Command ".

Viewing Subcommands

To view the subcommands for a specific command, enter `help <command>`. For example, entering `help` for the `account` command (`help account`) retrieves a brief overview of the subcommands available to the `account` command (illustrated in [Example 11–3](#)).

Example 11–3 Retrieving Help for a Specific Command

```
*** Description ***
```

```
Contains commands for management of user accounts.
```

```
In an account you can set if the account is active,
```

locked or if it perhaps should be a temporarily account.

Aliases: [no aliases]

Syntax:
account

Sub-commands:

```
# Adds a new account to the system
  account add uid=<string> [ active=<true|false> ] [ locked=<true|false> ] [
accountExpiresAt=<accountExpiresAt> ] [ tempAccount=<true|false> ] [
description=<string> ] [ lockExpiresAt=<lockExpiresAt> ] [
currentFailedLogins=<integer> ]

# Deletes an account
  account delete uid=<string>

# Updates an account
  account update uid=<string> [ active=<true|false> ] [ locked=<true|false> ] [
accountExpiresAt=<accountExpiresAt> ] [ tempAccount=<true|false> ] [
description=<string> ] [ lockExpiresAt=<lockExpiresAt> ] [
currentFailedLogins=<integer> ]

# Retrieve information about a particular account
  account info uid=<string>
```

In addition to the overview of the command group, the information displayed by entering `help <command>` also includes the aliases (if any) to the command. For example, the overview of the `account` command illustrated in [Example 11-3](#) notes [no aliases] for the command.

Note: The `delete` command used with `account`, `role`, `role system`, `role user`, `privateIdentity`, `publicIdentity`, and `identity` has the following aliases:

- `remove`
 - `del`
 - `rm`
-
-

Some commands require parameters. For example, if you enter `help role system add`, the system informs you that the `add` command requires the name of the role and an optional command for setting the description as well by displaying `role system add name=<string> [description=<string>]`.

Note: Optional commands such as `[description=<string>]` are enclosed within square brackets `[...]`.

The system alerts you if you omit a mandatory parameter or if you pass in a parameter that is not recognized.

Creating a User

This section describes the `publicIdentity` and `privateIdentity` commands and how to use them in conjunction with the `add`, `account`, `role`, and `credentials` subcommands listed in [Table 11-1](#) to provision a user account to the Oracle database.

The Private Identity (`privateIdentity`) uniquely identifies a user within a given authentication realm. The Public Identity (`publicIdentity`) is the SIP address that users enter to register devices. This address is the user's AOR (Address of Record), and the means through which users call one another. A user can have only one Private Identity, but can have several Public Identities associated with that Private Identity.

Note: To enable authentication to third-party databases (such as RADIUS), user accounts that contain authentication data and are stored externally must match the Private Identity to ensure the proper functioning of the Proxy Registrar and other applications that require authentication.

To create a user, first add the user to the system by creating a private identity and then a public identity for the user using the `privateIdentity` and `publicIdentity` commands with the `add privateId` and `add publicId` subcommands, respectively.

Once you create the private and public identity for the user, create an account for the user with the `account add uid` command and optionally set the status of the account (such as active or locked). The `role` command sets the role memberships for role-based permissions. Set the level of permissions for the users using the `role` command, and then set user credentials by defining the user's realm and password with the `credentials` command.

Creating a User from the Sash Command-Line Prompt

This section illustrates how to create a user from the Sash command prompt (`sash#`, illustrated in [Example 11-4](#)) by creating an OCMS user known as *alice* using the commands described in [Table 11-1](#).

1. Create a user using the `privateIdentity` command as follows:

```
privateIdentity add privateId=alice
```

2. Create the public identity for alice by entering the SIP address:

```
publicIdentity add publicId=sip:alice@test.company.com privateId=alice
```

3. Add an account for alice and use one of the optional commands described in [Table 11-1](#) to set the status of the account. To create an active account for alice, enter the following:

```
account add uid=alice active=true
```

Note: OCMS Version 10.1.3.2 requires that the `uid` be in lower-case. Oracle Communicator users provisioned using OCMS Version 10.1.3.2 must also enter their account names in lower case during login. OCMS Version 10.1.3.3 and 10.1.3.4 support mixed-case `uids`. However, Oracle Communicator users can only log in by entering their user name exactly as it was provisioned. For example, if you define the `uid` as Alice, then the user must login as Alice. If you upgrade to 10.1.3.4 from 10.1.3.2, users provisioned in 10.1.3.2 must continue to log in using lower case.

4. Use the `role` command to add alice to the *Location Service* user group. Doing so grants alice permission to the Proxy Registrar's Location Service lookup:

```
role user add uid=alice name="Location Service"
```

5. Add user authentication credentials for alice:

```
credentials add uid=alice realm=test.company.com password=welcome1
```

The `credentials` command is not needed for applications configured to use the RADIUS Login Module to authenticate users against RADIUS servers. For more information on these login modules, see [Chapter 4, "Configuring Security and Login Modules"](#).

Note: You must also configure `realms` using the [SIP Servlet Container MBean](#) before you use Sash to add authorization credentials to a user. For more information, see ["Configuring the SIP Servlet Container MBeans"](#).

Example 11–4 Creating a User from the Sash Command-Line Prompt

```
sash# privateIdentity add privateId=alice
sash# publicIdentity add publicId=sip:alice@test.company.com privateId=alice
sash# account add uid=alice active=true
sash# role user add uid=alice name="Location Service"
sash# credentials add uid=alice realm=test.company.com password=welcome1
```

Tip: You can create multiple users by creating Sash batch files. For more information, see ["Scripting with Sash"](#).

Creating a User with the Command Service MBean

You can execute Sash commands using the [CommandService MBean's](#) *execute* operation. The Command Service MBean is defined within the Subscriber Data Services application. For information on accessing application-defined MBeans, see ["Accessing the MBeans for a Selected SIP Application"](#).

To create a user:

1. Select the *execute* operation. The *Operation* page for the *execute* operation appears.
2. Enter `privateIdentity add privateId=alice` in the *Value* field ([Figure 11–1](#)).
3. Click **Invoke Operation**. Repeat this process for each of the user creation commands. For example, the subsequent `publicIdentity` and `account` commands would both be followed by **Invoke Operation**.

Figure 11–1 Creating a User with the Command Service MBean

The screenshot shows the Oracle Enterprise Manager 10g Application Server Control interface. The breadcrumb navigation is: Cluster Topology > Application Server: as10132_omcs.stanf10.us.oracle.com > OC4J: ocms > Application Server Control. An information banner at the top states "Operation executed successfully." Below this, the operation is identified as "execute" with "Return" and "Invoke Operation" buttons. The MBean name is "MBeansubscriberdataservices:service=CommandService,name=CommandService,SIPApplic". The description is "Executes a Command Service command." The return type is "java.lang.String". A "Parameters" table is shown with one row: "command" with description "The command to execute.", type "java.lang.String", and value "privateIdentity add privateId=alic". Below the table is a "Return Value" section showing "User alice was successfully added." and another "Return" and "Invoke Operation" button.

See "[CommandService](#)" for more information on executing Sash commands through Application Server Control.

Creating a User with the identity add Command

The `identity add` command enables you to create a user with one command string. This command, which is an alias to the `privateIdentity`, `publicIdentity`, `account`, `role` and `credentials` commands, enables you to quickly create a basic user account that contains the minimum information needed for users to connect to OCMS through a SIP client. For example, to create a basic account for user *alice* using this command, enter the following from either the command line or through the Command Service Mbean's *execute* operation:

```
identity add privateId=alice publicId=sip:sip.alice@company.com role="Location
Service" realm=company.com password=welcome1
```

Note: For applications configured to authenticate users against a RADIUS system (the applications with the RADIUS Login Module as the security provider), the command to create a user account is as follows:

```
identity add privateId=alice publicId=sip:sip.alice@company.com
role="Location Service"
```

The `identity add` command only enables you to create a basic user account. Accounts that require more complex construction, such as those that associate multiple `publicIds` with a single `privateId`, must be created using multiple Sash commands as illustrated in [Example 11–4](#).

Deleting a User Account with the identity delete Command

The `identity delete` command enables you to delete all of a user's roles, credentials, account information, public and private identities using a single command string. For example, to delete an account for user *alice* using this command, enter the following from either the command line or through the Command Service MBean's *execute* operation:

```
identity delete privateId=alice
```

Provisioning the XDMS Using Sash

The commands for provisioning the XDMS are included in the `xcap` group. Each of these commands is preceded by `xcap`. The XDMS commands within the `xcap` group that support user provisioning are included in the `user` and `applicationUsage` subgroups. You can provision XDMS from the Sash prompt or by using the `CommandService` MBean that is provided with the Presence application.

Provisioning XDMS User Accounts Using the CommandService MBean

You can provision XDMS using the *execute* command provided by the `CommandService` MBean that is registered to the Presence application (Figure 11–2). Use the `CommandService` MBean's *execute* operation as described in "Creating a User with the Command Service MBean" to provision accounts to the XDMS. For more information on the MBean itself, see "Command Service (XDMS Provisioning)".

Figure 11–2 Using the CommandService MBean for XCAP Account Management

ORACLE Enterprise Manager 10g
Application Server Control

OC4J: home > Application: presence > Application MBeans >

Operation: **execute**

Return Invoke Operation

MBean Name: `presence:service=CommandService,name=CommandService,SIPApplication=presenceapplication`

Description: Executes a Command Service command.

Return Type: `java.lang.String`

Parameters

Name	Description	Type	Value
command	The command to execute.	java.lang.String	xcap user appusages userName=ron

Return Value

[resource-lists, pdf-manipulation, org.openmobilealliance.pres-rules, pres-rules]

Return Invoke Operation

Provisioning XDMS User Accounts from the Sash Prompt

To use XDMS commands to provision users and application usages from the Sash prompt, you must first connect to an application that consumes XDMS, such as Presence.

For Windows systems, enter the following from the command prompt:

```
launch_sash.bat -a <application name>
```

On Linux, enter the following:


```
launch_sash.sh -a <application name>
```

You connect to the application through a command prompt, such as the Windows command shell (`Cmd.exe`). You cannot connect to these applications directly from the Sash prompt.

For example, to connect to the Presence application on a Windows system:

1. From the command prompt, navigate to the `sbin` directory that contains the Sash executable. This file is located at `ORACLE_HOME\sdp\sash\sbin`.
2. Enter the name of the Presence application using `sash.bat -a presenceapplication`. For example, enter the following:

```
c:\product\10.1.3.4\ocms\sdp\sash\sbin>sash.bat -a presenceapplication
```

3. When prompted, login to Sash using your OC4J administrator name and password.
4. From the Sash command prompt, enter an XDMS command, such as `xcap user list`.

Using xcap Commands

This section describes how to manage user accounts and application usages using the `xcap` group of commands.

Provisioning XDMS User Accounts

The `add`, `delete` and `list` commands enable you to manage user accounts on the XDMS.

Adding XDMS Users

The `xcap user add` command adds an XDMS user with the given user name and application usage. For example, to add a user from the Sash prompt, enter:

```
sash# xcap user add userName=<string> applicationUsage=<string>
```

Caution: Do not use the `add` command if the XDMS is configured to automatically create users.

Removing an XDMS User

The `xcap user delete` command removes an XDMS user with the given user name from the application usage. For example, to delete a user from the Sash prompt, enter:

```
sash# xcap user delete userName=<string> [ appusages=<string> ]
```

The application usage parameter (`appusages`) is optional. If no application usage is specified, then the user is removed from all application usages. When the server is configured to automatically create a user, the `delete` command removes all existing documents.

Searching for Application Usage for an XDMS User

The `xcap user appusages` command returns all the application usages applicable to a given user. To review the application usages assigned to a user, enter the following from the Sash prompt:

```
sash# xcap user appusages userName=<string>
```

Listing XDMS Users

The `xcap user list` command returns all of the XDMS users in the system, or optionally returns the XDMS users for a given application usage.

```
sash# xcap user list [ all=<true|false> ] [ appusage=<string> ]
```

If the optional `all` parameter is not set, then the resulting display is limited to a maximum of 100 users.

Provisioning Application Usage

The commands for provisioning of XDMS application usage are in the `appusage` group (`xcap appusage`).

Listing All Application Usages

The `xcap appusage list` command returns all the application usages on the server. For example, enter the following from the Sash prompt:

```
sash# xcap appusage list
```

Scripting with Sash

You can construct scripts for common tasks that contain several operations. Sash can be evoked to execute a file containing a list of commands. To enable scripting, Sash provides such command-line flags as:

- `-- exec` (short name: `-e`): When this command-line flag is followed by a command enclosed within quotation marks, Sash executes the command and then exits.
- `-- file` (short name: `-f`): When this command-line flag is followed by a filename, Sash reads the file and executes all commands in the file as they were entered and then exits.

Example 11-5 illustrates a text file called `ocsm_users.txt`, which contains a group of users defined with the `identity add` command. You can provision these users by entering `-f ocms_users.txt` from the Sash prompt:

Example 11-5 Creating Users from a Text File (`ocms_users.txt`)

```
identity add publicId=sip:alice@doc.oracle.com privateId=alice role=user
password=1234 realm=doc.oracle.com
identity add publicId=sip:bob@doc.oracle.com privateId=bob role=user password=1234
realm=doc.oracle.com
identity add privateId=candace publicId=sip:candace@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=deirdre publicId=sip:deirdre@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=evelyn publicId=sip:evelyn@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=frank publicId=sip:frank@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=gretchen publicId=sip:gretchen@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=hans publicId=sip:hans@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=imogen publicId=sip:imogen@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=jack publicId=sip:jack@doc.oracle.com role=user
```

```
password=1234 realm=doc.oracle.com
```

- `-- nonewline`: This command-line flag facilitates parsing output by stripping returns or newlines from the messages returned from the executed commands. Although this command facilitates parsing, it makes reading messages manually more difficult.

Error Logging in Sash

Sash does not log to any files (with the default configuration), it only prints messages on the console. The log level for Sash is configured in `$ORACLE_HOME/sdp/sash/conf/logging.properties`.

Configuring the Logging System

This chapter describes the logging framework used by OCMS. This chapter includes the following sections:

- ["Overview of Oracle Diagnostic Logging in OCMS"](#)
- ["Logging Levels"](#)
- ["Setting the Log Levels for Components"](#)

Overview of Oracle Diagnostic Logging in OCMS

OCMS uses Oracle Diagnostic Logging which is provided with Oracle Application Server. Oracle Diagnostic Logging implements APIs to be used by Oracle products to emit error diagnostics and a LogLoader tool that collects error diagnostic logs for analysis. The interface to Oracle Diagnostic Logging is `java.util.logging`.

Logging Components

OCMS defines the following logging components:

- `oracle.sdp.ocms.customer`: this logger capture all the messages that a customer would like to look at. It follows the XML format of Oracle Diagnostic Logging and it is localized.
- `oracle.sdp.ocms.application`: this logger will capture all log messages from SIP applications deployed on the server. For example, `SipServlet.log()`.

The remaining logging components are provided for debug purposes only:

- `oracle.sdp.ocms.anomalousmsg`: this logger captures faulty or unparseable SIP messages. The usable levels are FINE and FINER.
- `oracle.sdp.ocms.traffic`: this logger captures SIP messages. The usable levels are FINE and FINER.
- `oracle.sdp.ocms.config`: this logger captures all configuration and system properties.
- `oracle.sdp.ocms.statistics` and `oracle.sdp.sipcluster.util.stat`: these loggers capture statistics, such as the number of sent messages.

Filtering of Logging Information by Single Class Files

The logging framework creates loggers with the above names appended with a fully qualified name of the class that created the logger. This allows developers to filter out logging information at the level of single class files. For example, when a class

oracle.sdp.commons.MyClass creates a customer logger, the name of the logger will be:

```
"oracle.sdp.ocms.customer.oracle.sdp.commons.MyClass"
```

The configuration file contains definitions of the loggers, references to physical log files, and a definition of the *trace logger*. The *trace logger* is a logger that is automatically created whenever any of the above loggers are created. For example when a class requests a customer logger through a call to `LogFactory.getLogger(Class)`, a trace logger is also created. This allows all customer messages to be written in the trace log and log messages with throwable data to print more information in the trace log.

Log Files

All base loggers in OCMS (system, anomalousmsg, customer, traffic, config, statistics, application) use a log directory and a log file of their own under the main logging directory *sdp*. The log file is not configurable, and is an XML file named `log.xml`.

For debug messages a directory called *trace* is also created.

Logger Interfaces

The main logging from the SIP Container is to a customer log and a trace log. The customer log contains the localized customer messages, and the trace log is used for debug logging (disabled in a default installation). The customer logger name is *oracle.sdp.ocms.system*.

The following logger interfaces are implemented:

- `oracle.sdp.commons.logging.Logger`
- `oracle.sdp.commons.logging.CustomerLogger` – This interface and `oracle.sdp.commons.logging.TraceLogger` are the interfaces most classes will use in normal types of logging. It is the only interface where localized messages are used. All other interfaces are considered to be of debug type.
- `oracle.sdp.commons.logging.TraceLogger` – This interface and `oracle.sdp.commons.logging.CustomerLogger` are the interfaces most classes will use in normal types of logging. It is the only interface where localized messages are used. All other interfaces are considered to be of debug type.
- `oracle.sdp.commons.logging.MessageLogger` – Logs messages received and messages sent from the server.
- `oracle.sdp.commons.logging.ConfigurationLogger` – Logs configuration information at the start-up of a service. For example, `coreVersion = "10.1.3.4"`.
- `oracle.sdp.commons.logging.TimerLogger` – Logs timing information needed for performance tuning. For example, logging of execution time for a database call.
- `oracle.sdp.commons.logging.StatisticsLogger` – Logs statistics at regular intervals.

Logging Levels

The following logging levels are provided:

- *SEVERE* – These message indicate a serious problem that requires immediate attention from the administrator
- *WARNING* – These message indicate a potential problem that should be reviewed by the administrator.

- *INFO* – The *INFO* level designates informational messages that highlight the progress of the application at a general level. These messages indicate a major life-cycle event, such as the activation or de-activation of a primary sub-component. This is the default log level.
- *CONFIG* – These messages provide a finer level of granularity for reporting normal events. Enabling logging at this level has a minimal performance impact.
- *FINE* – These messages provide trace or debug information. Enabling logging at this level may have a small performance impact.
- *FINER*, *FINEST* – These logging levels are for debug purposes only and are not recommended for production environments.

Setting the Log Levels for Components

You can set the logging level of components dynamically through Oracle Application Server Enterprise Manager. Logging configuration changes are picked up automatically every 60 seconds and do not require a server reboot.

To set log levels:

1. In Enterprise Manager select **Administration**.
2. Select the Go to task icon next to 'Logger Configuration'.
3. In the Logger Configuration page enter the name of the logging interface (for example, 'oracle.sdp.ocms.traffic') in the search field and click **Go**.
4. Change the logging level as required.

By default, all of the component loggers except for *traffic* and *anomalousmsg* are set to *INFO*. The *traffic* and *anomalousmsg* logs are set to *OFF*. As a consequence, a system using these default settings will not write any messages to the *traffic* or *anomalousmsg* logs. Refer to "[Logging Levels](#)" for more information.

Deploying Applications

This chapter, through the following sections, describes deploying SIP servlet applications to application servers:

- ["Overview of SIP Servlet Applications"](#)
- ["Deploying SIP Applications"](#)

Overview of SIP Servlet Applications

A SIP application can be comprised of servlets, class files, static resources and content, along with descriptive meta information which unifies these elements. As specified in JSR-116, a SIP servlet application is a structured hierarchy of directories. For converged applications (those comprised of both HTTP and SIP), the root of the hierarchy serves as the document root for files published from a Web server.¹ Within the hierarchy of the SIP servlet application, the WEB-INF directory stores the directories containing the `sip.xml` deployment descriptor file (`/WEB-INF/sip.xml`), the utility classes available to the application loader class (`/WEB-INF/classes`), and the directories containing the JAR files, servlets, beans, and utility classes useful to the Web application (`/WEB-INF/lib`).

The Deployment Descriptor File

The SIP application's deployment descriptor file, `sip.xml`, is comprised of the following elements:

```
<sip-app>
  <context-param>...</context param>
  <display-name>...</display-name>
  <distributable>...</distributable>
  <session-config>...</session-config>
  <servlet>...</servlet>
  <servlet mappings>...</servlet mappings>
  <listener>...</listener>
  <security-constraint>...</security-constraint>
</sip-app>
```

The application's common parameters are set in within the `<context-param>` element. The `<session-config>` element defines the application sessions. The `<servlet>` element defines the servlet for the container through its `<servlet-name>` and `<servlet-class>` child elements. The `<servlet mappings>` element defines how the application's servlets respond to requests. The application's life cycle listener classes and error handling are defined within the

¹ SIP Servlet API, Version 1.0

<listener> element. Security is declared for each servlet using the <security-constraint> element. Refer to *Oracle Communication and Mobility Server Developer's Guide* for a full description of the `sip.xml` file's elements.

Development to Deployment

The cycle from development to deployment of a SIP servlet application is as follows:

- Creating a SIP servlet by extending `javax.servlet.sip.SipServlet` and then by overriding the required methods for a particular service.
- Defining the SIP servlet application's initialization parameters (servlet definitions) and invocation rules (servlet mappings).
- Creating the deployment descriptor file (`sip.xml`) and `web.xml` file.
- Building and packaging the application files and the `sip.xml` and `web.xml` files into a Web Archive format file (WAR file).
- Packaging the WAR file into an Enterprise Archive (EAR) file.
- Deploying the EAR file to OC4J.

Once the SIP servlet application has been successfully deployed and started on OC4J, view the log files and test it using a softphone client such as the Oracle Communicator client.

Deploying SIP Applications

OCMS accepts Enterprise Archive (EAR) files and Web Application Archive (WAR), but not SAR files. You must package a WAR file as an EAR file to enable the deployment of the SIP application to OC4J.

An EAR file can contain SAR files, JAR files, Web Application Archive (WAR) and EJB modules as follows:

```
J2EEAppName.ear
  META-INF/
    application.xml
    orion-application.xml (optional)
WebModuleName.war
  static HTML files, such as index.html
  JSP pages
  images
  WEB-INF/
    web.xml (Standard J2EE descriptor)
    orion-web.xml (optional OC4J Web descriptor)
  classes/
    servlet classes, according to package
  lib/
    JAR files for dependency classes
SIPApplicationName.sar
  WEB-INF/
    sip.xml (deployment descriptor)
    web.xml (for converged applications)
  classes/
    servlet classes, according to package
  lib/*.JAR
  JAR files for dependency classes
```

For more information on deployment and EAR application structure, see *Oracle Containers for J2EE Developer's Guide*.

Deploying, Undeploying, and Redeploying SIP Applications Using Oracle Application Server Control

Application Server Control provides a JSR 88-based deployment wizard, accessed by clicking the **Deploy** button on the *Applications* page. This wizard enables deployment and redeployment of J2EE applications and includes both task-oriented deployment plan editors for assigning or mapping the common deployment descriptors at deployment time as well as a generic deployment plan editor that enables you to access all deployment descriptors for advanced configuration. For information on undeploying and redeploying applications, see "[Deploying, Undeploying, and Redeploying SIP Servlet Applications with Application Server Control](#)".

Note: Although you can change a deployment plan using Application Server Control, you cannot use Application Server Control to alter the `sip.xml` deployment descriptor file. For information on deployment plans, see *Oracle Containers for J2EE Deployment Guide*.

Enterprise applications deployed beneath the [Subscriber Data Services](#) application inherit security infrastructure and authentication-related EJBs (Enterprise Java Beans). This infrastructure is required to support authentication against the OCMS JAAS Security providers.

Note: When an application has been undeployed, its MBeans are also undeployed.

Once an application has been deployed to the OC4J SIP Server instance, you can start or stop it using the `admin_client.jar` utility by executing the following command:

```
java -jar admin_client.jar uri adminId adminPwd -start|-stop appName
```

For Oracle Application Server, the URI parameter has the following format:

```
deployer:oc4j:opmn://host.example.com:6003/ocms
```

For OC4J standalone, the URI parameter has the following format:

```
deployer:oc4j:localhost:23791
```

This section gives a brief overview of both of these options through the following topics:

- "[Deploying, Undeploying, and Redeploying SIP Servlet Applications with Application Server Control](#)"
- "[Deploying, Undeploying, and Redeploying an Application Using the admin_client.jar Utility](#)"

For more information, refer to *Oracle Containers for J2EE Deployment Guide*.

Note: SIP applications can only be deployed to OC4J if they are packaged into a J2EE-compliant EAR file. For more information, see "[Deploying Applications](#)".

Deploying, Undeploying, and Redeploying SIP Servlet Applications with Application Server Control

The Application Server Control Console provides a wizard that steps you through deploying, undeploying, and redeploying SIP applications.

Tip: Use a firewall to block all incoming SIP traffic until all of the applications have been fully deployed and the server started.

You can filter SIP traffic using a shell script, such as the following Linux script, `blockport.sh`, which uses the `iptables` tool.

```
#!/bin/bash

if [ $# != 1 ]
then
    echo "blockport.sh <port>"
    exit
fi

iptables -A INPUT -p tcp -m tcp --dport $1 -j DROP
iptables -A INPUT -p udp -m udp --dport $1 -j DROP
service iptables save

echo "Port \"$1\" blocked."
```

Likewise, you can use a shell script to enable the flow of SIP traffic once the server is running and all applications have been fully deployed. The following Linux script, `unblockport.sh`, is an example of script that enables SIP traffic:

```
#!/bin/bash

if [ $# != 1 ]
then
    echo "unblockport.sh <port>"
    exit
fi

iptables -D INPUT -p tcp -m tcp --dport $1 -j DROP
iptables -D INPUT -p udp -m udp --dport $1 -j DROP
service iptables save

echo "Port \"$1\" unblocked."
```

For more information, see *Deploying with Application Server Control Console* in *Oracle Containers for J2EE Deployment Guide*.

Deploying an Application using the Deployment Wizard

The **Deploy** button on the *Applications* page invokes the deployment wizard which guides you through the deployment process through the following pages:

- The *Select Archive* page (Figure 13–1) is the first page of the wizard. To complete this page, point OC4J to the location of the EAR (Enterprise Archive) file containing the SIP application. This page also enables you to select the option to create or apply a deployment plan, a client-side aggregation of all of the configuration data needed to deploy an archive into OC4J. If you use an existing

deployment plan, you enter its location. If you opt for new deployment plan, select the **Automatically Create a New Deployment Plan** option. Refer to ["Deploying Applications"](#) for information on EAR file structure and how to package a SIP application for deployment.

Tip: The wizard automatically creates a new deployment plan if you do not enter the location of an existing plan.

Figure 13–1 Deploying an Application: Entering the Archive Location

ORACLE Enterprise Manager 10g
Application Server Control

Setup Logs Help Logout

Select Archive Application Attributes Deployment Settings

Deploy: Select Archive

Cancel Step 1 of 3 Next

Archive

The following types of archives can be deployed: J2EE application (EAR files), Web Modules (WAR files), EJB Modules (EJB JAR files) and Resource Adapter Modules (RAR files).

Archive is present on local host. Upload the archive to the server where Application Server Control is running.
Archive Location Browse...

Archive is already present on the server where Application Server Control is running.
Location on Server
The location on server must be the absolute path or the relative path from j2ee/home

Deployment Plan

The deployment plan is an XML file that contains the deployment settings for an application. If you do not have a deployment plan, one will be created automatically during the deployment process. Later in the deployment process, you can optionally edit the deployment plan and save it for a future deployment of this application.

Automatically create a new deployment plan.
The deployment plan settings will be based on OC4J defaults and information contained in the archive

Deployment plan is present on local host. Upload the deployment plan to the server where Application Server Control is running.
Plan Location Browse...

Deployment plan is already present on server where Application Server Control is running.
Location on Server
The location on server must be the absolute path or the relative path from j2ee/home

Cancel Step 1 of 3 Next

- Clicking **Next** invokes the *Application Attributes* page (Figure 13–2). This page enables you to enter the application name and select the parent application. The application name cannot contain spaces.

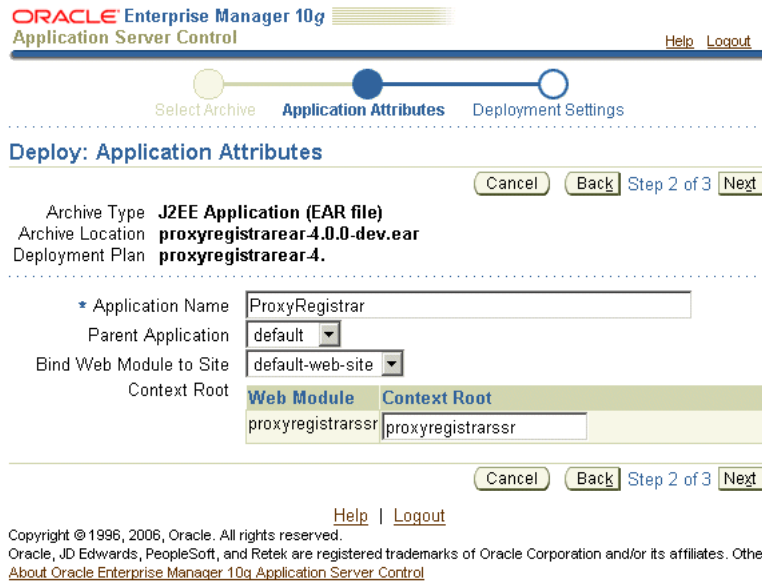
Select Subscriber Data Services as the parent application for OCMS applications requiring authentication.

Note: The SIP application becomes a child of the default application if you do not specify a parent application.

The *Application Attributes* page also enables you to set the binding of a Web application to a Web site by specifying the name portion of the `name-web-site.xml` configuration file that defines the Web site. A Web application deployed as part of a J2EE application must be bound to the Web site through which it is accessed.

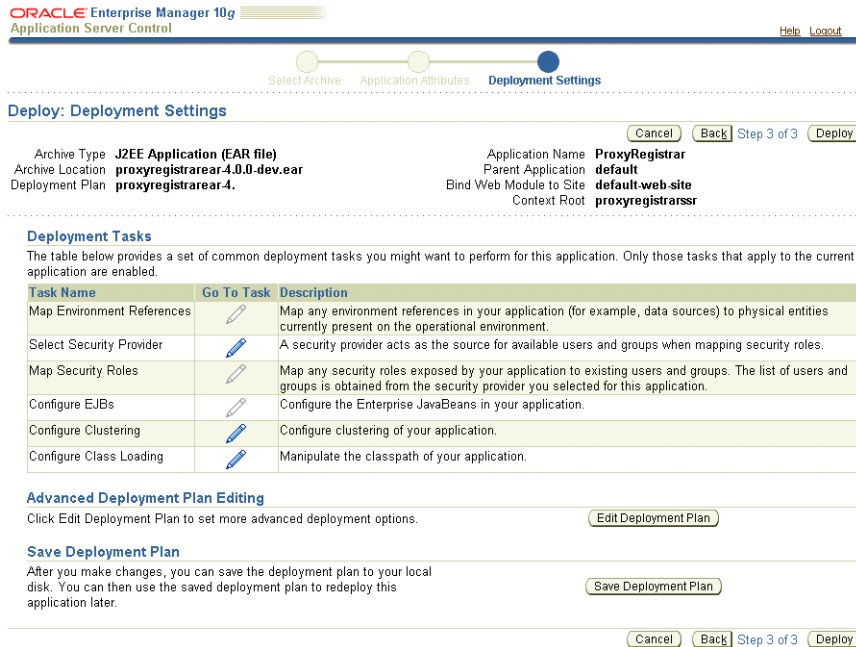
The Web module context root, which will be appended to the URL used to access the application through a Web browser, is also set as part of the process to enable Web access. This value is typically read from the `application.xml` deployment descriptor packaged with the application.

Figure 13–2 Deploying an Application: Entering the Application Name and Parent Application



- The *Deployment Settings* page (Figure 13–3) provides a tasks that enable you to edit the deployment plan.

Figure 13–3 Deploying an Application: Configuring the Deployment Settings



Complete deployment tasks as needed and then click **Deploy**. The confirmation page appears (Figure 13–4).

Figure 13–4 Confirming the Deployment

The screenshot shows the Oracle Enterprise Manager 10g Application Server Control interface. At the top, it says "ORACLE Enterprise Manager 10g Application Server Control" with "Help" and "Logout" links. Below that is a "Confirmation" section with a "Return" button. The main message states: "The Application 'ProxyRegistrar' has been successfully deployed." Underneath is a "Progress Messages" section with a scrollable log of deployment steps, including unpacking EAR and WAR files, initializing containers, and starting the application. The log ends with "Application Deployer for ProxyRegistrar COMPLETES. Operation time: 10565 msec". At the bottom of the log area is another "Return" button, and below that are "Help" and "Logout" links.

Confirmation Return

The Application "ProxyRegistrar" has been successfully deployed.

Progress Messages

```
[Jun 30, 2006 10:58:59 AM] Unpacking ProxyRegistrar.ear
[Jun 30, 2006 10:58:59 AM] Done unpacking ProxyRegistrar.ear
[Jun 30, 2006 10:58:59 AM] Unpacking proxyregistrarssr-4.0.0-dev.war
[Jun 30, 2006 10:58:59 AM] Done unpacking proxyregistrarssr-4.0.0-dev.war
[Jun 30, 2006 10:58:59 AM] Initialize D:\ORACLE_HOME\j2ee\home\applications\ProxyRegistrar.ear ends...
[Jun 30, 2006 10:58:59 AM] Starting application : ProxyRegistrar
[Jun 30, 2006 10:58:59 AM] Initializing ClassLoader(s)
[Jun 30, 2006 10:58:59 AM] Initializing EJB container
[Jun 30, 2006 10:58:59 AM] Loading connector(s)
[Jun 30, 2006 10:58:59 AM] Starting up resource adapters
[Jun 30, 2006 10:58:59 AM] Processing EJB module: locationsservice-4.0.0-dev.jar
[Jun 30, 2006 10:59:00 AM] Compiling EJB generated code
[Jun 30, 2006 10:59:09 AM] Initializing EJB sessions
[Jun 30, 2006 10:59:09 AM] Committing ClassLoader(s)
[Jun 30, 2006 10:59:09 AM] Initialize proxyregistrarssr-4.0.0-dev begins...
[Jun 30, 2006 10:59:09 AM] Initialize proxyregistrarssr-4.0.0-dev ends...
[Jun 30, 2006 10:59:09 AM] Started application : ProxyRegistrar
[Jun 30, 2006 10:59:09 AM] Binding web application(s) to site default-web-site begins...
[Jun 30, 2006 10:59:09 AM] Binding proxyregistrarssr-4.0.0-dev web-module for application ProxyRegistrar to site
default-web-site under context root proxyregistrarssr
[Jun 30, 2006 10:59:10 AM] Binding web application(s) to site default-web-site ends...
[Jun 30, 2006 10:59:10 AM] Application Deployer for ProxyRegistrar COMPLETES. Operation time: 10565 msec
```

Return

Help | Logout

Note: To ensure that the OMCS SIP servlet container responds appropriately to incoming requests, you must add any deployed application that processes requests to the Application Router's *SIPUriList* attribute. Configuring this attribute ensures that applications are added to the *ROUTE* header of incoming *INVITE*, *MESSAGE*, *PUBLISH*, *REGISTER*, or *SUBSCRIBE* requests from non-OCMS (that is, non-OCMS Communicator) SIP clients. The **Proxy Registrar** must always be the last item listed in the *SIPUriList* attribute.

Undeploying an Application Using the Deployment Wizard

You can remove (undeploy) an application by first selecting it and then by clicking the **Undeploy** button. When you undeploy an application, you likewise undeploy the MBeans registered to the application.

Likewise, if you undeploy a parent application, its the child applications are also undeployed. As a result, the parent application and all related applications must be redeployed. See *Oracle Containers for J2EE Deployment Guide* for information on when to restart OC4J when undeploying an application.

Redeploying an Application Using the Deployment Wizard

The **Redeploy** button enables you undeploy an application without restarting OC4J. Redeploying a SIP application packaged within an EAR file prompts OC4J to undeploy the previous instance; you do not have to first select the application and then click **Undeploy**.

Like deploying an application, the wizard prompts you through a three-step process for redeploying an application in which you point OC4J to the EAR file, select or create a deployment plan, select the parent application and Web bindings, and complete deployment descriptor configuration tasks.

Deploying, Undeploying, and Redeploying an Application Using the `admin_client.jar` Utility

The `admin_client.jar` command-line utility used to perform deployment-related operations on active OC4J instances in an Oracle Application Server clustered environment as well as on standalone OC4J servers.

The `admin_client.jar` utility is installed by default in `ORACLE_HOME/j2ee/home` in an OC4J instance. OC4J must be started before this utility can be used.

Deploying an Application Using `admin_client.jar`

To deploy an EAR, use the `-deploy` command with the EAR-specific context as follows:

```
java -jar admin_client.jar uri adminId adminPassword -deploy -file
path/filename -deploymentName appName [-bindAllWebApps [webSiteName]]
[-targetPath path] [-parent appName] [-deploymentDirectory path]
-enableIIOP [-iiopClientJar path/filename]
```

Undeploying an Application Using `admin_client.jar`

To undeploy an application:

```
java -jar admin_client.jar uri adminId adminPassword -undeploy appName
```

Redeploying an Application Using `admin_client.jar`

To redeploy a previously deployed archive, use the `-redeploy` command and with the following syntax:

```
java -jar admin_client.jar uri adminId adminPassword -redeploy -file
path/filename -deploymentName appName [-keepSettings] [-sequential]
```

Refer to *Deploying with the `admin_client.jar` Utility* in *Oracle Containers for J2EE Deployment Guide* for more information on the `-redeploy` command subswitches.

Deploying the SIP Application Using the `admin_client.jar` Command-Line Utility

You can deploy an EAR file from the command line using the `admin_client.jar` utility as follows:

```
java -jar admin_client.jar uri adminId adminPassword -deploy -file
path/filename -deploymentName appName [-bindAllWebApps [webSiteName]]
[-targetPath path] [-parent appName] [-deploymentDirectory path]
-enableIIOP [-iiopClientJar path/filename] [-deploymentPlan path/filename]
```

For more information undeploying and redeploying applications using the `admin_client.jar` utility, see "[Deploying, Undeploying, and Redeploying an Application Using the `admin_client.jar` Utility](#)". See also *Oracle Containers for J2EE Configuration and Administration Guide* and *Oracle Containers for J2EE Deployment Guide*.

Note: The `admin_client.jar` utility is installed by default in `ORACLE_HOME/j2ee/home` in an OC4J instance. OC4J must be started before this utility can be used.

Supported Protocols, RFCs, and Standards

This appendix lists the protocols, RFCs, and standards supported by Oracle Communication and Mobility Server in the following sections:

- ["SIP Servlet Container"](#)
- ["Presence Server"](#)

SIP Servlet Container

The following sections list the RFCs, drafts, and specification requests supported by the OCMS SIP Servlet container:

- [RFCs](#)
- [Drafts](#)
- [Specification Requests](#)

RFCs

RFC 2246 The TLS Protocol

RFC 2543 SIP: Session Initiation Protocol

RFC 2782 A DNS RR for Specifying the Location of Services (DNS SRV)

RFC 2806 URLs for Telephone Calls

RFC 2976 The SIP INFO Method

RFC 3261 SIP: Session Initiation Protocol

RFC 3262 Reliability of Provisional Responses in the Session Initiation Protocol

RFC 3263 Session Initiation Protocol (SIP): Locating SIP Servers

RFC 3264 An Offer/ Answer Model with the Session Description Protocol (SDP)

RFC 3265 Session Initiation Protocol (SIP)-Specific Event Notification

RFC 3266 Support for IPv6 in Session Description Protocol (SDP)

RFC 3311 The Session Initiation Protocol (SIP) UPDATE Method

RFC 3312 Integration of Resource Management and Session Initiation Protocol (SIP)

RFC 3323 A Privacy Mechanism for the Session Initiation Protocol (SIP)

RFC 3325 Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks

RFC 3326 The Reason Header Field for the Session Initiation Protocol (SIP)

RFC 3327 Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts

RFC 3420 Internet Media Type message/sipfrag

RFC 3428 Session Initiation Protocol (SIP) Extension for Instant Messaging

RFC 3455 Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3rd-Generation Partnership Project (3GPP)

RFC 3489 STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)

RFC 3515 The Session Initiation Protocol (SIP) Refer Method

RFC 3556 Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth

RFC 3581 An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing

RFC 3605 Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)

RFC 3608 Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration

RFC 3665 Session Initiation Protocol (SIP) Basic Call Flow Examples

RFC 3725 Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)

RFC 3761 The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)

RFC 3824 Using E.164 numbers with the Session Initiation Protocol (SIP)

RFC 3840 Caller Preferences for the Session Initiation Protocol (SIP): partial support

RFC 3891 The Session Initiation Protocol (SIP) "Replaces" Header

RFC 3903 Session Initiation Protocol (SIP) Extension for Event State Publication

RFC 3911 The Session Initiation Protocol (SIP) "Join" Header

RFC 3959 The Early Session Disposition Type for the Session Initiation Protocol (SIP)

RFC 3966 The tel URI for Telephone Numbers

RFC 4028 Session Timers in the Session Initiation Protocol (SIP)

RFC 4320 Actions Addressing Identified Issues with the Session Initiation Protocol's (SIP) Non-INVITE Transaction

RFC 4321 Problems Identified Associated with the Session Initiation Protocol's (SIP) Non-INVITE Transaction

RFC 4483 A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages

Drafts

campen-sipping-stack-loop-detect An Efficient Loop Detection Algorithm for SIP Proxies

lawrence-maxforward-problems Problems with Max-Forwards Processing (and Potential Solutions)

ietf-sip-connect-reuse Connection Reuse in the Session Initiation Protocol (SIP)

sparks-sipping-max-breadth Limiting the Damage from Amplification Attacks in SIP Proxies

ietf-sip-fork-loop-fix Addressing an Amplification Vulnerability in Forking Proxies

Specification Requests

JSR 116 SIP Servlet 1.0

Presence Server

The following sections list protocols, RFCs, drafts, and standards supported by the OCMS Presence Server and its components:

- [RFCs](#)
- [Drafts Referenced in the Composition Policies](#)
- [XDMS Server](#)
- [Authorization and Privacy Filtering](#)
- [Presence Data Modeling and Processing](#)
- [OMA Extensions](#)
- [Hard State via XCAP](#)

RFCs

RFC 2778 A Model for Presence and Instant Messaging

RFC 2779 Instant Messaging / Presence Protocol Requirements

RFC 3265 Session Initiation Protocol (SIP)-Specific Event Notification

RFC 3856 A Presence Event Package for the Session Initiation Protocol (SIP)

RFC 3857 A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)

RFC 3858 An Extensible Markup Language (XML) Based Format for Watcher Information

RFC 3859 Common Profile for Presence (CPP)

RFC 3863 Presence Information Data Format (PIDF)

RFC 3903 Session Initiation Protocol (SIP) Extension for Event State Publication

RFC 4119 A Presence-based GEOPRIV Location Object Format

RFC 4479 A Data Model for Presence

RFC 4480 RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)

RFC 4481 Timed Presence Extensions to the Presence Information Data Format (PIDF) to Indicate Status Information for Past and Future Time Intervals

RFC 4825 The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)

Note: RFC 4825 is partially supported. There is no support for partial document manipulation (XPath).

4827 An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Manipulating Presence Document Contents

Drafts Referenced in the Composition Policies

The following drafts are transparent to the server, but are important for clients. These drafts are specifically referenced in composition policies such as the OMA composition policy.

"RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)", draft-ietf-simple-rpid-10

"CIPID: Contact Information in Presence Information Data Format", draft-ietf-simple-cipid-07

"Timed Presence Extensions to the Presence Information Data Format (PIDF) to Indicate Presence Information for Past and Future Time Intervals", draft-ietf-simple-future-05

Partial Publish and Partial Notify:

- "Session Initiation Protocol (SIP) extension for Partial Notification of Presence Information", draft-ietf-simple-partial-notify
- "Presence Information Data format (PIDF) Extension for Partial Presence", draft-ietf-simple-partial-pidf-format
- "Publication of Partial Presence Information", draft-ietf-simple-partial-publish
- "Presence Authorization Rules", draft-ietf-simple-presence-rules-10

"OMA Extensions to the Presence Data Model", OMA-TS-Presence_SIMPLE-V1_0-20051122-C

"OMA extensions to geopriv common policy", OMA-TS-XDM_Core-V1_0-20051122-C

"OMA Extensions to Presence Rules", OMA-TS-Presence_SIMPLE_XDM-V1_0-20051122-C

XDMS Server

"The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-ietf-simple-xcap-08

"A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-07

"A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", draft-ietf-sip-content-indirect-mech-05

Exceptions:

The current XCAP application does not support partial document put or get.

Authorization and Privacy Filtering

"A Document Format for Expressing Privacy Preferences", draft-ietf-geopriv-common-policy-05

"Presence Authorization Rules", draft-ietf-simple-presence-rules-04
draft-ietf-geopriv-common-policy-05

Presence Data Modeling and Processing

"A Data Model for Presence", draft-ietf-simple-presence-data-model-07
"A Processing Model for Presence",
draft-rosenberg-simple-presence-processing-model-01

OMA Extensions

[urn:oma:params:xml:ns:pidf:oma-pres] OMA extensions to the presence data model
(OMA-TS-Presence_SIMPLE-V1_0-20051122-C)
[urn:oma:params:xml:ns:common-policy] OMA extensions to geopriv common policy
(OMA-TS-XDM_Core-V1_0-20051122-C)
[urn:oma:params:xml:ns:pres-rule] OMA extensions to presence rules
(OMA-TS-Presence_SIMPLE_XDM-V1_0-20051122-C)

Hard State via XCAP

"An Extensible Markup Language (XML) Configuration Access Protocol (XCAP)
Usage for Manipulating Presence Document Contents",
draft-ietf-simple-xcap-pidf-manipulation-usage-02

Third-Party Licensing

This Appendix lists third-party licensing information for software included in this release.

Third-Party Licenses

Third-party software license information for software included in this release is listed in [Table B-1](#).

Table B-1 *Third-party licenses*

Library/Component	License
Saxon B	http://www.mozilla.org/MPL/MPL-1.1.html
Commons HttpClient 3.1	http://apache.org/licenses/
Commons Logging1.0.4	http://apache.org/licenses/
Commons Codec 1.3	http://apache.org/licenses/
Commons Lang 2.1	http://apache.org/licenses/
Log4J 1.2.9	http://apache.org/licenses/
XMLBeans (xbean.jar) 2.3.0	http://apache.org/licenses/
DNSJava 2.0.1	View DNSJava License
Jline 0.9.1	View Jline License
JSR116	View JSR116 License
JSR289	View JSR289 License
spring 2.5.1	http://apache.org/licenses/
stax-api (JSR 173) 1.0.1	http://apache.org/licenses/
jainsip (JSR 32) 1.2	http://www.jcp.org/en/home/index
Boost	http://www.boost.org/LICENSE_1_0.txt
Wtl 7.5	http://sourceforge.net/projects/wtl , http://www.microsoft.com/resources/sharedsource/licensingbasics/permisivelicense.mspx

Table B-1 (Cont.) Third-party licenses

Library/Component	License
RSA	<p data-bbox="683 260 1338 312">/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.</p> <p data-bbox="683 327 1369 443">License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.</p> <p data-bbox="683 457 1344 564">License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.</p> <p data-bbox="683 579 1333 686">RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.</p> <p data-bbox="683 701 1347 747">These notices must be retained in any copies of any part of this documentation and/or software. */</p>
RFC 2617	<p data-bbox="683 768 976 795">http://rfc.net/rfc2617.html</p> <p data-bbox="683 810 915 837">IETF's IPR policy is at</p> <p data-bbox="683 852 1114 879">http://www.ietf.org/rfc/rfc3978.txt and</p> <p data-bbox="683 894 1068 921">http://www.ietf.org/rfc/rfc3978.txt</p>
GUI header file (atlgdix.h, menubutton.h)	<p data-bbox="683 936 1032 963">Additional GDI/USER wrappers</p> <p data-bbox="683 978 1149 1005">Written by Bjarke Viksoe (bjarke@viksoe.dk)</p> <p data-bbox="683 1020 1089 1047">Copyright (c) 2001-2002 Bjarke Viksoe.</p> <p data-bbox="683 1062 1214 1089">Thanks to Daniel Bowen for COffscreenDrawRect.</p> <p data-bbox="683 1104 1360 1199">This code may be used in compiled form in any way you desire. This file may be redistributed by any means PROVIDING it is not sold for profit without the authors written consent, and providing that this notice and the authors name is included.</p> <p data-bbox="683 1213 1369 1314">This file is provided "as is" with no expressed or implied warranty. The author accepts no liability if it causes any damage to you or your computer whatsoever. It's free, so don't hassle me about it. Beware of bugs.</p>
String handling (stdstring.h)	<p data-bbox="683 1335 829 1362">COPYRIGHT:</p> <p data-bbox="683 1377 1369 1503">1999 Joseph M. O'Leary. This code is free. Use it anywhere you want. Rewrite it, restructure it, whatever. Please don't blame me if it makes your \$30 billion dollar satellite explode in orbit. If you redistribute it in any form, I'd appreciate it if you would leave this notice here.</p> <p data-bbox="683 1518 1117 1545">If you find any bugs, please let me know:</p> <p data-bbox="683 1560 932 1587">jmoleary@earthlink.net</p> <p data-bbox="683 1602 1084 1629">http://home.earthlink.net/~jmoleary</p>
GUI Slider implementation (slider.js)	<p data-bbox="683 1650 1206 1677">Slider000118 by Christiaan Hofman, January 2000</p> <p data-bbox="683 1692 1338 1734">You may use or modify this code provided that this copyright notice appears on all copies.</p>

Index

A

Aggregation Proxy, 2-3
 authenticating XCAP traffic, 7-12
 configuring HTTPS connections, 7-14
application layer, 1-4
Application Router
 about, 1-13, 1-14
 configuring, 3-17
 incremental mode, 1-13, 1-14
 standard mode, 1-13, 1-14
applications
 deployment, 13-3
architecture, 1-3, 1-4
Associating nodes with OPMN, 5-3
 Discovery Server Method, 5-4
 Dynamic Discovery Method, 5-3
Authentication and Authorization Data, 1-15

C

cluster, 5-2, 5-3
 start, 5-5
 stop, 5-5
 verify status, 5-5
CommandService MBean
 executing Sash commands, 3-14, 4-4
Configuration Recommendations, 2-13
Contact Management API, 1-2

D

data layer, 1-4
Deployment Topologies, 2-1, 2-3, 2-5, 2-9, 2-11
disabling high availability in the application, 5-12
Discovery Server Method, 5-4
drafts, A-1, A-3
Dynamic Discovery Method, 5-3

E

Edge Proxy, 2-2
 about, 1-8
 high availability, 5-6
edgeproxy Mbean, 5-6
ENUM Service, 1-10

F

Fetcher, 1-11

H

High Availability, 2-1
 Application session data replication, 5-10
 associating nodes with OPMN, 5-3, 5-4
 clustering, 5-3
 configuring, 5-1
 Edge Proxy, 5-6
 SIP Servlet applications, 5-8, 5-11, 5-12
 SIP servlet containers, 5-5

I

IETF, A-1, A-3

J

JAAS, 4-1

L

Location Lookup Data, 1-15
Location Lookup Service, 1-10
logging, 1-15
 error logging in Sash, 11-10
login
 configuring account locking, 3-13, 4-3

N

NOTIFY, 1-11

O

OCMS, 1-2
 system components, 1-4
 three layer model, 1-3, 1-4
OCMS as a Highly Available SIP Network, 2-3
OCMS as a Presence Server, 2-5
OCMS as an Instant Messaging Platform, 2-9
OCMS Testing Environment, 2-11
opmn.xml, 5-4
Oracle Communication and Mobility Server, 1-2

- system components, 1-4
- three layer model, 1-3, 1-4
- orion-application.xml, 5-10
- Overload Policy
 - setting threshold levels for capacity, 5-13

P

- Parlay X
 - Presence custom error codes, 9-7
- Parlay X Web Services, 9-1
- Presence
 - about, 1-11, 1-12
 - configuring, 7-1
- Presence User Agent, 1-11
- Presence Web Services interfaces, 9-1, 10-1
 - code examples, 9-3, 10-3
 - using, 9-3, 10-3
- PresenceConsumer interface, 9-2, 10-2
- PresenceNotification interface, 9-2, 10-2
- PresenceSupplier interface, 9-2
- Presentity, 1-11
- protocols, A-1, A-3
- proxy layer, 1-3
- Proxy Registrar, 2-3
 - about, 1-9
 - configuring, 3-16
- PUBLISH, 1-11

R

- RADIUS Login Module, 4-2
- replication, 5-10
- RFCs, A-1, A-3

S

- Sash
 - command and subcommands, 11-2
 - connection to external instances, 11-2
 - error logging, 11-10
- Session Replication, 1-15
- SIP Application Servers, 2-2
- SIP applications
 - about, 1-5
 - deployment through Application Server Control, 13-3
 - high availability, 5-8, 5-11, 5-12
 - inheriting authentication and security from Subscriber Data Services, 3-13, 4-3
 - setting aliases, 3-10
 - setting the default application, 3-7
 - typical, 1-6
 - upgrading, 5-12
- SIP servlet container
 - about, 1-7
 - configuring with the sipservletcontainer MBean, 3-5
 - high availability, 5-5
- SIP servlets, 1-5
- sip.xml, 5-9, 5-11

- basic structure of deployment descriptor file, 13-1
- specification requests, A-1, A-3
- Standards, A-1, A-3
- STUN
 - configuring the STUN server, 1-2, 3-12, 7-14
- Subscriber, 1-11
- Subscriber Data Services, 1-15, 3-13, 4-3
- system usage statistics, 6-1

T

- testing environment, 2-11
- third-party load balancer, 2-2

U

- unpublish, 9-5, 10-3
- Upgrading SIP applications, 5-12
- User Data, 1-15
- User Dispatcher, 1-2, 7-14
- users
 - authentication against RADIUS authentication system, 4-2
 - bulk provisioning, 11-12
 - provisioning users using Sash, 11-7
 - provisioning using the CommandService MBean, 11-8

W

- Watcher, 1-11
- web.xml, 5-9, 5-11

X

- XDMS
 - provisioning with Sash, 11-10
 - provisioning with the CommandService MBean, 11-10
- XDMS Server, 2-6