Oracle® Communications WebRTC Session Controller

System Administrator's Guide

Release 7.0

E40973-01

November 2013

ORACLE®

Oracle Communications WebRTC Session Controller System Administrator's Guide, Release 7.0

E40973-01

Contents

Preface xi

Audience xi
Related Documents xi
Documentation Accessibility xi

Part I Configuring WebRTC Session Controller

1 WebRTC Session Controller Configuration Overview

About the Oracle WebLogic Platform 1-1
Overview of Configuration and Administration Tools 1-1
Administration Console 1-1
WebLogic Scripting Tool 1-2
WebRTC Session Controller Console 1-2
Additional Configuration Methods 1-2
Editing Configuration Files 1-2
Custom JMX Applications 1-3
Common Configuration Tasks 1-3

2 Configuring WebRTC Session Controller Signaling Properties and Media Nodes

About WebRTC Session Controller Console Configuration 2-1
About Signaling Engine Properties 2-1
About Media Engine Nodes Configuration and Status 2-2
Accessing the WebRTC Session Controller Console Configuration Tab 2-2
Configuring Signaling Engine Parameters 2-3
Configuring Media Engine Nodes 2-3
Adding Media Engine Nodes 2-3
Removing Media Engine Nodes 2-4
Blocking and Unblocking Media Node Traffic 2-4
Setting Media Node Information Auto Refresh 2-4

3 Using the Administration Console and WLST

Accessing the Administration Console 3-1
Locking and Persisting the Configuration 3-2

Using WLST (JMX) to Configure WebRTC Session Controller ... 3-3
Configuring the SIP Container with WLST ... 3-3
Managing Configuration Locks ... 3-3
Configuration MBeans for the SIP Servlet Container ... 3-4
Locating the SIP Container MBeans ... 3-5
Configuring the WebRTC Session Controller Application with WLST ... 3-6
Managing Configuration Locks ... 3-6
Configuration MBeans for WebRTC Session Controller ... 3-6
Accessing WebRTC Session Controller Application MBeans ... 3-7
WLST Configuration Examples ... 3-7
Invoking WLST ... 3-7
WLST Template for Configuring Container Attributes ... 3-8
Creating and Deleting MBeans ... 3-8
WebRTC Session Controller Code Sample ... 3-9
Setting Logging Levels ... 3-9
Startup Sequence for a WebRTC Session Controller Domain ... 3-10
Startup Command Options ... 3-11
Reverting to the Original Boot Configuration ... 3-12

4 Configuring WebRTC Session Controller Authentication

About WebRTC Session Controller Security Schemes ... 4-1
About Provisioning WebRTC Session Controller Guest Access ... 4-1
Configuring the WebLogic Server Guest Access Provider ... 4-1
Configuring the WebRTC Session Controller Guest Access Application ... 4-2
About Provisioning WebRTC Session Controller HTTP Access ... 4-3
Configuring the WebLogic Server HTTP Authentication Provider ... 4-3
Configuring the WebRTC Session Controller HTTP Access Application ... 4-4
About Provisioning WebRTC Session Controller OAuth Access ... 4-4
Configuring the WebLogic Server OAuth Access Provider ... 4-5
Configuring the WebRTC Session Controller OAuth Access Application ... 4-6

5 Configuring WebRTC Session Controller Diameter Rx to PCRF Integration

About the WebRTC Session Controller Rx Interface ... 5-1
Overview of Diameter Rx Protocol Configuration ... 5-1
Installing the Diameter Domain Template ... 5-1
Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol ... 5-2
Configuring Two-Way SSL for Diameter TLS Channels ... 5-4
Configuring and Using SCTP for Diameter Messaging ... 5-4
Configuring Diameter Nodes ... 5-5
Creating a New Node Configuration (General Node Configuration) ... 5-5
Configuring Diameter Applications ... 5-7
Configuring the Rx Client Application ... 5-7
Configuring Peer Nodes ... 5-7
Configuring Routes ... 5-8
Troubleshooting Diameter Configurations ... 5-9

6 Configuring WebRTC Session Controller Media Engine

About WebRTC Session Controller Media Engine Configuration 6-1
Common Media Engine Configuration Tasks 6-1
Configuring Permissions, Users and Authorization 6-2
Enabling Media Engine Interfaces and Protocols 6-2
Enabling Media Engine Services 6-2
Configuring Accounting and Archiving Services 6-3
Configuring Media Engine Domain Name Systems 6-3
Configuring SIP Servers 6-3

7 Configuring WebRTC Session Controller Container Properties

Configure General SIP Application Server Properties 7-1
Adding Servers to the WebRTC Session Controller Cluster 7-2
Configuring Timer Processing 7-2
Configuring Timer Affinity (Optional) 7-2
Configuring NTP for Accurate SIP Timers 7-3

8 Configuring SIP Data Tier Partitions and Replicas

Overview of SIP Data Tier Configuration 8-1
datatier.xml Configuration File 8-1
Configuration Requirements and Restrictions 8-2
Best Practices for Configuring and Managing SIP Data Tier Servers 8-2
Example SIP Data Tier Configurations and Configuration Files 8-3
SIP Data Tier with One Partition 8-4
SIP Data Tier with Two Partitions 8-4
SIP Data Tier with Two Partitions and Two Replicas 8-4
Monitoring and Troubleshooting SIP Data Tier Servers 8-5

9 Configuring Network Connection Settings

Overview of Network Configuration 9-1
Configuring External IP Addresses in Network Channels 9-2
About IPv4 and IPv6 Support 9-2
Enabling DNS Support 9-3
Configuring Network Channels for SIP or SIPS 9-3
Reconfiguring an Existing Channel 9-3
Creating a New SIP or SIPS Channel 9-4
Configuring Custom Timeout, MTU, and Other Properties 9-5
Configuring SIP Channels for Multihomed Machines 9-7
Configuring Engine Servers to Listen on Any IP Interface 9-7
Configuring Static Source Port for Outbound UDP Packets 9-7
Configuring Unique Listen Address Attributes for SIP Data Tier Replicas 9-8

10 Configuring Server Failure Detection

Overview of Failover Detection 10-1
WlssEchoServer Failure Detection 10-1

Forced Shutdown for Failed Replicas 10-2
WlssEchoServer Requirements and Restrictions 10-2
Starting WlssEchoServer on SIP Data Tier Server Machines 10-2
Enabling and Configuring the Heartbeat Mechanism on Servers 10-3

11 Using the Engine Tier Cache

Overview of Engine Tier Caching 11-1
Configuring Engine Tier Caching 11-1
Monitoring and Tuning Cache Performance 11-2

12 Configuring Coherence

Overview of WebRTC Session Controller Coherence Implementation 12-1
Configuring Coherence 12-1

13 Upgrading Production WebRTC Session Controller Software

Overview of System Upgrades 13-1
Requirements for Upgrading a Production System 13-1
Upgrading to a New Version of WebRTC Session Controller 13-2
Configure the Load Balancer 13-2
Configure the New Engine Tier Cluster 13-3
Define the Cluster-to-Load Balancer Mapping 13-3
Duplicate the SIP Servlet and WebRTC Session Controller Configurations 13-4
Upgrade Engine Tier Servers and Target Applications to the New Cluster 13-5
Applying Software Patches and Updates 13-6

Part II Monitoring and Troubleshooting

14 Logging SIP Requests and Responses

Overview of SIP Logging 14-1
Defining Logging Servlets in sip.xml 14-2
Configuring the Logging Level and Destination 14-2
Specifying the Criteria for Logging Messages 14-2
Using XML Documents to Specify Logging Criteria 14-2
Using Servlet Parameters to Specify Logging Criteria 14-3
Specifying Content Types for Unencrypted Logging 14-4
Enabling Log Rotation and Viewing Log Files 14-5
trace-pattern.dtd Reference 14-5
Adding Tracing Functionality to SIP Servlet Code 14-7
Order of Startup for Listeners and Logging Servlets 14-8

15 Avoiding and Recovering From Server Failures

Failure Prevention and Automatic Recovery Features 15-1
Overload Protection 15-1
Redundancy and Failover for Clustered Services 15-2
Automatic Restart for Failed Server Instances 15-2

Managed Server Independence Mode 15-2
Automatic Migration of Failed Managed Servers 15-2
Geographic Redundancy for Regional Site Failures 15-3
Directory and File Backups for Failure Recovery 15-3
Enabling Automatic Configuration Backups 15-3
Storing the Domain Configuration Offline 15-4
Backing Up Server Start Scripts 15-5
Backing Up Logging Servlet Applications 15-5
Backing Up Security Data 15-5
Backing Up the WebLogic LDAP Repository 15-5
Backing Up SerializedSystemIni.dat and Security Certificates 15-6
Backing Up Additional Operating System Configuration Files 15-6
Restarting a Failed Administration Server 15-6
Restarting an Administration Server on the Same System 15-7
Restarting an Administration Server on Another System 15-7
Restarting Failed Managed Servers 15-8

16 Tuning JVM Garbage Collection for Production Deployments

Goals for Tuning Garbage Collection Performance 16-1
Modifying JVM Parameters in Server Start Scripts 16-1
Tuning Garbage Collection with Oracle JDK 16-2

17 Avoiding JVM Delays Caused By Random Number Generation

Avoiding JVM Delays Caused by Random Number Generation 17-1

Part III Reference

18 Engine Tier Configuration Reference (sipserver.xml)

Overview of sipserver.xml 18-1
Editing sipserver.xml 18-1
Steps for Editing sipserver.xml 18-1
XML Schema 18-2
Example sipserver.xml File 18-2
XML Element Description 18-2
enable-timer-affinity 18-2
overload 18-2
Selecting an Appropriate Overload Policy 18-4
Overload Control Based on Session Generation Rate 18-4
Overload Control Based on Capacity Constraints 18-5
Two Levels of Overload Protection 18-5
message-debug 18-5
proxy—Setting Up an Outbound Proxy Server 18-5
t1-timeout-interval 18-7
t2-timeout-interval 18-7
t4-timeout-interval 18-7
timer-b-timeout-interval 18-7

timer-f-timeout-interval 18-7
max-application-session-lifetime 18-8
enable-local-dispatch 18-8
cluster-loadbalancer-map 18-8
default-behavior 18-9
default-servlet-name 18-9
retry-after-value 18-10
sip-security 18-10
route-header 18-10
engine-call-state-cache-enabled 18-10
server-header 18-11
server-header-value 18-11
persistence 18-11
use-header-form 18-12
enable-dns-srv-lookup 18-13
connection-reuse-pool 18-13
globally-routable-uri 18-14
domain-alias-name 18-14
enable-rport 18-15
image-dump-level 18-15
stale-session-handling 18-16
enable-contact-provisional-response 18-16

19 SIP Data Tier Configuration Reference (datatier.xml)

Overview of datatier.xml 19-1
Editing datatier.xml 19-1
XML Schema 19-1
Example datatier.xml File 19-1
XML Element Description 19-2

20 Diameter Configuration Reference (diameter.xml)

Overview of diameter.xml 20-1
Graphical Representation 20-1
Editing diameter.xml 20-2
Steps for Editing diameter.xml 20-3
XML Schema 20-3
Example diameter.xml File 20-3
XML Element Description 20-3
configuration 20-3
target 20-3
host 20-4
realm 20-4
address 20-4
port 20-4
tls-enabled 20-4
sctp-enabled 20-4
debug-enabled 20-5

message-debug-enabled 20-5
application 20-5
class-name 20-5
param* 20-5
name 20-5
value 20-5
peer-retry-delay 20-5
allow-dynamic-peers 20-5
request-timeout 20-5
watchdog-timeout 20-5
include-origin-state-id 20-6
supported-vendor-id+ 20-6
peer+ 20-6
host 20-6
address 20-6
port 20-6
protocol 20-6
route 20-6
realm 20-6
application-id 20-6
action 20-7
server+ 20-7
default-route 20-7
action 20-7
server+ 20-7

Preface

This book describes system administration tasks for Oracle Communications WebRTC Session Controller.

Audience

This book is intended for system administrators who configure and manage WebRTC Session Controller implementations. Service providers use WebRTC Session Controller to make their communications services available to WebRTC-enabled web browsers and applications.

Related Documents

For more information, see the following documents in:

- *Oracle Communications WebRTC Session Controller Concepts*
- *Oracle Communications WebRTC Session Controller Security Guide*
- *Oracle Communications WebRTC Session Controller Extension Developer's Guide*
- *Oracle Communications WebRTC Session Controller Web Application Developer's Guide*

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Part I

Configuring WebRTC Session Controller

This part provides information on configuring the Oracle Communications WebRTC Session Controller Signaling Engine properties, Media Engine nodes, Diameter Rx to PCRF integration, and the Media Engine.

This part contains the following chapters:

- WebRTC Session Controller Configuration Overview
- Configuring WebRTC Session Controller Signaling Properties and Media Nodes
- Using the Administration Console and WLST
- Configuring WebRTC Session Controller Authentication
- Configuring WebRTC Session Controller Diameter Rx to PCRF Integration
- Configuring WebRTC Session Controller Media Engine
- Configuring WebRTC Session Controller Container Properties
- Configuring SIP Data Tier Partitions and Replicas
- Configuring Network Connection Settings
- Configuring Server Failure Detection
- Using the Engine Tier Cache
- Configuring Coherence
- Upgrading Production WebRTC Session Controller Software

1

WebRTC Session Controller Configuration Overview

This chapter introduces Oracle Communications WebRTC Session Controller configuration and administration.

About the Oracle WebLogic Platform

WebRTC Session Controller is based on Oracle WebLogic Server. Many system-level configuration tasks are the same for both products. This guide addresses system-level configuration tasks that are unique to WebRTC Session Controller, such as tasks related to network and security configuration and cluster configuration for the engine and SIP data tiers.

WebLogic server configuration and other basic configuration tasks such as logging are addressed in the WebLogic Server documentation. This guide will refer you to the WebLogic documentation for information where appropriate rather than repeat that information here.

Overview of Configuration and Administration Tools

You configure the WebRTC Session Controller domain using the Administration Console or the command-line using the WebLogic Scripting Tool (WLST). Changes to certain SIP Servlet container properties require a restart of the engine tier server for the change to take affect. Configuration for SIP data tier nodes cannot be changed dynamically, so you must restart SIP data tier servers to change the number of partitions or replicas.

You configure WebRTC application behavior properties in the WebRTC Session Controller console, which is separate from the Administration Console.

Administration Console

The WebRTC Session Controller extends the WebLogic Administration Console with additional configuration and monitoring pages. The Administration Console interface for WebRTC Session Controller settings are similar to the core console available in Oracle WebLogic Server.

All WebRTC Session Controller configuration and monitoring is provided through these nodes in the left pane of the console:

- **SipServer**: presents SIP Servlet container properties and other engine tier functionality. This extension also enables you to view (but not modify) SIP data tier partitions and replicas.

- **Converged Load Balancer**: presents configuration settings and monitoring pages for the activities of the converged load balancers in the implementation.

See "Accessing the Administration Console" for more information about using the console.

WebLogic Scripting Tool

The WebLogic Scripting Tool enables you to perform interactive or automated (batch) configuration operations using a command-line interface. View and manipulate the MBeans available in a running WebRTC Session Controller domain using the WLST.

See "Using WLST (JMX) to Configure WebRTC Session Controller" for more information about modifying SIP Servlet container properties using WLST.

For general WLST information, including information about WLST commands, see *Oracle Fusion Middleware WebLogic Scripting Tool* documentation.

WebRTC Session Controller Console

You configure Signaling Engine and Media Engine parameters and entries in the WebRTC Session Controller console. Signaling Engine parameters include time limit parameters for SIP sessions and WebSocket connections. Media Engine entries represent media hosts that you use with WebRTC Session Controller.

See "Configuring WebRTC Session Controller Signaling Properties and Media Nodes" for more information on the WebRTC Session Controller console.

Additional Configuration Methods

Most WebRTC Session Controller configuration is performed using the interfaces above. The methods described in the following sections may also be used for certain configuration tasks.

Editing Configuration Files

You may also modify the configuration by editing configuration files.

The WebRTC Session Controller custom resources use the basic domain resources defined in **config.xml**, such as network channels, cluster and server configuration, and Java EE resources. The **config.xml** file applies to all managed servers in the domain. However, standalone WebRTC Session Controller components are configured in separate configuration files based on functionality:

- **sipserver.xml** contains general SIP container properties and engine tier configuration settings.
- **datatier.xml** identifies servers that participate as replicas in the SIP data tier, and also defines the number and layout of SIP data tier partitions.
- **diameter.xml** defines Diameter nodes and Diameter protocol applications used in the domain.

The component configuration files determine the role of each server instance, such as whether they behave as SIP data tier replicas or engine tier nodes.

See Part III, "Reference" for more information on the configuration files.

If you edit configuration files manually, you must restart all servers to apply the configuration changes.

Custom JMX Applications

You configure WebRTC Session Controller properties using JMX-compliant MBeans. You can program JMX applications for configuring SIP container properties using the appropriate WebRTC Session Controller MBeans.

See "Using WLST (JMX) to Configure WebRTC Session Controller" for the general procedure for modifying WebRTC Session Controller MBean properties using JMX. For more information about the individual MBeans used to manage SIP container properties, see *WebRTC Session Controller JavaScript API Reference*.

Common Configuration Tasks

General administration and maintenance of WebRTC Session Controller requires that you manage both WebLogic Server configuration properties and WebRTC Session Controller container properties.

Common configuration tasks include:

- Configure SIP Container Properties using the Administration Console or using WLST to perform batch configuration. See "Configuring WebRTC Session Controller Container Properties" for more information.
- Assign WebRTC Session Controller instances to the SIP data tier partitions and setting up call state replication using data tier instances. See "Configuring SIP Data Tier Partitions and Replicas" for more information.
- Configure WebLogic Server network channels to handle SIP and HTTP traffic. See "Configuring Network Connection Settings" for more information.
- Configure WebRTC Session Controller Signaling and Media Engine properties. See "Configuring WebRTC Session Controller" for more information.
- Create and deploy logging Servlets to record SIP requests and responses and manage log records. See "Logging SIP Requests and Responses" for more information.

2

Configuring WebRTC Session Controller Signaling Properties and Media Nodes

This chapter describes the Signaling Engine and Media Engine configurations you perform in the Oracle Communications WebRTC Session Controller web console.

About WebRTC Session Controller Console Configuration

You use the WebRTC Session Controller console for configuring Signaling Engine properties and Media Engine nodes. Additionally, you manage WebRTC applications, packages, and scripts in the console. See *WebRTC Session Controller Extension Developer's Guide* for more information on managing applications, packages, and scripts.

You can also configure WebRTC Session Controller console options using configuration Mbeans. See the **oracle.wsc.core.configuration.admin.mbean** package page for more information on using these MBeans in *Oracle Communications WebRTC Session Controller Configuration API Reference*.

About Signaling Engine Properties

You configure the Signaling Engine run-time parameters in the WebRTC Session Controller console. Table 2–1 describes the configurable Signaling Engine parameters.

Table 2–1 Configurable Signaling Engine Run-time Parameters

Parameter	Description
Glare Handling	Selecting glare handling enables the Signaling Engine to avoid race conditions that arise when a caller and callee send simultaneous invitations, re-invitations, or session updates. By default, glare handling is selected and enabled.
Sip Session Default Time	The default SIP session time (in seconds). The default value is 3600 seconds.
Sip Session Minimum Time	The minimum SIP session time (in seconds). The default value is 90 seconds.
WebSocket Disconnect Time Limit	The time interval after which the websocket times out (in milliseconds). The defalt value is 60000 milliseconds.
WebSocket Idle Time Limit	The idle time interval after which the websocket times out (in seconds). The default value is 30 seconds.
WebSocket Maximum Connections	The maximum number of websocket connections allowed. Set this value to **-1** for unlimited connections. The default value is -1.

About Media Engine Nodes Configuration and Status

You configure, remove, and manage Media Engine nodes in the WebRTC Session Controller console. Managing Media Engine nodes includes blocking and unblocking WebRTC network traffic to media nodes, monitoring their availability and ensuring the their load factor remains within acceptable limits.

Table 2–2 describes the configurable and viewable media node properties in the WebRTC Session Controller console.

Table 2–2 Media Node Properties

Property	Description
User	The user name required to connect to the Media Engine server.
Password	The password require to connect to the Media Engine server.
Address	The IP address of the Media Engine server.
Port	The port number of the media server connection.
Media Node Traffic Enabled	Whether traffic is enabled to the media node.
Media Node Status	Whether a connection to the media node is active.
Load Factor	The load percentage on a node controlled by the internal load balancer that attempts to distribute load evenly to available media nodes. WebRTC Session Controller will stop sending requests to media nodes with a Load Factor of 100%.

Accessing the WebRTC Session Controller Console Configuration Tab

The WebRTC Session Controller console resides on the same domain as your WebRTC Session Controller installation. When you start your domain, both the Oracle WebLogic Administration Console an the WebRTC Session Controller console become available.

You configure Signaling Engine and Media Engine parameters in the **Configuration** tab of the WebRTC Session Controller console. To access the WebRTC Session Controller console **Configuration** tab:

1. Start the WebRTC Session Controller domain server.
2. Open a web browser.
3. Access this URL:

   http://*host*:*port***/wsc-console**

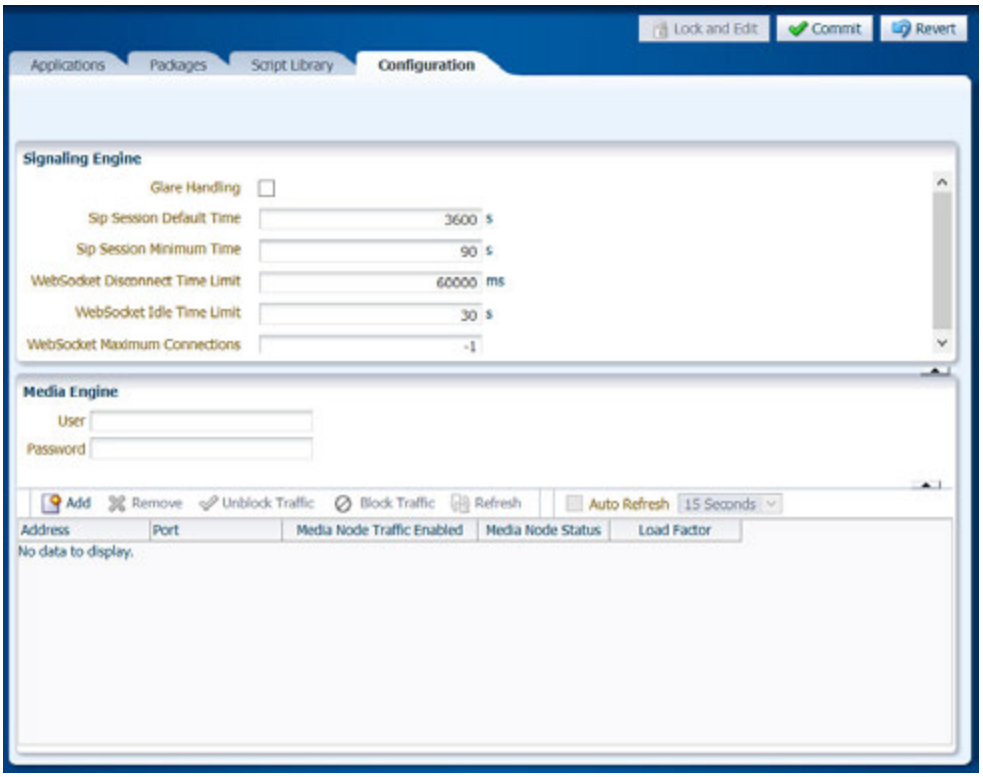
   where *host:port* represents the server and administration port number used by your domain server. If your environment uses HTTP security, use **https**.

   The WebLogic user login screen appears.
4. Enter the **Username** and **Password** you set when creating the WebLogic domain.
5. Click **Login**.

   The WebRTC Session Controller console window appears.
6. Select the **Configuration** tab.

   The **Signaling Engine** parameters and **Media Engine** node window are displayed as shown in Figure 2–1.

Figure 2–1 WebRTC Session Controller Console Configuration Tab



Configuring Signaling Engine Parameters

To configure Signaling Engine parameters:

1. Click **Lock and Edit** in the upper right corner of the screen.
2. Alter the Signaling Engine parameters listed in Table 2–1 as needed for your environment.
3. Click **Commit**.

Configuring Media Engine Nodes

You can perform the following media node configuration actions in the WebRTC Session Controller console in the **Media Engine** window:

- Adding Media Engine Nodes
- Removing Media Engine Nodes
- Blocking and Unblocking Media Node Traffic
- Setting Media Node Information Auto Refresh

Adding Media Engine Nodes

To add a Media Engine node:

1. Click **Lock and Edit** in the upper right corner of the screen.
2. Click **Add**.
3. Enter the **Address** and **Port** of the media server node.

4. Click **OK**.
5. Enter a **User** name and **Password**.
6. Click **Commit**.

Removing Media Engine Nodes

To remove a Media Engine node:

1. Click **Lock and Edit** in the upper right corner of the screen.
2. Select the row with the media node you want to remove.
3. Click **Block Traffic**.
4. Click **Remove**.

   The **Remove Media Node** window appears.
5. Click **OK**.
6. Click **Commit**.

Blocking and Unblocking Media Node Traffic

To block or unblock traffic to a media node:

1. Click **Lock and Edit** in the upper right corner of the screen.
2. Select the row with the media node you want to block or unblock traffic to.
3. Click **Unblock Traffic** or **Block Traffic**.
4. Click **Commit**.

Setting Media Node Information Auto Refresh

You can configure media node information to auto refresh at specified time intervals.

To enable media node data auto refresh and set the refresh interval:

1. Select the **Auto Refresh** check box in the **Media Engine** window.
2. Select the interval in the drop-down menu.

3

Using the Administration Console and WLST

This chapter describes managing Oracle Communications WebRTC Session Controller domain services using the Administration Console and WebLogic Scripting Tool (WLST).

Accessing the Administration Console

The Administration Console enables you to configure and monitor core Oracle WebLogic Server functionality and the SIP Servlet container functionality provided with WebRTC Session Controller.

See *Oracle WebLogic Server Administration Console Online Help* for more information about the Administration Console.

To configure or monitor SIP Servlet features using the Administration Console:

1. Ensure that your WebLogic Administration Server is running.
2. Use your browser to access the URL:

   `http://address:port/console`

   where *address* is the Administration Server's listen address and *port* is the listen port.
3. Select the **SipServer** node in the left pane.

   The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring WebRTC Session Controller. Table 3–1 summarizes the available pages and provides links to additional information about configuring SIP container properties.

Table 3–1 WebRTC Session Controller Configuration and Monitoring Pages

Tab	SubTab	Function
Configuration	General	Configure SIP timer values, session timeout duration, default WebRTC Session Controller behavior (proxy or user agent), server header format, call state caching, DNS name resolution, timer affinity, domain aliases, rport support, diagnostic image format, and stale session handling.
Configuration	Application Router	WebRTC Session Controller does not use this configuration tab.
Configuration	Proxy	Configure proxy routing URIs and proxy policies.
Configuration	Overload Protection	Configure the conditions for enabling and disabling automatic overload controls.
Configuration	Message Debug	Enable or disable SIP message logging on a development system.

Table 3–1 (Cont.) WebRTC Session Controller Configuration and Monitoring Pages

Tab	SubTab	Function
Configuration	SIP Security	Identify trusted hosts for which authentication is not performed.
Configuration	Persistence	Configure persistence options for storing long-lived session data in an RDBMS, or for replicating long-lived session data to a remote, geographically-redundant site.
Configuration	Data Tier	View the current configuration of SIP data tier servers. You can also add, delete and configure partitions here.
Configuration	LoadBalancer Map	Configure the mapping of multiple clusters to internal virtual IP addresses during a software upgrade.
Configuration	Targets	Configure the list of servers or clusters that receive the engine tier configuration. The target server list determines which servers and clusters provide SIP Servlet container functionality.
Configuration	Connection Pools	Configure connection reuse pools to minimize communication overhead with a Session Border Control (SBC) function or Serving Call Session Control Function (S-CSCF).
Monitoring	General	View run-time information about messages and sessions processed in engine tier servers.
Monitoring	SIP Performance	View run-time performance information on SIP traffic throughput and number of successful and failed transactions.
Monitoring	SIP Applications	View run-time session information for deployed SIP applications.
Monitoring	Data Tier Information	View run-time information about the current status and the work performed by servers in the SIP data tier.

Locking and Persisting the Configuration

The Administration Console Change Center provides a way to lock a domain configuration allowing configuration changes while preventing other administrators from making changes during your edit session. You can enable or disable this feature in development domains. It is disabled by default when you create a development domain.

See `"Enable and disable the domain configuration lock"` in the *Oracle WebLogic Server Administration Console Online Help* for more information on the domain configuration lock.

Some changes you make in the Administration Console take place immediately when you activate them. Other changes require you to restart the server or module affected by the change. These latter changes are called non-dynamic changes. Non-dynamic changes are indicated in the Administration Console with a warning icon containing an exclamation point. If an edit is made to a non-dynamic configuration setting, no edits to dynamic configuration settings will take effect until after you restart the server.

To make changes to your WebRTC Session Controller domain when domain configuration lock is enabled:

1. Locate the Change Center in the upper left corner of the Administration Console.
2. Click **Lock & Edit** to lock the editable configuration hierarchy for the domain.
3. Make the changes you want on the relevant page of the console and click **Save** on each page where you make a change.
4. When you have finished making all the changes, click **Activate Changes** in the Change Center.

Note:

- You can instead discard your current changes by clicking **Undo All Changes**. This deletes any temporary configuration files that were written with previous **Save** operations.
- If you need to discard all configuration changes made since the server was started, you can revert to original boot configuration file. See "Reverting to the Original Boot Configuration" for more information.

Using WLST (JMX) to Configure WebRTC Session Controller

The WebLogic Scripting Tool (WLST) is a utility that you can use to monitor or modify JMX MBeans available on a WebLogic Server or WebRTC Session Controller instance. You use WLST to configure both the WebRTC Session Controller SIP container and application. The following sections describe configuring WebRTC Session Controller with WLST:

- Configuring the SIP Container with WLST
- Configuring the WebRTC Session Controller Application with WLST

For more information on using the WLST, see `"Using the WebLogic Scripting Tool"` in the *Oracle WebLogic Scripting Tool* documentation.

Before using WLST to configure a WebRTC Session Controller domain, set your environment to add required WebRTC Session Controller classes to your classpath. Use either a domain environment script or the **setWLSEnv.sh** script located in *WL_home*/**server/bin** where *WL_home* is the directory where WebLogic Server is installed.

Configuring the SIP Container with WLST

This section provides information on configuring the WebRTC Session Controller SIP container using WLST.

Managing Configuration Locks

Table 3–2 summarizes the WLST methods used to lock a SIP container configuration and apply changes.

Table 3–2 SIP Container ConfigManagerRuntimeMBean Method Summary

Method	Description
`activate()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to the **sipserver.xml** configuration file and applies changes to the running servers.
`cancelEdit()`	Cancels an edit session, releasing the edit lock, and discarding all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.
`cd`	Navigate the hierarchy of configuration or run-time beans.
`connect`	Connect WLST to a WebLogic Server instance.
`edit()`	Starts an edit session.
`save()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to a temporary configuration file.

Table 3–2 (Cont.) SIP Container ConfigManagerRuntimeMBean Method Summary

Method	Description
`startEdit()`	Locks changes to the SIP Servlet container configuration. Other JMX applications cannot alter the configuration until you explicitly call `stopEdit()`, or until your edit session is terminated. If you attempt to call `startEdit()` when another user has obtained the lock, you receive an error message that states the user who owns the lock.
`set`	Set the specified attribute value for the current configuration bean.
`stopEdit()`	Releases the lock obtained for modifying SIP container properties and rolls back any pending MBean changes, discarding any temporary files.

A typical configuration session using WLST involves the following tasks:

1. Call **startEdit()** to obtain a lock on the active configuration.
2. Modify existing SIP Servlet container configuration MBean attributes (or create or delete configuration MBeans) to modify the active configuration. See "Configuration MBeans for the SIP Servlet Container" for a summary of the configuration MBeans.
3. Do one of the following:
   - Call save() to persist all changes to a temporary configuration file named sipserver.xml.saved
   - Call **activate()** to persist changes to the **sipserver.xml.saved** file, rename **sipserver.xml.saved** to **sipserver.xml** (copying over the existing file), and apply changes to the running engine tier server nodes.

Note: When you start the Administration Server for a WebRTC Session Controller domain, the server parses the current container configuration in **sipserver.xml** and creates a copy of the initial configuration in a file named **sipserver.xml.booted**. You can use this copy to revert to the booted configuration, as described in "Reverting to the Original Boot Configuration".

Configuration MBeans for the SIP Servlet Container

ConfigManagerRuntimeMBean manages access to and persists the configuration MBean attributes described in Table 3–3. Although you can modify other configuration MBeans, such as WebLogic Server MBeans that manage resources such as network channels and other server properties, those MBeans are not managed by ConfigManagerRuntimeMBean.

See "Configuring the WebRTC Session Controller Application with WLST" for information on MBeans used to configure WebRTC Session Controller application properties.

Table 3–3 SIP Container Configuration MBeans

MBean Type	MBean Attributes	Description
`ClusterToLoadBalancerMap`	`ClusterName`, `LoadBalancerSipURI`	Manages the mapping of multiple clusters to internal virtual IP addresses during a software upgrade. This attribute is not used during normal operations. See also "Define the Cluster-to-Load Balancer Mapping".
`OverloadProtection`	`RegulationPolicy`, `ThresholdValue`, `ReleaseValue`	Manages overload settings for throttling incoming SIP requests. See also "overload".
`Proxy`	`ProxyURIs`, `RoutingPolicy`	Manages the URIs routing policies for proxy servers. See also "proxy—Setting Up an Outbound Proxy Server".
`SipSecurity`	`TrustedAuthenticationHosts`	Defines trusted hosts for which authentication is not performed. See also "sip-security".

`SipServer`	`DefaultBehavior`, `EnableLocalDispatch`, `MaxApplicationSessionLifeTime`, `OverloadProtectionMBean`, `ProxyMBean`, `T1TimeoutInterval`, `T2TimeoutInterval`, `T4TimeoutInterval`, `TimerBTimeoutInterval`, `TimerFTimeoutInterval`
	
	SipServer also has several helper methods:
	`createProxy()`, `destroyProxy()`, `createOverloadProtection()`, `destroyOverloadProtection()`, `createClusterToLoadBalancerMap()` `destroyClusterToLoadBalancerMap()`
		Configuration MBean that represents the entire **sipserver.xml** configuration file. You can use this MBean to obtain and manage each of the individual MBeans described in this table, or to set SIP timer or SIP Session timeout values. See also:
		■ Creating and Deleting MBeans
		■ default-behavior,
		■ enable-local-dispatch
		■ max-application-session-lifetime
		■ t1-timeout-interval
		■ t2-timeout-interval
		■ t4-timeout-interval
		■ timer-b-timeout-interval
		■ timer-f-timeout-interval

Locating the SIP Container MBeans

All SIP Servlet container configuration MBeans are located in the **serverConfig** MBean tree, accessed using the **serverConfig()** command in WLST. Within this bean tree, individual configuration MBeans can be accessed using the path:

```
CustomResources/sipserver/Resource/sipserver
```

For example, to browse the default Proxy MBean for a WebRTC Session Controller domain you would enter these WLST commands:

```
serverConfig()
cd('CustomResources/sipserver/Resource/sipserver/Proxy')
ls()
```

Run-time MBeans, such as **ConfigManagerRuntime**, are accessed in the **custom** MBean tree, accessed using the **custom()** command in WLST. Run-time MBeans use the path:

```
mydomain:Location=myserver,Name=myserver,Type=mbeantype
```

Certain configuration settings, such as proxy and overload protection settings, are defined by default in **sipserver.xml**. Starting an associated server generates Configuration MBeans for these settings. You can immediately browse the **Proxy** and **OverloadProtection** MBeans. Other configuration settings are not configured by default and you will need to create the associated MBeans before they can be accessed. See "Creating and Deleting MBeans" for more information.

Configuring the WebRTC Session Controller Application with WLST

This section provides information on configuring the WebRTC Session Controller application using WLST.

Managing Configuration Locks

Table 3–4 summarizes the WebRTC Session Controller methods included in **ConfigAdminMBean**.

Table 3–4 WebRTC Session Controller ConfigAdminMBean Method Summary

Method	Description
`lockAndEdit()`	Begins the configuration update transaction.
`isLocked()`	Checks if the configuration is locked (by any user).
`getCurrentLock()`	Gets current lock if one exist. The request fails if the lock is owned by another user.
`revert()`	Reverts a configuration update transaction.
`commit()`	Commits a configuration update transaction.
`validate()`	Validates the transaction.
`validateAllScripts()`	Validates all the scripts.
`validateScript()`	Validates a particular script.
`validateScriptLibrary()`	Validates the script libraries.
`validateMediaEngines()`	Validates all the Media Engines and checks to see if they are reachable from the Signaling Engine.

Configuration MBeans for WebRTC Session Controller

Table 3–5 lists the configuration MBeans for WebRTC Session Controller. See the **oracle.wsc.core.configuration.admin.mbean** package in *Oracle Communications WebRTC Session Controller Configuration API Reference* for detailed information about each MBean.

See "Accessing WebRTC Session Controller Application MBeans" for information on how to access and use the WebRTC Session Controller MBeans.

Table 3–5 WebRTC Session Controller Configuration MBeans

MBean	Description
`ApplicationMBean`	WebRTC Session Controller application configuration MBean.
`AscMBean`	Media Engine configuration MBean.

Table 3–5 (Cont.) WebRTC Session Controller Configuration MBeans

MBean	Description
AscMBeans	Media Engine configuration MBean.
ConfigAdminMBean	WebRTC Session Controller configuration administration MBean.
PackageMBean	WebRTC Session Controller package configuration MBean.
ResourceLimitsProfileMBean	WebRTC Session Controller resource limits configuration MBean.
ScriptLibraryMBean	WebRTC Session Controller script library configuration MBean.
ScriptMBean	WebRTC Session Controller script configuration MBean.
SystemConfigurationsMBean	WebRTC Session Controller application configuration MBean.
WebSocketMBean	WebRTC Session Controller WebSocket configuration MBean.
WscConfigMBean	WebRTC Session Controller main configuration MBean.

Accessing WebRTC Session Controller Application MBeans

You configure the WebRTC Session Controller MBeans using the Java **MbeanServerConnection** interface. Use the **mbs** variable at the WLST interface prompt to access the MBeans.

See the "WLST Variable Reference" in *WLST Command Reference for WebLogic Server* for information about the **mbs** variable.

To configure the WebRTC Session Controller MBeans using **mbs**:

1. Connect to the WebLogic instance using WLST.
2. Use the **MBeanServerConnection** to interact with the WebRTC Session Controller MBean server. See the following link for more information, including available methods, about **MBeanServerConnection**:

   http://docs.oracle.com/javase/7/docs/api/javax/management/MBeanServerConnection.html
3. Access the WebRTC Session Controller administration MBean, which is the root of all WebRTC Session Controller MBeans, using the following object name:

   ```
   oracle.wsc:Location=AdminServer,Type=ConfigAdminMBean
   ```
4. Use the **getAttribute**, **setAttribute**, and **invoke** operations to interact with the MBeans and configure the WebRTC Session Controller.

See "WebRTC Session Controller Code Sample" for an example showing how to use the **MBeanServerConnection** method to perform common configuration tasks.

WLST Configuration Examples

The following sections provide example WLST scripts and commands for configuring SIP Servlet container properties.

Invoking WLST

To use WLST with WebRTC Session Controller, you must ensure that all WebRTC Session Controller JAR files are included in your classpath. Follow these steps:

1. Set your WebRTC Session Controller environment:

   ```
   cd ~/domain_home/bin
   ./setDomainEnv.sh
   ```

   where *domain_home* is the path to the domain's home directory.

2. Start WLST:

   ```
   java weblogic.WLST
   ```

3. Connect to the Administration Server for your WebRTC Session Controller domain:

   ```
   connect('system','weblogic','t3://myadminserver:port_number')
   ```

WLST Template for Configuring Container Attributes

Because a typical configuration session involves accessing **ConfigManagerRuntimeMBean** twice—once for obtaining a lock on the configuration, and once for persisting or applying changes—JMX applications that manage container attributes generally have a similar structure.

Example 3–1 shows a WLST script that contains the common commands needed to access **ConfigManagerRuntimeMBean**. The example script modifies the proxy **RoutingPolicy** attribute, which is set to **supplemental** by default in new WebRTC Session Controller domains. You can use this listing as a basic template, modifying commands to access and modify the configuration MBeans as necessary.

Example 3–1 Template WLST Script for Accessing ConfigManagerRuntimeMBean

```
# Connect to the Administration Server
connect('username','password','t3://localhost:7001')
# Start an edit session
edit()
startEdit()
# --MODIFY THIS SECTION AS NECESSARY--
# Edit SIP Servlet container configuration MBeans
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=myserver,SipServer=myser
ver,Type=Proxy')
set('RoutingPolicy','domain')
# Commit changes
save()
activate()
```

Creating and Deleting MBeans

The **SipServer** MBean represents the entire contents of the **sipserver.xml** configuration file. In addition to having several attributes for configuring SIP timers and SIP application session timeouts, SipServer provides helper methods to help you create or delete MBeans representing proxy settings and overload protection controls.

Example 3–2 shows an example of how to use the helper commands to create and delete configuration MBeans that configuration elements in **sipserver.xml**. See also *WebRTC Session Controller JavaScript API Reference* for more information.

Example 3–2 WLST Commands for Creating and Deleting MBeans

```
connect('username','password','t3://localhost:7001')
edit()
startEdit()
```

```
cd('CustomResources/sipserver/Resource/sipserver')
cmo.destroyOverload()
cmo.createProxy()
save()
activate()
```

WebRTC Session Controller Code Sample

Oracle recommends using **MBeanServerConnection** (**mbs**) methods when using WLST to perform WebRTC Session Controller configuration instead of the built-in WLST operations. Example 3–3 provides sample code including how to connect to an administration server, lock configuration, retrieve and modify attributes, create test packages, and commit configurations using the **mbs** variable.

See "Accessing WebRTC Session Controller Application MBeans" for more information on using MBeanServerConnection.

Example 3–3 Connecting and Performing MBean Operations with mbs

```
# Connect to Admin Server
connect('username', 'password', 't3://127.0.0.1:7001')

# Lock configuration
noObjs = jarray.array([],java.lang.Object)
noStrs = jarray.array([],java.lang.String)
admin = ObjectName('oracle.wsc:Location=AdminServer,Type=ConfigAdminMBean')
myLock = mbs.invoke(admin, 'lockAndEdit', noObjs, noStrs)

# Get some attribute
mbs.getAttribute(myLock, 'Packages')

# Change some attributes
myApp=ObjectName('oracle.wsc:Type=ApplicationMBean,Location=AdminServer,Name=unsec
ure,User=weblogic')
activeAttr=Attribute('Active', Boolean('false'))
mbs.setAttribute(myApp, activeAttr)
descAttr=Attribute('Description', 'Disabled this app')
mbs.setAttribute(myApp, descAttr)

# Create test package
packageObjs = jarray.array(['test-package'], java.lang.Object)
packageStrs = jarray.array(['java.lang.String'], java.lang.String)
myPackage = mbs.invoke(myLock, 'createPackage', packageObjs, packageStrs)

# Commit configuration
commitObjs = jarray.array([myLock], java.lang.Object)
commitStrs = jarray.array(['javax.management.ObjectName'], java.lang.String)
mbs.invoke(admin, 'commit', commitObjs, commitStrs)
```

Setting Logging Levels

The WebRTC Session Controller is subject to the common configuration settings defined for WebLogic servers. To modify the logging settings for a WebRTC Session Controller in the Administration Console, access the logging configuration settings page as follows:

1. Expand the **Environment** node in the Domain Structure tree.
2. Click **Servers**.

3. Click the name of the server you want to configure logging for in the **Configuration** tab.
4. In the right pane, click the **Logging** tab.
5. Modify the default logging settings and then click **Save** to commit your changes.

Alternatively, use the **logging.xml** WebLogic file to manually configure logging properties for the servers.

WebRTC Session Controller supports additional logging features that provide for SIP message logging. SIP message logging should be enabled in development environments only. It is not intended for production environments.

To configure SIP message logging:

1. Expand the **SipServer** node in the Domain Structure tree.
2. In the **Configuration** tab, click the **Message Debug** subtab.
3. Select the **Enable Debug** check box.
4. Configure other message logging settings as needed. Other settings include the logging verbosity level, the log entry pattern, and the target log file name. See the on-screen field description for more information.
5. Click **Save** to commit your changes.
6. Restart the WebLogic server.

See "Logging SIP Requests and Responses" for information about creating custom log listeners and more information about logging settings.

Startup Sequence for a WebRTC Session Controller Domain

WebRTC Session Controller start scripts use default values for many JVM parameters that affect performance. For example, JVM garbage collection and heap size parameters may be omitted, or may use values that are acceptable only for evaluation or development purposes.

In a production system, you must rigorously profile your applications with different heap size and garbage collection settings to realize adequate performance. See "Modifying JVM Parameters in Server Start Scripts" in the chapter for suggestions about maximizing JVM performance in a production domain.

Caution: When you configure a domain with multiple Signaling Engine servers, you must accurately synchronize all system clocks to a common time source (to within one or two milliseconds) in order for the SIP protocol stack to function properly. See "Configuring NTP for Accurate SIP Timers" for more information.

Because a typical WebRTC Session Controller domain contains numerous Signaling Engine and SIP data tier servers, with dependencies between the different server types, you should generally follow this sequence when starting up a domain:

1. Start the Administration Server for the domain.

   Start the Administration Server to provide the initial configuration to engine servers in the domain. The Administration Server can also be used to monitor the startup/shutdown status of each Managed Server.

You generally start the Administration Server by using the **startWebLogic.sh** or **startWebLogic.cmd** script (depending on your operating system) installed with the Configuration Wizard, or a custom startup script.

2. Start SIP data tier servers in each partition.

   The engine tier cannot function until servers in the SIP data tier are available to manage call state data. Although all replicas in each partition need not be available to begin processing requests, at least one replica in each configured partition must be available to manage the concurrent call state. All replicas should be started and available before opening the system to production network traffic.

   You generally start each SIP data tier server by using either the **startManagedWebLogic.cmd** script installed with the Configuration Wizard, or a custom startup script. **startManagedWebLogic.cmd** requires that you specify the name of the server to startup and the URL of the Administration Server for the domain. For example:

   ```
   startManagedWebLogic.cmd datanode0-0 t3://adminhost:7001
   ```

3. Start engine tier servers.

   Start Signaling Engine servers and begin processing client requests. Signaling engine tier servers are generally started using the **startManagedWebLogic.cmd** script or a custom startup script.

Following the above startup sequence ensures that all Managed Servers use the latest SIP Servlet container and SIP data tier configuration. This sequence also avoids engine tier error messages that are generated when servers in the SUP data tier are unavailable.

Startup Command Options

Table 3–6 lists startup options available to WebRTC Session Controller. For more information about these and other options, see "weblogic.Server Command-Line Reference" in the *Command Reference for Oracle Weblogic Server* documentation.

Table 3–6 Startup Command Options

Application	Startup Option	For More Information
WebRTC Session Controller	-Dwlss.udp.listen.on.ephemeral	See information about single network adapter card configurations with TCP and UDP channels in *Oracle WebLogic Server SIP Container Administrator's Guide*.
WebRTC Session Controller	-Dwlss.udp.lb.masquerade	See information about single network adapter card configurations with TCP and UDP channels in *Oracle WebLogic Server SIP Container Administrator's Guide*.
WebRTC Session Controller	-Dweblogic.management.discover	See "Restarting an Administration Server on the Same System" for more information.
WebRTC Session Controller	-Dweblogic.RootDirectory	See "Restarting an Administration Server on Another System" for more information.

Table 3–6 (Cont.) Startup Command Options

Application	Startup Option	For More Information
Installer	-Djava.io.tmpdir	See the discussion on temporary disk space requirements in *Installation Guide for Oracle WebLogic Server* in the WebLogic Server documentation.
WlssEchoServer	-Dwlss.ha.echoserver.port	See "Enabling and Configuring the Heartbeat Mechanism on Servers" for more information.
WlssEchoServer	-Dwlss.ha.echoserver.logfile	See "Enabling and Configuring the Heartbeat Mechanism on Servers" for more information.
WlssEchoServer	-Dreplica.host.monitor.enabled	See "Enabling and Configuring the Heartbeat Mechanism on Servers" for more information.
WlssEchoServer	-Dwlss.ha.heartbeat.interval	See "Enabling and Configuring the Heartbeat Mechanism on Servers" for more information.
WlssEchoServer	-Dwlss.ha.heartbeat.count	See "Enabling and Configuring the Heartbeat Mechanism on Servers" for more information.
WlssEchoServer	-Dwlss.ha.heartbeat.SoTimeout	See "Enabling and Configuring the Heartbeat Mechanism on Servers" for more information.

Reverting to the Original Boot Configuration

When you start the Administration Server for a WebRTC Session Controller domain, the server creates parses the current container configuration in **sipserver.xml**, and generates a copy of the initial configuration in a file named **sipserver.xml.booted** in the **config/custom** subdirectory of the domain directory. This backup copy of the initial configuration is preserved until you next start the server; modifying the configuration using JMX does not affect the backup copy.

If you modify the SIP Servlet container configuration and later decide to roll back the changes, copy the **sipserver.xml.booted** file over the current **sipserver.xml** file. Then restart the server to apply the new configuration.

4

Configuring WebRTC Session Controller Authentication

This chapter describes WebRTC Session Controller authentication schemes and the steps to configure them.

About WebRTC Session Controller Security Schemes

Before WebRTC Session Controller can process any signaling traffic, you must configure an authentication scheme.

WebRTC Session Controller provides out of the box support for these authentication schemes:

- Guest authentication

  This scheme allows anonymous guest access to WebRTC Session Controller.

- HTTP authentication

  This provider sends a HTTP GET request to a remote HTTP endpoint (for instance, a Representational State Transformation (REST) endpoint) using HTTP BASIC authentication headers. A return code of 200 indications that authentication was successful.

- OAuth 2.0 authentication

  This authentication scheme lets you leverage OAuth 2.0 authentication support provided by companies such as Facebook or Google, and lets WebRTC Session Controller retrieve user information such as an email address, with the consent of that user.

The following sections describe the configuration steps for each of those authentication schemes.

About Provisioning WebRTC Session Controller Guest Access

To provision guest access for WebRTC Session Controller, you must configure settings in the WebLogic Administration Console and then define a new WebRTC Session Controller application in the WebRTC Session Controller console.

Configuring the WebLogic Server Guest Access Provider

To configure the WebLogic Server guest access provider:

1. Start your Signaling Engine servers if they are not already running. See *Oracle Communications WebRTC Session Controller Installation Guide* for more information.

2. Navigate to the WebLogic Server Administration Console and log in with your administrator user name and password:

   `http://host:port/console`

   where *host* is the name of your WebRTC Session Controller server and *port* is the Administration Console access port.

   Note: The default Administration Console port is 7001.

3. In the Domain Structure pane, select **Security Realms**.
4. Click **myrealm** in the Realms table.
5. Click the **Providers** tab and then click **New**.
6. Enter a name in the Name text box, in the Type drop down list, select **WscServletAuthenticator**, and click **OK**.
7. Click the newly created authentication provider in the list of Authentication Providers, and click the **Provider Specific** tab.
8. Make a note of the Guest Uri Match Pattern. The default is **/ws/webrtc/guest**.
9. Navigate back to the **myrealm** Providers tab, and in the list of Authentication Providers, click **DefaultAuthenticator**.
10. Select the **Common** tab, choose **OPTIONAL** in the Control Flag drop down list, and click **Save**.
11. Log out of the WebLogic administration interface.

Continue to "Configuring the WebRTC Session Controller Guest Access Application".

Configuring the WebRTC Session Controller Guest Access Application

For more details on WebRTC Session Controller application configuration options, see the discussion on creating applications in *Oracle Communications WebRTC Session Controller Extension Developer's Guide*.

To configure the WebRTC Session Controller guest access application:

1. Navigate to the WebRTC Session Controller console and log in with your administrator user name and password:

   `http://host:port/wsc-console`

   where *host* is the name of your WebRTC Session Controller server and *port* is the Administration Console access port.

   Note: The default Signaling Engine console port is 7001.

2. Select the **Applications** tab.
3. Click **Lock and Edit**.
4. Click **Create**.
5. Enter a name for the application in Create Application and click **OK**.
6. In the Request URI text box, enter the URI that you noted in "Configuring the WebLogic Server Guest Access Provider". The default value is **/ws/webrtc/guest**.

7. Enter **guest** for the Security Group.
8. Enter * for Allowed Domains, or customize as your deployment requires.
9. Choose **call**, **message_notification**, and **register** for the Packages.
10. Click **Commit**.
11. Restart WebRTC Session Controller.

About Provisioning WebRTC Session Controller HTTP Access

To provision HTTP access for WebRTC Session Controller, you must configure settings in the WebLogic Administration Console and then define a new WebRTC Session Controller application in the WebRTC Session Controller console.

In addition you must have your own HTTP endpoint defined to handle authentication requests.

Configuring the WebLogic Server HTTP Authentication Provider

To configure the WebLogic Server HTTP access provider:

1. Start your Signaling Engine servers if they are not already running. See *Oracle Communications WebRTC Session Controller Installation Guide* for more information.
2. Navigate to the WebLogic Server Administration Console and log in with your administrator user name and password:

   `http://host:port/console`

   where *host* is the name of your WebRTC Session Controller server and *port* is the Administration Console access port.

   Note: The default Administration Console port is 7001.

3. In the Domain Structure pane, select **Security Realms**.
4. Click **myrealm** in the Realms table.
5. Click the **Providers** tab and then click **New**.
6. Enter a name in the Name text box, in the Type drop down list, select **WscRestAuthenticator**, and click **OK**.
7. Click the newly created authentication provider in the list of Authentication Providers, and click the **Provider Specific** tab.
8. Enter a Group name to associate a group with authentication requests rather than individual user names. Make a note of this group name.
9. To enable authentication over http, check **Allow Http**. By default, only https is allowed.
10. Enter your REST endpoint in the Rest End Point Uri text box and click **Save**.
11. Log out of the WebLogic administration interface.

Continue to "Configuring the WebRTC Session Controller HTTP Access Application".

Configuring the WebRTC Session Controller HTTP Access Application

For more details on WebRTC Session Controller application configuration options, see the discussion on creating applications in *Oracle Communications WebRTC Session Controller Extension Developer's Guide*.

To configure the WebRTC Session Controller HTTP access application:

1. Navigate to the WebRTC Session Controller console and log in with your administrator user name and password:

   `http://host:port/wsc-console`

   where *host* is the name of your WebRTC Session Controller server and *port* is the Administration Console access port.

   Note: The default Signaling Engine console port is 7001.

2. Select the **Applications** tab.
3. Click **Lock and Edit**.
4. Click **Create**.
5. In **Create Application**, enter a name for the application.
6. Click **OK**.
7. In the **Request URI** text box, enter the URI endpoint that you want WebRTC applications to use to access WebRTC Session Controller.
8. Enter the group name you defined in "Configuring the WebLogic Server HTTP Authentication Provider" for the Security Group.
9. Click the pencil icon under **Allowed Domains**.
10. In the **Allowed Domains** window, enter * to allow all domains, or customize as your deployment requires.
11. Click **OK**.
12. Click the pencil icon under **Packages**.
13. In the **Packages** window, select the **call**, **message_notification**, and **register** packages and move them to **Selected Packages**.
14. Click **OK**.
15. Click **Commit**.

About Provisioning WebRTC Session Controller OAuth Access

To provision OAuth access for WebRTC Session Controller, you must configure settings in the WebLogic Administration Console and then define a new WebRTC Session Controller application in the WebRTC Session Controller console.

In addition you must procure a developer's account from the provider from whom you want to leverage OAuth authentication services and obtain the following information:

- The OAuth service provider's OAuth user information URI
- An OAuth client ID supplied to you by the OAuth service provider
- The service provider's OAuth server URI

- Your OAuth client secret, defined when you create your account with your OAuth service provider

Configuring the WebLogic Server OAuth Access Provider

To configure the WebLogic Server REST access provider:

1. Start your Signaling Engine servers if they are not already running. See *Oracle Communications WebRTC Session Controller Installation Guide* for more information.
2. Navigate to the WebLogic Server Administration Console and log in with your administrator user name and password:

   `http://host:port/console`

   where *host* is the name of your WebRTC Session Controller server and *port* is the Administration Console access port.

   Note: The default Administration Console port is 7001.

3. In the Domain Structure pane, select **Security Realms**.
4. Click **myrealm** in the **Realms** table.
5. Click the **Providers** tab and then click **New**.
6. Enter a name in the Name text box, in the Type drop down list, select **WscServletAuthenticator**, and click **OK**.

   The console creates the new provider and returns to the **Authentication Providers** table.

   Note: The WscServletAuthenticator must be deployed to enable OAuth security authentication, but it does not need to be further configured.

7. Click **New**.
8. Enter a name in the Name text box, in the Type drop down list, select **WscOAuthIdentityAsserter**, and click **OK**.
9. Click the newly created authentication provider in the list of Authentication Providers.
10. Assign an access token to the provider in Active Types and click **Save**.

    If you are provisioning multiple OAuth authentication sources, for example, Facebook, Google, and Microsoft, you should select a different OAuth token for each in the Active Types list.

    WARNING: The user interface will let you select multiple OAuth tokens for a single provider. Only select a single token for each OAuth provider you provision.

11. Select the Provider Specific tab and enter the following information as described in Table 4–1.

Table 4–1 OAuth Provider Specific Attributes

Attribute Name	Attribute Description
Group Name	Required. A group name used to associate a group with authentication requests. Specifying a group name allows both the user name and group name to be available in the authenticated subject. Make a note of this group name.
OAuth User Info Uri	Required. The OAuth providers URI that provides user information.
Proxy Port	Optional. The proxy port used to connect to the OAuth server.
OAuth Client ID	Required. The OAuth client ID provided to you by your OAuth service provider.
OAuth Server Uri	Required. The URI of your OAuth service provider's OAuth server which issues access tokens.
OAuth Redirect Uri	Optional. The URI to which the browser is re-directed after successful authentication by the OAuth provider.
OAuth Client Secret	Required. The OAuth client secret provided to you by your OAuth provider.
Proxy Server	Optional. The proxy URI used to connect to the OAuth server.

12. Click **Save**.
13. Log out of the WebLogic administration interface.

Continue to "Configuring the WebRTC Session Controller OAuth Access Application".

Configuring the WebRTC Session Controller OAuth Access Application

For more details on WebRTC Session Controller application configuration options, see the discussion on creating applications in *Oracle Communications WebRTC Session Controller Extension Developer's Guide*.

To configure the WebRTC Session Controller OAuth access application:

1. Navigate to the WebRTC Session Controller console and log in with your administrator user name and password:

   `http://host:port/wsc-console`

   where *host* is the name of your WebRTC Session Controller server and *port* is the Administration Console access port.

   Note: The default Signaling Engine console port is 7001.

2. Select the **Applications** tab.
3. Click **Lock and Edit**.
4. Click **Create**.
5. Enter a name for the application in **Create Application** and click **OK**.
6. In the **Description** text box, enter a description for your applicaiton.

7. In the **Request URI** text box, enter the URI endpoint that you want WebRTC applications to use to access WebRTC Session Controller.
8. In the **Security Group** text box, enter the group name you defined in "Configuring the WebLogic Server OAuth Access Provider".
9. Click the pencil icon under **Allowed Domains**.
10. In the **Allowed Domains** window, enter * to allow all domains, or customize as your deployment requires.
11. Click **OK**.
12. Click the pencil icon under **Packages**.
13. In the **Packages** window, select the **call**, **message_notification**, and **register** packages and move them to **Selected Packages**.
14. Click **OK**.
15. Click **Commit**.
16. Restart WebRTC Session Controller.

5

Configuring WebRTC Session Controller Diameter Rx to PCRF Integration

This chapter describes how to integrate Oracle Communications WebRTC Session Controller with a Diameter Rx Policy Control and Charging Rules Function (PCRF) server.

About the WebRTC Session Controller Rx Interface

You can use WebRTC Session Controller to enforce media and Quality of Service (QoS) policies by integrating with a PCRF using the Diameter Rx interface. The Diameter Rx interface includes session information and access charging identifiers that both your PCRF and WebRTC Session Controller implementation can use to enforce QoS limits.

See the chapter on using policy data in messages and the appendix section on Diameter Rx Protocol support in *WebRTC Session Controller Extension Developer's Guide* for more information on supported commands, requests and answers.

Overview of Diameter Rx Protocol Configuration

WebRTC Session Controller domain includes support for the Diameter base protocol and the IMS Diameter Rx interface deployed to engine tier servers that act as Diameter client nodes. SIP Servlets deployed on the engines can use the available Diameter application to initiate requests for PCRF functions.

Installing the Diameter Domain Template

You enable Diameter Rx functionality by extending an existing WebRTC Session Controller domain with the appropriate WebRTC Session Controller Diameter domain template JAR file located in:

*Middleware_Home***/wlserver/common/templates/wls** directory

where *Middleware_Home* is the directory where you installed WebRTC Session Controller.

Domain template files are provided for both basic domain and replicated domain configurations. Use the **wsc_diameter_basicdomain.jar** when updating basic domains and the **wsc_diameter_replicateddomain.jar** when updating replicated domains.

To upgrade an existing domain with the Diameter Domain template:

1. Log on to the host where you installed WebRTC Session Controller.

2. Navigate to the *Middleware_Home*/**common/bin** directory where *Middleware_Home* is the location where you installed WebRTC Session Controller.
3. Start the Fusion Middleware Configuration Wizard with `./config.sh`.
4. On the **Configuration Type** wizard screen, select **Update an existing domain**.
5. In the **Domain Location**, enter the path to the domain directory of the domain you are updating. Alternatively, click **Browse** to browse to and select the location.
6. Click **Next**.
7. In the **Templates** wizard screen, select **Update Domain Using Custom Template**.
8. Click **Browse**.
9. Browse to and select the *Middleware_Home*/**wlserver/common/templates/wls** directory.
10. Click **Open**.
11. Select either the **wsc_diameter_basicdomain.jar** or **wsc_diameter_replicateddomain.jar** template file corresponding to your domain.
12. Click **OK**.
13. Click **Next**.
14. Adjust any properties in the **Advance Configuration** wizard screen if needed.
15. Click **Next**.
16. In the **Configuration Summary** wizard screen click **Update**.
17. Click **Next** when the update is done.
18. Click **Finish** to exit the wizard.

Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol

The WebRTC Session Controller Diameter implementation supports the Diameter protocol over the TCP, TLS, and SCTP transport protocols. (SCTP transport is provided with certain restrictions as described in "Configuring and Using SCTP for Diameter Messaging".)

To enable incoming Diameter connections on a server, you must configure a dedicated network channel of the appropriate protocol type:

- **diameter** channels use TCP transport
- **diameters** channels use TCP/TLS transport
- **diameter-sctp** channels use TCP/SCTP transport.

Servers that use a TCP/TLS channel for Diameter (diameters channels) must also enable two-way SSL. WebRTC Session Controller may automatically upgrade Diameter TCP connections to use TLS as described in the Diameter specification (RFC 3558).

To configure a TCP or TCP/TLS channel for use with the Diameter provider:

1. Access the Administration Console for the WebRTC Session Controller domain.
2. Click **Lock & Edit** to obtain a configuration lock.

If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See "Enable and disable the domain configuration lock" in the *Administration Console Online Help* for more information.

3. In the **Domain Structure** tree, expand **Environment**.
4. Click **Servers**.
5. In the **Servers** table, select the server to configure.
6. Select the **Protocols** tab, and then select the **Channels** subtab to display the configured channels.
7. Click **New** to configure a new channel.
8. Fill in the fields of the **Identity Properties** page as follows:
   - **Name:** Enter an administrative name for this channel, such as "Diameter TCP/TLS Channel."
   - **Protocol:** Select **diameter** to support the TCP transport, **diameters** to support both TCP and TLS transports, or **diameter-sctp** to support TCP transport.

     Note: If a server configures at least one TLS channel, the server operates in TLS mode and will reject peer connections from nodes that do not support TLS (as indicated in their capabilities exchange).

9. Click **Next** to continue.
10. Fill in the fields of the **Network Channel Addressing** page as follows:
    - **Listen Address:** Enter the IP address or DNS name for this channel. On a multi-homed system, enter the exact IP address of the interface you want to configure, or a DNS name that maps to the exact IP address.
    - **Listen Port:** Enter the port number used to communication through this channel. Diameter nodes conventionally use port 3868 for incoming connections.
    - **External Listen Address**: The external IP address or DNS name for this channel.
    - **External Listen Port**: Re-enter the Listen Port value.
11. Click **Next** to continue.
12. Chose attributes in the **Network Channel Properties** page as follows:
    - **Enabled**: Select this attribute to ensure that the new channel accepts network traffic.
    - **Tunneling Enabled**: Un-check this attribute for Diameter channels.
    - **HTTP Enabled for this Protocol**: Un-check this attribute for Diameter channels.
    - **Outbound Enabled**: Select this attribute to ensure that the node can initiate Diameter messages using the channel.
13. Click **Next** to continue.
14. For **diameters** channels, select the following two attributes:
    - **Two Way SSL Enabled**: Two-way SSL is required for TLS transport.

- **Client Certificate Enforced**: Select this attribute to honor available client certificates for secure communication.

15. Click **Finish** to create the new channel.
16. Select the name of the newly-created channel in the **Network Channels** table.
17. Display the advanced configuration items for the newly-created channel by expanding the **Advanced** link.
18. Change the **Idle Connection Timeout** value from the default (65 seconds) to a larger value that will ensure the Diameter connection remains consistently available.

    Note: If you do not change the default value, the Diameter connection will be dropped and recreated every 65 seconds with idle traffic.

19. Click **Save**.
20. Click **Activate Changes**.

Configuring Two-Way SSL for Diameter TLS Channels

Diameter channels that use TLS (diameters channels) require that you also enable two-way SSL, which is disabled by default. If you have not already configured Two-Way SSL, see `"Configuring SSL"` in *Administering Security for Oracle WebLogic Server* for more information.

Configuring and Using SCTP for Diameter Messaging

SCTP is a reliable, message-based transport protocol that is designed for use in telephony networks. SCTP provides several benefits over TCP:

- SCTP preserves the internal structure of messages when transmitting data to an endpoint, whereas TCP transmits raw bytes that must be received in order.
- SCTP supports multihoming, where each endpoint may have multiple IP addresses. The SCTP protocol can transparently failover to another IP address should a connection fail.
- SCTP provides multistreaming capabilities, where multiple streams in a connection transmit data independently of one another.

WebRTC Session Controller supports SCTP for Diameter network traffic, with several limitations:

- Only 1 stream per connection is currently supported.
- Use SCTP only for Diameter network traffic; SIP traffic cannot use a configured SCTP channel.
- TLS is not supported over SCTP.

SCTP channels can operate on either IPv4 or IPv6 networks. "Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol" describes how to create a SCTP channel. To enable multihoming capabilities for an existing SCTP channel, specify the IPv4 address **0.0.0.0** as the listen address for the channel (or use the **::** address for IPv6 networks).

Configuring Diameter Nodes

The Diameter node configuration for WebRTC Session Controller engines is specified in the **diameter.xml** configuration file, which is located in the directory: *Middleware_Home*/**user_projects/domains/***domain_name*/**config/custom**

Where *Middleware_Home* is the directory in which the WebRTC Session Controller software is installed (the installation program used to install WebRTC Session Controller refers to this as Middleware Home), and *domain_name* is the name of the Diameter domain.

To provide diameter services on an engine tier server, you must create a Diamter node configuration and target the configuration to an existing engine server instance.

Diameter node configurations are divided into several categories:

- General configuration defines the host identity and realm for the node, and basic connection information and default routing behavior.
- Application configuration defines the Diameter application(s) that run on the node, and any optional configuration parameters passed to those applications.
- Peer configuration defines the other Diameter nodes with which this node operates.
- Routes configuration defines realm-based routes that the node can use when resolving messages.

The sections that follow describe how to configure each aspect of a Diameter node.

Creating a New Node Configuration (General Node Configuration)

Follow these steps to create a Diameter node configuration and target it to an existing WebRTC Session Controller engine tier instance:

1. Log in to the Administration Console for the WebRTC Session Controller domain you want to configure.
2. Click **Lock & Edit** to obtain a configuration lock.

   If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See `"Enable and disable the domain configuration lock"` in the *Administration Console Online Help* for more information.
3. In the **Domain Structure** tree, select **Diameter**.
4. Click **New** in the right pane to create a Diameter configuration.
5. Fill in the fields of the **Create a New Configuration** page as described in Table 5–1, then click Finish.

Table 5–1 Diameter Node General Configuration Properties

Property Name	Description
Name	Enter the administrative name for this Diameter node configuration.

Table 5–1 (Cont.) Diameter Node General Configuration Properties

Property Name	Description
Host	Enter the host identity of this Diameter node, or leave the field blank to automatically assign the host name of the target engine tier server as the Diameter node's host identity. The host identity may or may not match the DNS name. When configuring Diameter support for multiple Sh client nodes, it is best to omit the `host` element from the **diameter.xml** file. This omission enables you to deploy the same Diameter web Application to all servers in the engine tier cluster, and the host name is dynamically obtained for each server instance.
Realm	Enter the realm name for which this node has responsibility, or leave the field blank to use the domain name portion of the target engine tier server's fully-qualified host name (for example, host@oracle.com). You can run multiple Diameter nodes on a single host using different realms and listen port numbers. **Note:** An HSS, Application Server, and relay agents must all agree on a realm name or names. The realm name for the HSS and Application Server need not match.
Address	Enter the listen address for this Diameter node, using either the DNS name or IP address, or leave the field blank to use the host identity as the listen address. **Note:** The host identity may or may not match the DNS name of the Diameter node. Oracle recommends configuring the Address property with an explicit DNS name or IP address to avoid configuration errors.
TLS	Select this option if the Diameter node is configured with support for TLS (diameters network channels). This field advertises TLS capabilities when the node is interrogated by another Diameter node.
Debug	Select this option to enable debug message output. Debug messages are disabled by default.
Dynamic Peers Allowed	Select this option to allow dynamic discovery of Diameter peer nodes. Dynamic peer support is disabled by default. Oracle recommends enabling dynamic peers only when using the TLS transport, because no access control mechanism is available to restrict hosts from becoming peers.
Peer Retry Delay	Enter the amount of time, in seconds, this node waits before retrying a request to a Diameter peer. The default value is 30 seconds.
Request Timeout	Enter the amount of time, in milliseconds, this node waits for an answer message before timing out.
Watchdog Timeout	Enter the number of seconds this node uses for the value of the Diameter Tw watchdog timer interval.
Targets	Enter one or more target engine tier server names. The Diameter node configuration only applies to servers listed in this field.
Default Route Action	Specify an action type that describes the role of this Diameter node when using a default route. The value of this element can be one of the following: ■ none ■ local ■ relay ■ proxy ■ redirect
Default Route Servers	Specifies one or more target servers for the default route. Any server you include in this element must also be defined as a peer to this Diameter node, or dynamic peer support must be enabled.

6. Click **Finish**.
7. Click **Activate Changes** to apply the configuration to target servers.

After creating a general node configuration, the configuration name appears in the list of Diameter nodes. You can select the node to configure Diameter applications, peers, and routes, as described in the sections that follow.

Configuring Diameter Applications

Each Diameter node can deploy one or more applications. To configure Diameter Rx applications:

1. Log in to the Administration Console for the WebRTC Session Controller domain you want to configure.
2. Click **Lock & Edit** to obtain a configuration lock.

   If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See `"Enable and disable the domain configuration lock"` in the Administration Console Online Help for more information.
3. In the **Domain Structure** tree, select **Diameter**.
4. In the **Diameter Configurations** table, select the name of a Diameter node configuration.
5. Select the **Applications** tab.
6. Click **New** to configure a new Diameter application, or select an existing application configuration from the table.
7. Fill in the application properties as follows:
   - **Application Name**: Enter a name for the application configuration.
   - **Class Name**: Enter the classname of the application to deploy on this node.
   - **Parameters**: Enter optional parameters to pass to the application upon startup.
8. Click **Finish** to create the new application configuration.
9. Click **Activate Changes** to apply the configuration to the Diameter node.

Configuring the Rx Client Application

The WebRTC Session Controller Rx client application enables SIP Servlets to issue PCRF messages using the IMS Rx interface. To configure the Rx application, specify the class `com.bea.wcp.diameter.charging.RxApplication`.

See the chapter on using policy data in messages in *WebRTC Session Controller Extension Developer's Guide* for more information about using the Rx application API in deployed applications.

Configuring Peer Nodes

A Diameter node should define peer connection information for each other Diameter node in the realm, or enable dynamic peers in combination with TLS transport to allow peers to be recognized automatically.

To configure Diameter Peer Nodes:

1. Log in to the Administration Console for the WebRTC Session Controller domain you want to configure.

2. Click **Lock & Edit** to obtain a configuration lock.

   If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See "Enable and disable the domain configuration lock" in the *Administration Console Online Help* for more information.

3. In the **Domain Structure** tree, select **Diameter**.
4. In the **Diameter Configurations** table, select the name of a Diameter node configuration you want to add a peer to.
5. Select the **Peers** tab.
6. Click **New** to define a new peer entry.
7. Fill in the fields of the **Create a New Peer** page as follows:
   - **Host**: Enter the peer node's host identity.
   - **Address**: Enter the peer node's address (DNS name or IP address).
   - **Port Number**: Enter the listen port number of the peer node.
   - **Protocol**: Select the protocol used to communicate with the peer (TCP or SCTP).

     Note: WebRTC Session Controller attempts to connect to the peer using *only* the protocol you specify (TCP or SCTP). The other protocol is not used, even if a connection fails using the selected protocol. TCP is used as by default if you do not specify a protocol.

   - **Watchdog**: Indicate whether the peer supports the Diameter Tw watchdog timer interval.
8. Click **Finish** to create the new peer entry.
9. Click **Activate Changes** to apply the configuration.

Configuring Routes

Certain Diameter nodes, such as relays, should configure realm-based routes for use when resolving Diameter messages. You configure Diameter routes in the Administration Console.

To configure Diameter routes:

1. Log in to the Administration Console for the WebRTC Session Controller domain you want to configure.
2. Click **Lock & Edit** to obtain a configuration lock.

   If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See "Enable and disable the domain configuration lock" in the Administration Console Online Help for more information.

3. In the **Domain Structure** tree, select **Diameter**.
4. In the **Diameter Configurations** table, select the name of a Diameter node you want to configure a route for.
5. Select the **Routes** tab.
6. Click **New** to configure a new Route.

7. Fill in the fields of the **Create a New Route** page as follows:
   - **Name**: Enter an administrative name for the route.
   - **Realm**: Enter the target realm for this route.
   - **Application ID**: Enter the target Diameter application ID for this route.
   - **Action**: Select an action that this node performs when using the configured route. The action type may be one of: none, local, relay, proxy, or redirect.
   - **Server Names**: Enter the names of target servers that will use the route.
8. Click **Finish** to create the new route entry.
9. Click **Activate Changes** to apply the configuration.

Troubleshooting Diameter Configurations

SIP Servlets deployed on WebRTC Session Controller use the available Diameter applications to initiate requests for PCRF information. If a SIP Servlet performing these requests generates an error similar to:

```
Failed to dispatch Sip message to servlet ServletName
java.lang.IllegalArgumentException: No registered provider for protocol: Protocol
```

The message may indicate that you have not properly configured the associated Diameter application for the protocol. See "Configuring Diameter Applications" for more information.

If you experience problems connecting to a Diameter peer node, verify that you have configured the correct protocol for communicating with the peer in "Configuring Peer Nodes". Be aware that WebRTC Session Controller tries only the protocol you specify for the peer configuration (or TCP if you do not specify a protocol).

6

Configuring WebRTC Session Controller Media Engine

This chapter provides an overview of the configuration of the Oracle WebRTC Session Controller Media Engine.

About WebRTC Session Controller Media Engine Configuration

You configure the WebRTC Session Controller Media Engine using one of the following interfaces: a command-line interface (CLI), web interface or SNMP

- Command-Line Interface (CLI) through console, telnet or secure shell (SSH)
- Http/web using a browser interface provided by the Media Engine
- SNMP Interface
- HTTP/SOAP/WSDL Interface

Configuring the Media Engine does not use the WebLogic domain Administration Console For more information about setting up and accessing Media Engine interfaces see the chapter on Managing and Administering Net-Net OS-E Systems in *Oracle Communications Net-Net OS-E System Administration Guide*.

Common Media Engine Configuration Tasks

You must configure the WebRTC Session Controller Media Engine to receive requests from Signaling Engines and to forward the requests to your SIP network infrastructure. The Media Engine also allows requests originating from SIP clients on your network to communicate with WebRTC clients through the Signaling Engine.

You perform the following Media Engine configuration tasks using of the Media Engine administration interfaces:

- Configuring Permissions, Users and Authorization
- Enabling Media Engine Interfaces and Protocols
- Enabling Media Engine Services
- Configuring Media Engine Domain Name Systems
- Configuring SIP Servers

The following sections provide a general overview of each task along with a reference to the related chapter in the *Oracle Communications Net-Net OS-E System Administration Guide*. For detailed information on each task, see the referenced chapter.

Configuring Permissions, Users and Authorization

Media Engine requires setting up users unique to those configured in the WebRTC Session Controller Oracle WebLogic domain. You must also assign the proper permissions and authorizations to each user to secure your Media Engine server.

For more information see the chapter on configuring permissions, user and authorizations in *Oracle Communications Net-Net OS-E System Administration Guide*.

Enabling Media Engine Interfaces and Protocols

You must configure the web and network interfaces used by the Signaling Engine and Media Engine to communicate with each other. The network configuration required by your environment determines whether you need to configure multiple IP interfaces, VLANs, and other network properties.

For more information on setting up the Media Engine network interfaces and properties see the chapter on enabling Net-Net OS-E interfaces and protocols in *Oracle Communications Net-Net OS-E System Administration Guide*.

Enabling Media Engine Services

The Media Engine provides several services and interfaces with external systems that are used with WebRTC Session Controller. These services include:

- Cluster-Master Services
- Directory Services
- Accounting Services
- Authentication Services
- OS-E Database
- Registration Services
- Server Load
- Call Failover (Signaling and Media)
- Load-Balancing
- File-Mirror
- Route Server
- Sampling
- Third-Party-Call-Control (3PCC)
- Logging
- Data and Archiving
- External Database Connections
- System Maintenance
- Back Up

The services you use depend on the WebRTC Session Controller implementation in your environment.

For more information on all of the services listed, including how to enable a service using an administration interface like CLI, see the chapter on enabling Net-Net OS-E Services in *Oracle Communications Net-Net OS-E System Administration Guide*.

Configuring Accounting and Archiving Services

The Media Engine can capture SIP call detail records (CDRs) and other accounting records associated with the SIP sessions generated by WebRTC Session Controller from WebRTC and SIP client requests.

For information on using Media Engine accounting features see the chapter on configuring Net-Net OS-E accounting and provisioning in *Oracle Communications Net-Net OS-E System Administration Guide*.

Configuring Media Engine Domain Name Systems

The Media Engine contains a configurable Domain Name Systems (DNS) resolver used to configure and cache heavily used static server mappings, including SIP service to SIP server mappings, and mappings resulting from interactions with external servers. You can also configure the Media Engine to ignore lookups involving specific domain names.

For more information on Media Engine DNS capabilities, see the chapter on configuring domain name systems in *Oracle Communications Net-Net OS-E System Administration Guide.*

Configuring SIP Servers

You must configure the Media Engine to communicate with SIP servers in your network infrastructure so WebRTC requests can be forwarded from external clients to your SIP clients. Media Engine supports SIP servers from multiple vendors.

For information on configuring the SIP server pools and SIP directories required by the Media Engine to communicate with your environment see the chapter on configuring SIP servers, directories and federations in *Oracle Communications Net-Net OS-E System Administration Guide*.

7

Configuring WebRTC Session Controller Container Properties

This chapter describes how to configure SIP container features in the engine tier of an Oracle Communications WebRTC Session Controller deployment.

Configure General SIP Application Server Properties

Loading SIP applications to the WebRTC Session Controller in the Administration Console is similar to loading any application to WebLogic server. You use the Deployments page in the Administration Console to load, update, or remove an application or module.

The WebRTC Session Controller defines general settings that apply to all SIP applications. Before deploying applications to the WebRTC Session Controller, you should verify and modify the default values for the general settings. You can configure the general settings in the SIP Server page of the Administration Console.

To configure general SIP application server properties:

1. Open the Administration Console for your domain.
2. Click the **SipServer** link in the Domain Structure pane.

   The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring WebRTC Session Controller. By default, the General configuration page appears.
3. Use the fields in the **General** subtab of the Configuration tab to configure the general settings applicable to serving SIP applications.

   Among the settings that determine common application handling are:

   - The default servlet invoked if a specific servlet is not identified for a request based on the servlet mapping rules.
   - Timer values. See "Configuring Timer Processing" for more information.
   - Header handling settings.
   - Application session settings.

   For details, see the on-screen field descriptions in the Administration Console.
4. Click **Save** to save your configuration changes.
5. Click **Activate Changes** to apply your changes to the engine tier servers.

Adding Servers to the WebRTC Session Controller Cluster

WebRTC Session Controller instances configured as replicated domains include the default **BEA_ENGINE_TIER_CLUST** and **BEA_DATA_TIER_CLUST** clusters for the signaling and data tiers. You can assign additional managed servers to each cluster as needed when performance requirements in your environment require them.

See *WebLogic Server Administration Console Online Help* for information on how to "Assign servers to clusters".

For more information on clustering, see "Understanding WebLogic Server Clustering" in *Oracle Fusion Middleware Using Clusters for Oracle WebLogic Server*.

Configuring Timer Processing

As engine tier servers add new call state data to the SIP data tier, SIP data tier instances queue and maintain the complete list of SIP protocol timers and application timers associated with each call. Engine tier servers periodically poll partitions in the SIP data tier to determine which timers have expired, given the current time. By default, multiple engine tier polls to the SIP data tier are staggered to avoid contention on the timer tables. Engine tier servers then process all expired timers using threads allocated in the **sip.timer.Default** execute queue.

Configuring Timer Affinity (Optional)

With the default timer processing mechanism, a given engine tier server processes all timers that are currently due to fire, regardless of whether that engine was involved in processing the calls associated with those timers. However, some deployment scenarios require that a timer is processed on the same engine server that last modified the call associated with that timer. One example of this scenario is a hot standby system that maintains a secondary engine that should not process any call data until another engine fails. WebRTC Session Controller enables you to configure timer affinity in such scenarios.

When you enable timer affinity, replicas request that each engine tier server periodically poll the SIP data tier for processed timers. When polling the SIP data tier, an engine processes only those timers associated with calls that were last modified by that engine, or timers for calls that have no owner.

> **Note:** When an engine tier server fails, any call states that were last modified by that engine no longer have an owner. Expired timers that have no owner are processed by the next engine server that polls the SIP data tier.

To enable timer affinity:

1. Access the Administration Console for your domain.
2. Select the **SipServer** node in the left pane. The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring WebRTC Session Controller.
3. Select the **Configuration**, then **General** tab in the right pane.
4. Select the box for **Enable Timer Affinity**.
5. Click Save to save your configuration changes.

6. Click Activate Changes to apply your changes to the engine tier servers.

The Enable Timer Affinity setting is persisted in **sipserver.xml** in the **enable-timer-affinity** element.

Configuring NTP for Accurate SIP Timers

In order for the SIP protocol stack to function properly, all engine and SIP data tier servers must accurately synchronize their system clocks to a common time source, to within one or two milliseconds. Large differences in system clocks cause severe problems such as:

- SIP timers firing prematurely on servers with fast clock settings.
- Poor distribution of timer processing in the engine tier. For example, one engine tier server may process all expired timers, whereas other engine tier servers process no timers.

Oracle recommends using a Network Time Protocol (NTP) client or daemon on each WebRTC Session Controller instance and synchronizing to a common NTP server.

Caution: You must accurately synchronize server system clocks to a common time source (to within one or two milliseconds) in order for the SIP protocol stack to function properly. Because the initial T1 timer value of 500 milliseconds controls the retransmission interval for INVITE request and responses, and also sets the initial values of other timers, even small differences in system clock settings can cause improper SIP protocol behavior. For example, an engine tier server with a system clock 250 milliseconds faster than other servers will process more expired timers than other engine tier servers, will cause retransmits to begin in half the allotted time, and may force messages to timeout prematurely.

8

Configuring SIP Data Tier Partitions and Replicas

This chapter describes how to configure Oracle Communications WebRTC Session Controller instances that form the SIP data tier cluster of a deployment.

Overview of SIP Data Tier Configuration

The WebRTC Session Controller SIP data tier is a cluster of server instances that manages the application call state for concurrent SIP calls. The SIP data tier may manage a single copy of the call state or multiple copies as needed to ensure that call state data is not lost if a server fails or network connections are interrupted.

The SIP data tier cluster is arranged into one or more *partitions*. A partition consists of one or more SIP data tier server instances that manage the same portion of concurrent call state data. In a single-server WebRTC Session Controller installation, or in a two-server installation where one server resides in the engine tier and one resides in the SIP data tier, all call state data is maintained in a single partition. Multiple partitions are required when the size of the concurrent call state exceeds the maximum size that can be managed by a single server instance. When more than one partition is used, the concurrent call state is split among the partitions, and each partition manages an separate portion of the data. For example, with a two-partition SIP data tier, one partition manages the call state for half of the concurrent calls (for example, calls A through M) while the second partition manages the remaining calls (N through Z).

In most cases, the maximum call state size that can be managed by an individual server corresponds to the Java Virtual Machine limit of approximately 1.6 GB per server.

Additional servers can be added within the same partition to manage copies of the call state data. When multiple servers are members of the same partition, each server manages a copy of the same portion of the call data, referred to as a *replica* of the call state. If a server in a partition fails or cannot be contacted due to a network failure, another replica in the partition supplies the call state data to the engine tier. Oracle recommends configuring two servers in each partition for production installations, to guard against system or network failures. A partition can have a maximum of three replicas for providing additional redundancy.

datatier.xml Configuration File

The **datatier.xml** configuration file is located in the

Middleware_Home/**user_projects/domains/***domain_home*/**config/custom**

subdirectory of the domain directory, where *Middleware_Home* is the WebRTC Session Controller installation directory and *domain_home* is your domain directory.

This file identifies SIP data tier servers and also defines the partitions and replicas used to manage the call state. If a server's name is present in **datatier.xml**, that server loads WebRTC Session Controller SIP data tier functionality at start time. (Server names that do not appear in **datatier.xml** act as engine tier nodes, and instead provide SIP Servlet container functionality configured by the **sipserver.xml** configuration file.)

The sections that follow show examples of the **datatier.xml** contents for common SIP data tier configurations. See Chapter 19, "SIP Data Tier Configuration Reference (datatier.xml)" for complete information about the XML Schema and its elements.

Configuration Requirements and Restrictions

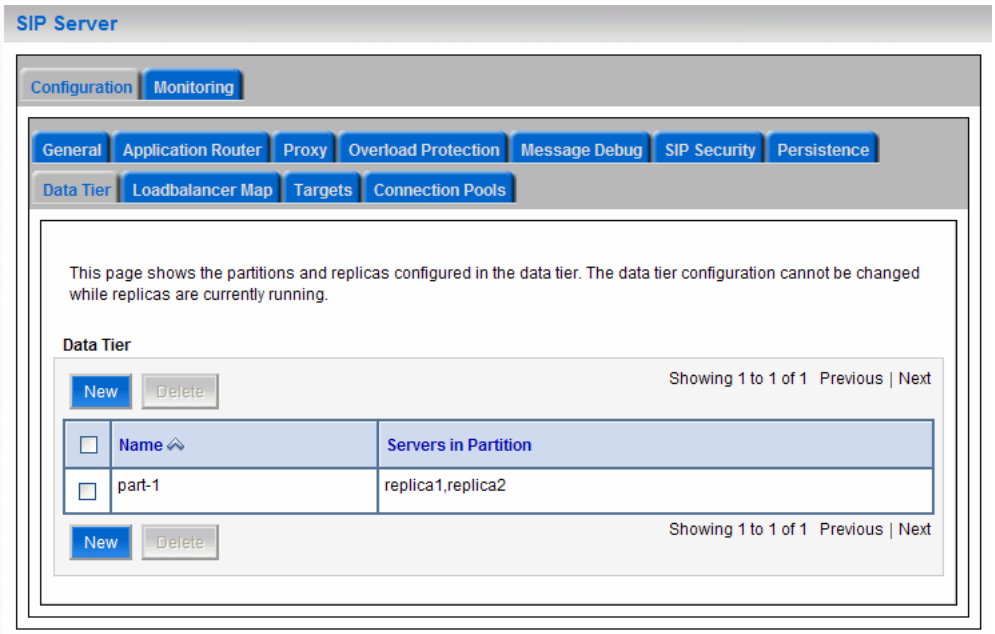
All servers that participate in the SIP data tier should be members of the same WebLogic Server cluster. The cluster configuration enables each server to monitor the status of other servers. Using a cluster also enables you to easily target the **sipserver** and **datatier** custom resources to all servers for deployment.

For high reliability, you can configure up to three replicas within a partition.

You cannot change the SIP data tier configuration on running replicas or engine tier nodes. You must restart servers in the domain to change SIP data tier membership or reconfigure partitions or replicas.

To view and configure the current SIP data tier configuration in the Administration Console, click the **SipServer** node, click the **Configuration** tab, and then click the **Data Tier** tab. The SIP data tier configuration page is shown in Figure 8–1.

Figure 8–1 SIP Data Tier Configuration Screen



Best Practices for Configuring and Managing SIP Data Tier Servers

Adding replicas can increase reliability for the system as a whole, however, each additional server in a partition requires additional network bandwidth to manage the

replicated data. With three replicas in a partition, each transaction that modifies the call state updates data on three different servers.

To ensure high reliability when using replicas, always ensure that server instances in the same partition reside on different machines. Hosting two or more replicas on the same system leaves all of the hosted replicas vulnerable to a system or network failure.

SIP data tier servers can have one of three different statuses:

- **ONLINE**: indicates that the server is available for managing call state transactions.
- **OFFLINE**: indicates that the server is shut down or unavailable.
- **ONLINE_LOCK_AUTHORITY_ONLY**: indicates that the server was restarted and is currently being updated (from other replicas) with the current call state data. A recovering server cannot yet process call state transactions, because it does not maintain a full copy of the call state managed by the partition.

If you need to take a SIP data tier server instance offline for scheduled maintenance, ensure that at least one other server in the same partition is active. If you shut down an active server and all other servers in the partition are offline or recovering, you will lose a portion of the active call state.

In a WebRTC Session Controller installation with a set of engine tier nodes and tier nodes in a partition, if the connection to the SIP data tier becomes "split" such that each engine tier server can only reach a different SIP data tier node, one of the replicas is forced offline.

To recover from this situation, always configure the Node Manager utility to restart SIP data tier replicas automatically when a replica fails. This configuration enables the replica to rejoin its associated partition and update its copy of the call state data without having to manually restart the server.

WebRTC Session Controller automatically divides the call state evenly over all configured partitions.

When configuring the data tier cluster, you have the option of choosing unicast or multicast as the mechanism by which servers in a cluster communicate with each other. While unicast may be easier to configure, for most production deployments, Oracle recommends use of multicast communication. Using multicast reduces the likelihood of performance degradation that may result from excessive work load on group leaders and retransmissions during periods of high traffic. For more information on clustering, see `"Setting up WebLogic Clusters"` in *Administering Clusters for Oracle WebLogic Server*.

If you configure two or more SIP data tier replicas using the default WebLogic Server Listen Address configuration (which specifies no listen address), multiple SIP data tier instances on the same system cannot connect to one another. This occurs because, using the default Listen Address configuration, JNDI objects in the first started server bind to all local IP addresses.

To avoid this problem, always enter a valid IP address for each configured SIP data tier server instance.

Example SIP Data Tier Configurations and Configuration Files

The sections that follow describe some common WebRTC Session Controller installations that use a separate SIP data tier.

SIP Data Tier with One Partition

A single-partition, single-server SIP data tier represents the simplest data tier configuration. Example 8–1 shows a SIP data tier configuration for a single-server deployment.

Example 8–1 SIP Data Tier Configuration for Small Deployment

```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
    <partition>
      <name>part-1</name>
      <server-name>replica1</server-name>
    </partition>
  </data-tier>
```

To add a replica to an existing partition, define a second **server-name** entry in the same partition. For example, the **datatier.xml** configuration file shown in Example 8–2 recreates a two-replica configuration.

Example 8–2 SIP Data Tier Configuration for Small Deployment with Replication

```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
    <partition>
      <name>Partition0</name>
      <server-name>DataNode0-0</server-name>
      <server-name>DataNode0-1</server-name>
    </partition>
  </data-tier>
```

SIP Data Tier with Two Partitions

Multiple partitions can be easily created by defining multiple **partition** entries in **datatier.xml**, as shown in Example 8–3.

Example 8–3 Two-Partition SIP Data Tier Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
    <partition>
      <name>Partition0</name>
      <server-name>DataNode0-0</server-name>
    </partition>
    <partition>
      <name>Partition1</name>
      <server-name>DataNode1-0</server-name>
    </partition>
  </data-tier>
```

SIP Data Tier with Two Partitions and Two Replicas

Replicas of the call state can be added by defining multiple SIP data tier servers in each partition. Example 8–4 shows the **datatier.xml** configuration file used to define a system having two partitions with two servers (replicas) in each partition.

Example 8–4 SIP Data Tier Configuration for Small Deployment

```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
```

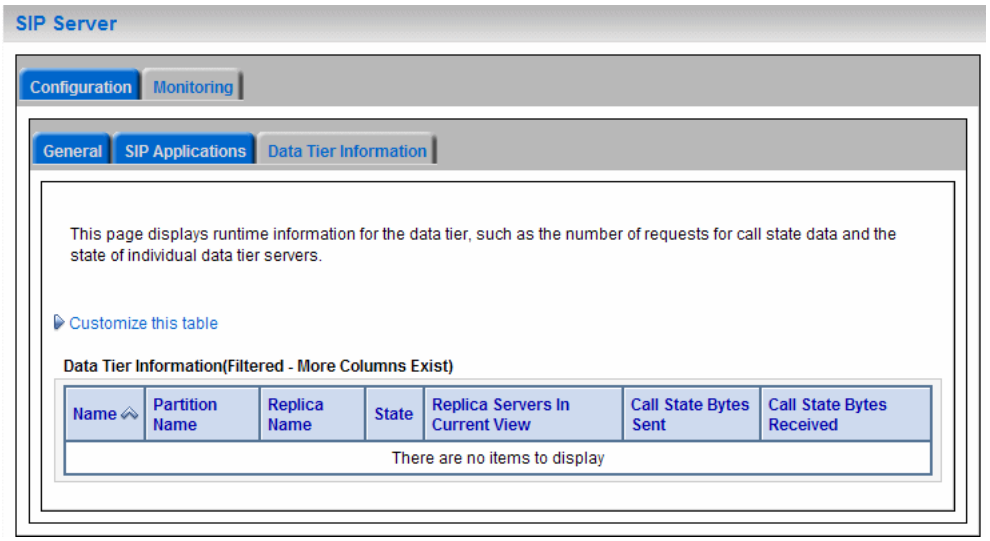
```
    <partition>
      <name>Partition0</name>
      <server-name>DataNode0-0</server-name>
      <server-name>DataNode0-1</server-name>
    </partition>
    <partition>
      <name>Partition1</name>
      <server-name>DataNode1-0</server-name>
      <server-name>DataNode1-1</server-name>
    </partition>
  </data-tier>
```

Monitoring and Troubleshooting SIP Data Tier Servers

A run-time MBean, **ReplicaRuntimeMBean**, provides valuable information about the current state and configuration of the SIP data tier. See *WebRTC Session Controller JavaScript API Reference* for a description of the attributes provided in this MBean.

Many of the attributes can be viewed in the Administration Console by navigating to the **SipServer** node, clicking the **Monitoring** tab, and then clicking the **Data Tier Information** tab, as shown in Figure 8–2.

Figure 8–2 Data Tier Information Tab



Example 8–5 shows a simple WLST session that queries the current attributes of a single Managed Server instance in a SIP data tier partition. Table 8–1 (following the example) describes the MBean services in more detail.

Example 8–5 Displaying ReplicaRuntimeMBean Attributes

```
connect('weblogic','weblogic','t3://datahost1:7001')
custom()
cd('com.bea')
cd('com.bea:ServerRuntime=replica1,Name=replica1,Type=ReplicaRuntime')
ls()
-rw-   BackupStoreInboundStatistics                null
-rw-   BackupStoreOutboundStatistics               null
-rw-   BytesReceived                               0
-rw-   BytesSent                                   0
-rw-   CurrentViewId                               2
```

```
-rw-   DataItemCount                          0
-rw-   DataItemsToRecover                     0
-rw-   DatabaseStoreStatistics                null
-rw-   HighKeyCount                           0
-rw-   HighTotalBytes                         0
-rw-   KeyCount                               0
-rw-   Name                                   replica1
-rw-   Parent                                 com.bea:Name=replica1,Type=S
erverRuntime
-rw-   PartitionId                            0
-rw-   PartitionName                          part-1
-rw-   ReplicaId                             0
-rw-   ReplicaName                           replica1
-rw-   ReplicaServersInCurrentView           java.lang.String[replica1,
replica2]
-rw-   ReplicasInCurrentView                 [I@75378c
-rw-   State                                  ONLINE
-rw-   TimerQueueSize                        0
-rw-   TotalBytes                            0
-rw-   Type                                   ReplicaRuntime
```

Table 8–1 ReplicaRuntimeMBean Method and Attribute Summary

Method/Attribute	Description
`dumpState()`	Records the entire state of the selected SIP data tier server instance to the WebRTC Session Controller log file. You may use the `dumpState()` method to provide additional diagnostic information to a Technical Support representative if a problem arises.
`BackupStoreInboundStatistics`	Provides statistics about call state data replicated from a remote geographical site.
`BackupStoreOutboundStatistics`	Provides statistics about call state data replicated to a remote geographical site.
`BytesReceived`	The total number of bytes received by this SIP data tier server. Bytes are received as servers in the engine tier provide call state data to be stored.
`BytesSent`	The total number of bytes sent from this SIP data tier server. Bytes are sent to engine tier servers when requested to provide the stored call state.
`CurrentViewId`	The current view ID. Each time the layout of the SIP data tier changes, the view ID is incremented. For example, as multiple servers in a SIP data tier cluster are started for the first time, the view ID is incremented when each server begins participating in the SIP data tier. Similarly, the view is incremented if a server is removed from the SIP data tier, either intentionally or due to a failure.
`DataItemCount`	The total number of stored call state keys for which this server has data. This attribute may be lower than the `KeyCount` attribute if the server is currently recovering data.
`DataItemsToRecover`	The total number of call state keys that must still be recovered from other replicas in the partition. A SIP data tier server may recover keys when it has been taken offline for maintenance and is then restarted to join the partition.
`HighKeyCount`	The highest total number of call state keys that have been managed by this server since the server was started.
`HighTotalBytes`	The highest total number of bytes occupied by call state data that this server has managed since the server was started.

Table 8–1 (Cont.) ReplicaRuntimeMBean Method and Attribute Summary

Method/Attribute	Description
`KeyCount`	The number of call data keys that are stored on the replica.
`PartitionId`	The numeric partition ID (from 0 to 7) of this server's partition.
`PartitionName`	The name of this server's partition.
`ReplicaId`	The numeric replica ID (from 0 to 2) of this server's replica.
`ReplicaName`	The name of this server's replica.
`ReplicaServersInCurrentView`	The names of other WebRTC Session Controller instances that are participating in the partition.
`State`	The current state of the replica. SIP data tier servers can have one of three statuses: ■ **ONLINE**: indicates that the server is available for managing call state transactions. ■ **OFFLINE**: indicates that the server is shut down or unavailable. ■ **ONLINE_LOCK_AUTHORITY_ONLY**: indicates that the server was restarted and is currently being updated (from other replicas) with the current call state data. A recovering server cannot yet process call state transactions, because it does not maintain a full copy of the call state managed by the partition.
`TimerQueueSize`	The current number of timers queued on the SIP data tier server. This generally corresponds to the KeyCount value, but may be less if new call states are being added but their associated timers have not yet been queued. **Note:** Engine tier servers periodically check with SIP data tier instances to determine if timers associated with a call have expired. In order for SIP timers to function properly, all engine tier servers must actively synchronize their system clocks to a common time source. Oracle recommends using a Network Time Protocol (NTP) client or daemon on each engine tier instance and synchronizing to a selected NTP server. See "Configuring Timer Processing".
`TotalBytes`	The total number of bytes consumed by the call state managed in this server.

9

Configuring Network Connection Settings

This chapter describes how to configure network resources for use with Oracle Communications WebRTC Session Controller.

Overview of Network Configuration

The default HTTP network configuration for each WebRTC Session Controller instance is determined from the Listen Address and Listen Port setting for each server. However, WebRTC Session Controller does not support the SIP protocol over HTTP. The SIP protocol is supported over the UDP and TCP transport protocols. SIPS is also supported using the TLS transport protocol.

To enable UDP, TCP, or TLS transports, you configure one or more **network channels** for a WebRTC Session Controller instance. A network channel is a configurable Oracle WebLogic Server resource that defines the attributes of a specific network connection to the server instance. Basic channel attributes include:

- The protocols supported by the connection
- The listen address (DNS name or IP address) of the connection
- The port number used by the connection
- (optional) The port number used by outgoing UDP packets
- The public listen address to embed in SIP headers when the channel is used for an outbound connection. This is typically the IP address presented by the IP sprayer or external load balancer as the virtual IP (VIP) for the telecommunication services.

You can assign multiple channels to a single WebRTC Session Controller instance to support multiple protocols or to use multiple interfaces available with multihomed server hardware. You cannot assign the same channel to multiple server instances.

When you configure a new network channel for the SIP protocol, both the UDP and TCP transport protocols are enabled on the specified port. You cannot create a SIP channel that supports only UDP transport or only TCP transport. When you configure a network channel for the SIPS protocol, the server uses the TLS transport protocol for the connection.

As you configure a new SIP Server domain, you will generally create multiple SIP channels for communication to each engine tier server in your system. Engine tier servers can communicate to SIP data tier replicas using the configured Listen Address attributes for the replicas. Note, however, that replicas must use unique Listen Addressees to communicate with one another.

Note: If you configure multiple replicas in a SIP data tier cluster, you must configure a unique Listen Address for each server (a unique DNS name or IP address). If you do not specify a unique Listen Address, the replica service binds to the default *localhost* address and multiple replicas cannot locate one another.

Configuring External IP Addresses in Network Channels

When you set up a network channel for your WebRTC Session Controller instance, you must specify the public IP address that external clients use to address the instance. In most cases, this address is presented by an IP sprayer or external load balancer or other network element capable of exposing a virtual IP (VIP) on behalf of the WebRTC Session Controller to the external network.

You configure the client-facing address as the external listen address. When a SIP channel has an external listen address that differs from the channel's primary listen address, WebRTC Session Controller embeds the host and port number of the external address in SIP headers, such as in the Response header. This causes subsequent messages from external clients to be directed to the public address rather than the local engine tier server address (which may not be accessible to clients).

If an external listen address is not specified for the network channel, the WebRTC Session Controller embeds the primary listen address for the channel in the headers.

If you have more than one IP sprayer or load balancer that may receive external traffic addressed to the WebRTC Session Controller servers, you must define a channel on each engine tier server for each one. When a particular network interface on the engine tier server is selected for outbound traffic, the network channel associated with the network interface card's (NIC's) address is examined to determine the external listen address to embed in SIP headers.

If your system uses a multihomed IP sprayer or load balancer having two public addresses, you must also define a pair of channels to configure both public addresses. If the engine tier server has only one NIC, you must define a second, logical address on the NIC to configure a dedicated channel for the second public address. In addition, you must configure your IP routing policies to define which logical address is associated with each public address.

About IPv4 and IPv6 Support

If your operating system and hardware support IPv6, you can also configure WebRTC Session Controller to use IPv6 for network communication. Enable IPv6 for SIP traffic by configuring a network channel with an IPv6 address. You must configure an IPv6 SIP channel on each engine tier server that will support IPv6 traffic.

Each SIP network channel configured on an engine supports either IPv6 or IPv4 traffic. You cannot mix IPv4 and IPv6 traffic on a single channel. You can configure a single engine with both an IPv4 and IPv6 channel to support multiple, separate networks.

It is also possible for WebRTC Session Controller engine and SIP data tier nodes to communicate on IPv4 (or IPv6) while supporting the other protocol version for external SIP traffic. To configure engine and SIP data tier nodes on an IPv6 network, simply specify IPv6 listen addresses for each server instance.

Enabling DNS Support

WebRTC Session Controller supports DNS for resolving the transport, IP address and port number of a proxy required to send a SIP message. This matches the behavior described in RFC 3263 (http://www.ietf.org/rfc/rfc3263.txt). DNS may also be used when routing responses to resolve the IP address and port number of a destination.

Caution: Because multihome resolution is performed within the context of SIP message processing, any multihome performance problems result in increased latency performance. Oracle recommends using a caching multihome server in a production environment to minimize potential performance problems.

To configure DNS support:

1. Log in to the Administration Console for the WebRTC Session Controller domain you want to configure.
2. Select the **SipServer** node in the left pane of the Console.
3. Select the **Configuration**, and then select the **General** tab in the right pane.
4. Select the option for **Enable DNS Server Lookup**.
5. Click **Save** to save your changes.

When you enable DNS lookup, the server can use DNS to:

- Discover a proxy server's transport, IP address, and port number when a request is sent to a SIP URI.
- Resolve an IP address and port number during response routing, depending on the contents of the Sent-by field.

For proxy discovery, WebRTC Session Controller uses DNS resolution only once per SIP transaction to determine transport, IP, and port number information. All retransmissions, ACKs, or CANCEL requests are delivered to the same address and port using the same transport. For details about how DNS resolution takes place, see RFC 3263 (http://www.ietf.org/rfc/rfc3263.txt).

When a proxy is required to send a response message, WebRTC Session Controller uses DNS lookup to determine the IP address and port number of the destination, using the information provided in the **sent-by** field and the **Via** the header.

Configuring Network Channels for SIP or SIPS

When you create a domain using the Configuration Wizard, WebRTC Session Controller instances are configured with a default network channel supporting the SIP protocol over UDP and TCP. This default channel is configured to use Listen Port 5060, but specifies no Listen Address. Follow the instructions in "Reconfiguring an Existing Channel" to change the default channel's listen address or listen port settings. See "Creating a New SIP or SIPS Channel" for information on creating a new channel resource to support additional protocols or additional network interfaces.

Reconfiguring an Existing Channel

You cannot change the protocol supported by an existing channel. To reconfigure an existing listen address/port combination to use a different network protocol, you must

delete the existing channel and create a channel using the instructions in "Creating a New SIP or SIPS Channel".

To reconfigure a channel:

1. Log in to the Administration Console for the WebRTC Session Controller domain you want to configure.
2. In the left pane, select the **Environment** entry to display its contents. Select **Servers** from the displayed entries.
3. In the right pane, select the name of the server you want to configure.
4. Select **Protocols**, then select the **Channels** tab to display the configured channels.
5. To delete an existing channel, select it in the table and click **Delete**.
6. To reconfigure an existing channel:
   a. Select the channel's link from **Name** column of the channel list (for example, the default **SIP** channel).
   b. Edit the **Listen Address** or **Listen Port** fields to correspond to the address of a NIC or logical address on the associated engine tier server.

      Note: The channel must be disabled before you can modify the listen address or listen port. Disable the channel by deselecting the **Enabled** check box.

   c. Set the External Listen Address or External Listen Port fields to the destination address and port addressed by external clients. This is typically the VIP address presented by an external load balancer or IP sprayer in your system.
   d. Edit the advanced channel attributes as necessary (see "Creating a New SIP or SIPS Channel" for details.)
7. Click **Save**.

Creating a New SIP or SIPS Channel

To add a new SIP or SIPS channel to the configuration of a WebRTC Session Controller instance:

1. Log in to the Administration Console for the WebRTC Session Controller domain you want to configure.
2. In the left pane, select the **Environment** node, and then select the **Servers** tab.
3. In the right pane, select the name of the server you want to configure.
4. Select the **Protocols** tab, then select the **Channels** tab to display the configured channels.
5. Click **New** to configure a new channel.
6. Fill in the new channel fields as follows:
   - **Name:** Enter an administrative name for this channel, such as *SIPS-Channel-eth0*.
   - **Protocol:** Select either **sip** to support UDP and TCP transport, or **sips** to support TLS transport. A SIP channel cannot support only UDP or only TCP transport on the configured port.

7. Click **Next**.
8. Fill in the new channel's addressing fields as follows:
   - **Listen Address:** Enter the IP address or DNS name for this channel. On a DNS server, enter the exact IP address of the interface you want to configure, or a multihome name that maps to the exact IP address.
   - **Listen Port:** Enter the port number used to communication through this channel. The combination of Listen Address and Listen Port must be unique across all channels configured for the server. SIP channels support both UDP and TCP transport on the configured port.
   - **External Listen Address** and **External Listen Port**: Edit these fields to match the external address and port used by clients to address the system. This is typically a virtual IP address presented by an external load balancer or IP sprayer.

     If this value differs from the **Listen Address** value, the WebRTC Session Controller embeds this value in SIP message headers for further call traffic.
9. Click **Next**.
10. Set the additional channel properties listed below if required:
    - **Enabled**: This attribute specifies whether to start the new channel.
    - **Tunneling Enabled**: This attribute specifies whether tunneling through HTTP should be enabled for this network channel. This value is not inherited from the server's configuration.
    - **HTTP Enabled for This Protocol**: This attribute cannot be selected for SIP and SIPS channels, because WebRTC Session Controller does not support HTTP transport SIP protocols.
    - **Outbound Enabled**: This attribute cannot be unchecked, because all SIP and SIPS channels can originate network connections.
11. Click **Finish**.

Configuring Custom Timeout, MTU, and Other Properties

SIP channels can be further configured using one or more custom channel properties. The custom properties cannot be set using the Administration Console. Instead, you must use a text editor to add the properties to a single, **custom-property** stanza in the channel configuration portion of the **config.xml** file for the domain.

WebRTC Session Controller provides the following custom properties that affect the transport protocol of SIP channels:

- **TcpConnectTimeoutMillis**: Specifies the amount of time WebRTC Session Controller waits before it declares a destination address (for an outbound TCP connection) as unreachable. The property is applicable only to SIP channels; WebRTC Session Controller ignores this attribute value for SIPS channels. A value of 0 disables the timeout completely. A default value of 3000 milliseconds is used if you do not specify the custom property.
- **SctpConnectTimeoutMillis**: Specifies the amount of time WebRTC Session Controller waits before it declares a destination address (for an outbound SCTP connection) as unreachable. The property is applicable only to SCTP channels (for Diameter traffic). A value of 0 disables the timeout completely. A default value of 3000 milliseconds is used if you do not specify the custom property. See

"Configuring Static Source Port for Outbound UDP Packets" for information about creating SCTP channels for Diameter.

- **SourcePorts**: Configures one or more static port numbers that a server uses for originating UDP packets.

  Caution: Oracle does not recommend using the SourcePorts custom property in most configurations because it degrades performance. Configure the property only in cases where you must specify the exact ports that WebRTC Session Controller uses to originate UDP packets.

- **Mtu**: Specifies the Maximum Transmission Unit (MTU) value for this channel. A value of -1 uses the default MTU size for the transport.
- **EnabledProtocolVersions**: Specifies the version of the SSL protocol to use with this channel when WebRTC Session Controller acts as an SSL client. When acting as an SSL client, by default the channel requires TLS V1.0 as the supported protocol. You can configure the server to use SSL V3.0 as well, if that is the highest version that the SSL peer servers support. You can set one of the following values for this property:
  - **TLS1**, the default, configures the channel to send and accept only TLS V1.0 messages. Peers must respond with a TLS V1.0 message, or the SSL connection is dropped.
  - **SSL3** configures the channel to send and accept only SSL V3.0 messages. Peers must respond with an SSL V3.0 message, or the SSL connection is dropped.
  - **ALL** supports either TLS V1.0 or SSL V3.0 messages. Peers must respond with a TLS V1.0 or SSL V3.0 message, or the SSL connection is dropped.

To configure a custom property, use a text editor to modify the **config.xml** file directly, or use a JMX client such as WLST to add the custom property. When editing **config.xml** directly, ensure that you add only one custom-properties element to the end of a channel's configuration stanza. Separate multiple custom properties within the same element using semicolons (;) as shown in Example 9–1.

Example 9–1 Setting Custom Properties

```
<network-access-point>
  <name>sip</name>
  <protocol>sip</protocol>
  <listen-port>5060</listen-port>
  <public-port>5060</public-port>
  <http-enabled-for-this-protocol>false</http-enabled-for-this-protocol>
  <tunneling-enabled>false</tunneling-enabled>
  <outbound-enabled>true</outbound-enabled>
  <enabled>true</enabled>
  <two-way-ssl-enabled>false</two-way-ssl-enabled>
  <client-certificate-enforced>false</client-certificate-enforced>

<custom-properties>EnabledProtocolVersions=ALL;Mtu=1000;SourcePorts=5060</custom-p
roperties>
</network-access-point>
```

Configuring SIP Channels for Multihomed Machines

If you are configuring a server that has multiple network interfaces (a "multihomed" server), you must configure a separate network channel for each IP address used by WebRTC Session Controller. WebRTC Session Controller uses the listen address and listen port values for each channel when embedding routing information into SIP message system headers.

Note: If you do not configure a channel for a particular IP address on a multihomed system, that IP address cannot be used when populating **Via**, **Contact**, and **Record-Route** headers.

Configuring Engine Servers to Listen on Any IP Interface

To configure WebRTC Session Controller to listen for UDP traffic on any available IP interface, create a SIP channel and specify **0.0.0.0** (or `::` for IPv6 networks) as the listen address. You must still configure at least one additional channel with an explicit IP address to use for outgoing SIP messages. (For multihomed machines, each interface used for outgoing messages must have a configured channel.)

Note: You must configure the 0.0.0.0 address directly on the server's network channel. If you configure a SIP channel without specifying the channel listen address, but you do configure a listen address for the server itself, then the SIP channel inherits the server listen address. In this case the SIP channel *does not* listen on IP_ANY.

Note: Using the `0.0.0.0` configuration affects only UDP traffic on Linux platforms. WebRTC Session Controller only creates TCP and HTTP listen threads corresponding to the configured host name of the server, and localhost. If multiple addresses are mapped to the host name, WebRTC Session Controller displays warning messages upon startup. To avoid this problem and listen on all addresses, specify the `::` address, which encompasses all available addresses for both IPv6 and IPv4 for HTTP and TCP traffic as well.

Configuring Static Source Port for Outbound UDP Packets

You can optionally use a static port rather than a dynamically assigned ephemeral port as the source port for outgoing UDP datagrams. WebRTC Session Controller network channels provide a **SourcePorts** attribute that you can use to configure one or more static ports that a server uses for originating UDP packets.

You can identify the ephemeral port currently used by the WebRTC Session Controller by examining the server log file. A log entry appears as follows:

```
<Nov 30, 2005 12:00:00 AM PDT> <Notice> <WebLogicServer> <BEA-000202> <Thread "SIP
Message Processor (Transport UDP)" listening on port 35993.>
```

Caution: Oracle does not recommend using the SourcePorts custom property in most configurations because it degrades performance. Configure the property only in cases where you must specify the exact ports that WebRTC Session Controller uses to originate UDP packets.

To use a static port for outgoing UDP datagrams, first disable use of the ephemeral port by specifying the following server start-up option:

```
-Dwlss.udp.listen.on.ephemeral=false
```

To configure the SourcePorts property, use a JMX client such as WLST or directly modify a network channel configuration in **config.xml** to include the custom property. SourcePorts defines an array of port numbers or port number ranges. Do not include spaces in the SourcePorts definition; use only port numbers, hyphens ("-") to designate ranges of ports, and commas (",") to separate ranges or individual ports. See Example 9–2 for an example configuration.

Example 9–2 Static Port Configuration for Outgoing UDP Packets

```
<network-access-point>
  <name>sip</name>
  <protocol>sip</protocol>
  <listen-port>5060</listen-port>
  <public-port>5060</public-port>
  <http-enabled-for-this-protocol>false</http-enabled-for-this-protocol>
  <tunneling-enabled>false</tunneling-enabled>
  <outbound-enabled>true</outbound-enabled>
  <enabled>true</enabled>
  <two-way-ssl-enabled>false</two-way-ssl-enabled>
  <client-certificate-enforced>false</client-certificate-enforced>
  <custom-properties>SourcePorts=5060</custom-properties>
</network-access-point>
```

Configuring Unique Listen Address Attributes for SIP Data Tier Replicas

Each replica in the SIP data tier must bind to a unique Listen Address attribute (a unique DNS name or IP address) to contact one another as peers. Follow these instructions for each replica to assign a unique Listen Address:

1. Access the Administration Console for the WebRTC Session Controller domain.
2. Select **Environment**, then select **Servers** from the left pane.
3. In the right pane, select the name of the server to configure.
4. Select **Configuration**, then select the **General** tab.
5. Enter a unique DNS name or IP address in the Listen Address field.
6. Click Save.

10

Configuring Server Failure Detection

This chapter describes how to use the Oracle Communications WebRTC Session Controller "echo server" process to improve SIP data tier failover performance when a server becomes physically disconnected from the network.

Overview of Failover Detection

In a production system, engine tier servers continually access SIP data tier replicas to retrieve and write call state data. The WebRTC Session Controller architecture depends on engine tier nodes to detect when a SIP data tier server has failed or become disconnected. When an engine cannot access or write call state data because a replica is unavailable, the engine connects to another replica in the same partition and reports the offline server. The replica updates the current view of the SIP data tier to account for the offline server, and other engines are then notified of the updated view as they access and retrieve call state data.

By default, an engine tier server uses its Remote Method Invocation (RMI) connection to the replica to determine if the replica has failed or become disconnected. The algorithms used to determine a failure of an RMI connection are reliable, but ultimately they depend on the TCP protocol's retransmission timers to diagnose a disconnection (for example, if the network cable to the replica is removed). Because the TCP retransmission timer generally lasts a full minute or longer, WebRTC Session Controller provides an alternate method of detecting failures that can diagnose a disconnected replica in a matter of a few seconds.

WlssEchoServer Failure Detection

WlssEchoServer is a separate process that you can run on the same server hardware as a SIP data tier replica. WlssEchoServer provides a simple UDP echo service to engine tier nodes used for determining when a SIP data tier server goes offline. The algorithm for detecting failures with WlssEchoServer is as follows:

1. For all normal traffic, engine tier servers communicate with SIP data tier replicas using TCP. TCP is used as the basic transport between the engine tier and SIP data tier regardless of whether WlssEchoServer is used.
2. Engine tier servers send a periodic heartbeat message to each configured WlssEchoServer over UDP. During normal operation, WlssEchoServer responds to the heartbeats so that the connection between the engine node and replica is verified.
3. Should there be a complete failure of the SIP data tier stack, or the network cable is disconnected, the heartbeat messages are not returned to the engine node. In this

case, the engine node can mark the replica as being offline *without* having to wait for the normal TCP connection timeout.

4. After identifying the offline server, the engine node reports the failure to an available SIP data tier replica, and the SIP data tier view is updated as described in the previous section.

Also, should a SIP data tier server notice that its local WlssEchoServer process has died, it automatically shuts down. This behavior ensures even quicker failover because avoids the time it takes engine nodes to notice and report the failure as described in "Overview of Failover Detection".

You can configure the heartbeat mechanism on engine tier servers to increase the performance of failover detection as necessary. You can also configure the listen port and log file that WlssEchoServer uses on SIP data tier servers.

Forced Shutdown for Failed Replicas

If any engine tier server cannot communicate with a particular replica, the engine access another, available replica in the SIP data tier to report the offline server. The replica updates its view of the affected partition to remove the offline server. The updated view is then distributed to all engine tier servers that later access the partition. Propagating the view in this manner helps to ensure that engine servers do not attempt to access the offline replica.

The replica that updates the view also issues a one-time request to the offline replica to ask it to shut down. This is done to try to shut-down running replica servers that cannot be accessed by one or more engine servers due to a network outage. If an active replica can reach the replica marked as "offline," the offline replica shuts down.

WlssEchoServer Requirements and Restrictions

Note: Using WlssEchoServer is not required in all WebRTC Session Controller installations. Enable the echo server only when your system requires detection of a network or replica failure faster than the configured TCP timeout interval.

Observe the following requirements and restrictions when using WlssEchoServer to detect replica failures:

- If you use the heartbeat mechanism to detect failures, you must ensure that the WlssEchoServer process is always running on each replica server. If the WlssEchoServer process fails or is stopped, the replica will be treated as being "offline" even if the server process is unaffected.
- The WlssEchoServer listens on all IP addresses available on the server.
- WlssEchoServer requires a dedicated port number to listen for heartbeat messages.

Starting WlssEchoServer on SIP Data Tier Server Machines

WlssEchoServer is a Java program that you can start directly from a shell or command prompt. The basic syntax for starting WlssEchoServer is:

```
java -classpath WLSS_HOME/server/lib/wlssechosvr.jar options
com.bea.wcp.util.WlssEchoServer
```

Where *WLSS_HOME* is the path to the WebLogic Server SIP directory and *options* may include one of the options described in Table 10–1.

Table 10–1 WlssEchoServer Options

Option	Description
-Dwlss.ha.echoserver.ipaddress	Specifies the IP address on which the WlssEchoServer instance listens for heartbeat messages. If you do not specify an IP address, the instance listens on any available IP address (0.0.0.0).
-Dwlss.ha.echoserver.port	Specifies the port number used to listen for heartbeat messages. Ensure that the port number you specify is not used by any other process on the server. By default WlssEchoServer uses port 6734.
-Dwlss.ha.echoserver.logfile	Specifies the log file location and name. By default, WebLogic writes log messages to **./echo_servertime.log** where time is the time expressed in milliseconds.

Oracle recommends that you include the command to start WlssEchoServer in the same script you use to start each WebRTC Session Controller SIP data tier instance. If you use the **startManagedWebLogic.sh** script to start an engine or SIP data tier server instance, add a command to start WlssEchoServer before the final command used to start the server. For example, change the lines:

```
"$JAVA_HOME/bin/java" ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}     \
  -Dweblogic.Name=${SERVER_NAME}                                  \
  -Dweblogic.management.username=${WLS_USER}                      \
  -Dweblogic.management.password=${WLS_PW}                        \
  -Dweblogic.management.server=${ADMIN_URL}                       \
  -Djava.security.policy="${WL_HOME}/server/lib/weblogic.policy" \
   weblogic.Server
```

to read:

```
"$JAVA_HOME/bin/java" -classpath WLSS_HOME/server/lib/wlssechosvr.jar    \
  -Dwlss.ha.echoserver.ipaddress=192.168.1.4                    \
  -Dwlss.ha.echoserver.port=6734 com.bea.wcp.util.WlssEchoServer &
"$JAVA_HOME/bin/java" ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}     \
  -Dweblogic.Name=${SERVER_NAME}                                  \
  -Dweblogic.management.username=${WLS_USER}                      \
  -Dweblogic.management.password=${WLS_PW}                        \
  -Dweblogic.management.server=${ADMIN_URL}                       \
  -Djava.security.policy="${WL_HOME}/server/lib/weblogic.policy" \
   weblogic.Server
```

Enabling and Configuring the Heartbeat Mechanism on Servers

To enable the WlssEchoServer heartbeat mechanism, you must include the **-Dreplica.host.monitor.enabled** JVM argument in the command you use to start all engine and SIP data tier servers. Oracle recommends adding this option directly to the script used to start Managed Servers in your system. For example, in the **startManagedWebLogic.sh** script, change the line:

```
# JAVA_OPTIONS="-Dweblogic.attribute=value -Djava.attribute=value"
```

to read:

```
JAVA_OPTIONS="-Dreplica.host.monitor.enabled=true"
```

Several additional JVM options configure the functioning of the heartbeat mechanism. Table 10–2 describes the options used to configure failure detection.

Table 10–2 WlssEchoServer Options

Option	Description
`-Dreplica.host.monitor.enabled`	This system property is required on both engine and SIP data tier servers to enable the heartbeat mechanism.
`-Dwlss.ha.heartbeat.interval`	Specifies the number of milliseconds between heartbeat messages. By default heartbeats are sent every 1,000 milliseconds.
`-Dwlss.ha.heartbeat.count`	Specifies the number of consecutive, missed heartbeats that are permitted before a replica is determined to be offline. By default, a replica is marked offline if the `WlssEchoServer` process on the server fails to respond to 3 heartbeat messages.
`-Dwlss.ha.heartbeat.SoTimeout`	Specifies the UDP socket timeout value.

11

Using the Engine Tier Cache

This chapter describes how to enable the Oracle Communications WebRTC Session Controller Signaling Engine tier cache for improved performance with SIP-aware load balancers.

Overview of Engine Tier Caching

The default WebRTC Session Controller Signaling Engine cluster is stateless. A separate SIP data tier cluster manages call state data in one or more partitions, and engine tier servers fetch and write data in the SIP data tier as necessary. Engines can write call state data to multiple replicas in each partition to provide automatic failover should a SIP data tier replica going offline.

WebRTC Session Controller also provides the option for engine tier servers to cache a portion of the call state data locally and in the SIP data tier. When a local cache is used, an engine tier server first checks its local cache for existing call state data. If the cache contains the required data, and the local copy of the data is up-to-date (compared to the SIP data tier copy), the engine locks the call state in the SIP data tier but reads directly from its cache. This improves response time performance for the request, because the engine does not have to retrieve the call state data from a SIP data tier server.

The engine tier cache stores only the call state data that has been most recently used by engine tier servers. Call state data is moved into an engine's local cache as necessary to respond to client requests or to refresh out-of-date data. If the cache is full when a new call state must be written to the cache, the least-recently accessed call state entry is first removed from the cache. The size of the engine tier cache is not configurable.

Using a local cache is most beneficial when a SIP-aware load balancer manages requests to the engine tier cluster. With a SIP-aware load balancer, all of the requests for an established call are directed to the same engine tier server, which improves the effectiveness of the cache. If you do not use a SIP-aware load balancer, the effectiveness of the cache is limited, because subsequent requests for the same call may be distributed to different engine tier severs (having different cache contents).

Configuring Engine Tier Caching

By default, engine tier caching is enabled. To disable partial caching of call state data in the engine tier, specify the **engine-call-state-cache-enabled** element in **sipserver.xml**:

```
<engine-call-state-cache-enabled>false</engine-call-state-cache-enabled>
```

When enabled, the cache size is fixed at a maximum of 250 call states. The size of the engine tier cache is not configurable.

Monitoring and Tuning Cache Performance

The **SipPerformanceRuntime** MBean monitors the behavior of the engine tier cache. Table 11–1 describes the MBean attributes.

Table 11–1 SipPerformanceRuntime Attribute Summary

Attribute	Description
`cacheRequests`	Tracks the total number of requests for session data items.
`cacheHits`	The server increments this attribute each time a request for session data results in a version of that data being found in the engine tier server's local cache. This counter is incremented even if the cached data is out-of-date and requires updating with data from the SIP data tier.
`cacheValidHits`	This attribute is incremented each time a request for session data is fully satisfied by a cached version of the data.

When enabled, the size of the cache is fixed at 250 call states. Because the cache consumes memory, you may need to modify the JVM settings used to run engine tier servers to meet your performance goals. Cached call states are maintained in the tenured store of the garbage collector. Try reducing the fixed **NewSize** value when the cache is enabled (for example, **-XX:MaxNewSize=32m -XX:NewSize=32m**). The actual value depends on the call state size used by applications and the size of the applications themselves.

12

Configuring Coherence

This chapter describes the implementation and configuration of Oracle Coherence in Oracle WebRTC Session Controller.

Overview of WebRTC Session Controller Coherence Implementation

WebRTC Session Controller includes an implementation of Coherence used for all engine tier internal clusterwide communication and state management. The Domain Creation Wizard creates a default Coherence cluster for managing WebRTC Session Controller information automatically when setting up new domains. The default cluster include the servers included in the engine tier and the administrative server in your environment.

Configuring Coherence

You configure the WebRTC Session Controller Coherence implementation using the Oracle WebLogic Administration Console. See the chapter on `"Configuring and Managing Coherence Clusters"` in *Administering Clusters for Oracle WebLogic Server* for more information on the parameters that can be set in the Administration Console.

To configure the default Coherence cluster installed with WebRTC Session Controller:

1. Log in to the Administration Console for the WebRTC Session Controller administration server.
2. In the **Domain Structure** tree, expand **Environment**.
3. Select **Coherence Clusters**.
4. In the **Coherence Clusters** table, select **defaultCoherenceCluster**.
5. Configure the parameters for the Coherence cluster as needed.
6. Click **Save**.

Each engine tier server and the administration server acts as a managed Coherence server. See `"Configuring Managed Coherence Servers"` in *Administering Clusters for Oracle WebLogic Server* for more information about managed Coherence servers.

To configure Coherence settings for individual engine tier servers and the administration server:

1. Log in to the Administration Console for the WebRTC Session Controller administration server.
2. In the **Domain Structure** tree, expand **Environment**.
3. Select **Servers**.

The Administration Console displays a list of servers included in your WebRTC Session Controller installation.

4. From the **Servers** table, select an engine tier or the administration server you want to configure Coherence settings for.
5. In the **Configuration** tab, select **Coherence**.
6. Configure the Coherence parameters for the server.
7. Click **Save**.

13

Upgrading Production WebRTC Session Controller Software

This chapter provides general instructions for upgrading Oracle Communications WebRTC Session Controller software to a new Service Pack release. The service pack may contain additional tools or instructions for performing the upgrade which may supersede the information in this chapter. Refer to those instructions if they are available.

Overview of System Upgrades

Because a typical production WebRTC Session Controller installation uses multiple server instances, upgrading the WebRTC Session Controller software requires that you follow very specific practices. These practices ensure that:

- Existing clients of deployed SIP Servlets are not interrupted or lost during the upgrade procedure.
- The upgrade procedure can be "rolled back" to a previous state if any problems occur.

The sections that follow describe how to use a configured load balancer to perform a "live" upgrade of the WebRTC Session Controller software on a production installation.

When upgrading the WebRTC Session Controller software (for example, in response to a Service Pack), a new engine tier cluster is created to host newly-upgraded engine tier instances. One-by-one, servers in the engine tier are shut-down, upgraded, and then restarted in the new target cluster. While servers are being upgraded, WebRTC Session Controller automatically forwards requests from one engine tier cluster to the other as necessary to ensure that SIP data tier requests are always initiated by a compatible engine tier server. After all servers have been upgraded, the older cluster is removed and no longer used. After upgrading the engine tier cluster, servers in the SIP data tier may also be upgraded, one-by-one.

Requirements for Upgrading a Production System

To upgrade a production WebRTC Session Controller installation you need:

- A compatible load balancer product and administrator privileges for reconfiguring the load balancer virtual IP addresses and pools.
- Adequate disk space on the Administration Server system and on each Managed Server system for installing a copy of the new WebRTC Session Controller software.

- Privileges for modifying configuration files on the WebRTC Session Controller Administration Server system.
- Privileges for shutting down and starting up individual Managed Server instances.
- Three or more replicas in each partition of the SIP data tier, to upgrade the WebRTC Session Controller software to a new version. With fewer than three replicas in each partition, it is not possible to safely upgrade a production SIP data tier deployment as no backup replica would be available during the upgrade procedure.

Caution: Before modifying any production installation, thoroughly test your proposed changes in a controlled, "stage" environment to ensure software compatibility and verify expected behavior.

Upgrading to a New Version of WebRTC Session Controller

Follow these steps to upgrade a production installation of WebRTC Session Controller to a newer Service Pack version of the WebRTC Session Controller software. These instructions upgrade both the SIP Servlet container implementation and the SIP data tier replication and failover implementation.

The steps for performing a software upgrade are divided into several high-level procedures:

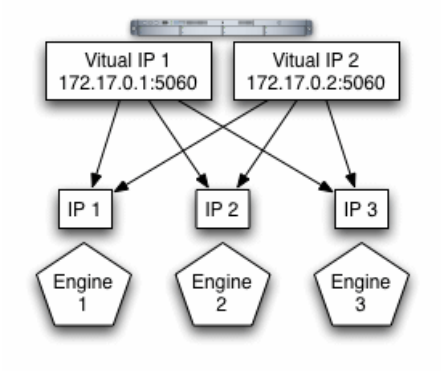
1. Configure the Load Balancer: Define a new, internal Virtual IP address for the new engine tier cluster you will configure.
2. Configure the New Engine Tier Cluster: Create and configure a new, empty engine tier cluster that will host upgraded engine tier servers and your converged applications.
3. Define the Cluster-to-Load Balancer Mapping: Modify the SIP Servlet container configuration to indicate the virtual IP address of each engine tier cluster.
4. Duplicate the SIP Servlet Configuration: Copy the active **sipserver.xml** configuration file to duplicate your production container configuration.
5. Upgrade Engine Tier Servers and Target Applications to the New Cluster: Shut down individual engine tier server instances, restarting them in the new engine tier cluster.
6. Upgrade SIP Data Tier Servers: Shut down individual SIP data tier servers, restarting them with the new SIP data tier software implementation.

The sections that follow describe each procedure.

Configure the Load Balancer

Begin the software upgrade procedure by defining a new internal virtual IP address for the new engine tier cluster you will create in "Configure the New Engine Tier Cluster". The individual server IP addresses (the pool definition) for the new virtual IP address should be identical to the pool definition of your currently-active engine tier cluster. Figure 13–1 shows a sample configuration for a cluster having three engine tier server instances; both virtual IP addresses define the same servers.

Figure 13–1 Virtual IP Address Configuration for Parallel Clusters



See your load balancer documentation for more information about defining virtual IP addresses.

In the next section, you will configure the WebRTC Session Controller domain to identify the virtual IP addresses that map to each engine tier cluster. WebRTC Session Controller uses this mapping during the upgrade procedure to automatically forward requests to the appropriate "version" of the cluster. This ensures that SIP data tier requests always originate from a compatible version of the WebRTC Session Controller engine tier.

Configure the New Engine Tier Cluster

Follow these steps to create an Engine Tier cluster to host upgraded containers, and to configure both clusters in preparation for a software upgrade:

1. On the Administration Server system, install the new WebRTC Session Controller software into a new *ORACLE_HOME***Middleware** home directory. The steps that follow refer to **c:\Oraclenew** as the home directory in which the new software was installed. **c:\Oracle** refers to the software implementation that is being upgraded.
2. Log in to the Administration Console for the active WebRTC Session Controller domain.
3. In the Administration Console, create an empty engine tier cluster for hosting the upgraded engine tier servers:
   a. In the left pane, select the **Environment** node, and then select the **Clusters** node.
   b. Click **New** to configure a new cluster.
   c. Enter a name for the new cluster.
   d. Select Unicast or Multicast for the **Messaging Mode**.
   e. Enter a **Unicast Broadcast Channel**, or **Multicast Address** and **Multicast Port** number for the cluster.
   f. Click **OK** to create the cluster.
4. Proceed to "Define the Cluster-to-Load Balancer Mapping".

Define the Cluster-to-Load Balancer Mapping

In this procedure, you configure the cluster-to-load balancer mapping to define the internal, virtual IP address that is assigned to the older and newer engine tier clusters.

To define the cluster-to-load balancer mapping:

1. Log in to the Administration Console for the active WebRTC Session Controller domain.
2. In the Administration Console, create a load balancer map entry for the new cluster you created:
   a. In the left pane, select the **SipServer** node.
   b. Select **Configuration**, then select the **Loadbalancer Map** tab.
   c. Click **New** to create a mapping.
   d. Enter the **Cluster Name** and **SIP URI** of the new cluster you created in Configure the New Engine Tier Cluster.
   e. Click **Finish** to create the new mapping.
3. Repeat Step 2 to enter a mapping for the older cluster. For example, create an entry for "EngineCluster" and "sip:172.17.0.1:5060".

   You must include a mapping for both the older and the newer engine tier cluster. A mapping consists of the internal virtual IP address of the cluster configured on the load balancer and the cluster name defined in the WebRTC Session Controller domain. Example 13–1 shows an entry in the **sipserver.xml** file for the sample clusters described earlier.

Example 13–1 Sample cluster-loadbalancer-map Definition

```
<sip-server>
...
  <cluster-loadbalancer-map>
      <cluster-name>EngineCluster</cluster-name>
      <sip-uri>sip:172.17.0.1:5060</sip-uri>
   </cluster-loadbalancer-map>
   <cluster-loadbalancer-map>
      <cluster-name>NewEngineCluster</cluster-name>
      <sip-uri>sip:172.17.0.2:5060</sip-uri>
   </cluster-loadbalancer-map>
</sip-server>
```

Duplicate the SIP Servlet and WebRTC Session Controller Configurations

Before upgrading individual engine tier servers, you must ensure that the SIP Servlet container configuration, SIP data tier configuration, and the WebRTC Session Controller configuration in the new engine tier cluster matches your current production configuration.

To duplicate the SIP Servlet container configuration, copy your production **sipserver.xml** configuration file to the new installation. For example:

```
cp c:\Oracle\Middleware\user_projects\domains\mydomain\config\custom\*.xml
c:\Oraclenew\Middleware\user_projects\domains\mydomain\config\custom
```

To duplicate the WebRTC Session Controller configuration, copy your production configuration files to the new installation. For example:

```
cp c:\Oracle\Middleware\user_projects\domains\mydomain\config\wsc\*.xml
c:\Oraclenew\Middleware\user_projects\domains\mydomain\config
```

As engine tier servers are restarted in the new engine tier cluster in the next procedure, they will have the same configuration but will use the new container implementation.

Upgrade Engine Tier Servers and Target Applications to the New Cluster

To upgrade individual engine tier servers, you gracefully shut each server down, change its cluster membership, and then restart it. Follow these steps:

1. Access the Administration Console for your production domain.
2. Select the first running engine tier server that you want to upgrade:
   a. Select **Environment**, then select the **Servers** tab in the left pane.
   b. Select the **Control** tab in the right pane.
   c. Select the name of the server you want to upgrade.
3. Select **Shutdown**, then select **When work completes from the table**.

   The server remains active while clients are still accessing the server, but no new connection requests are accepted. After all existing client connections have ended or timed out, the server shuts down. Other server instances in the engine tier process client requests during the shutdown procedure.
4. Select **Environment**, then select the **Servers** tab in the left pane and verify that the Managed Server has shut down.
5. Change the stopped server's cluster membership so that it is a member of the new engine tier cluster:
   a. Expand the **Environment** node, and the select the **Clusters** tab in the left pane.
   b. Select the name of the active engine tier cluster.
   c. Select **Configuration**, and then select the **Servers** tab in the right pane.
   d. Select the check box next to the name of the stopped server, and click **Remove**.
   e. Click **Yes** to remove the server from the cluster.
   f. Next, expand the **Environment** node, then select the **Clusters** tab and select the newly-created engine tier cluster ("NewEngineCluster").
   g. Select **Configuration**, then select the **Servers** tab in the right pane.
   h. Click **Add** to add a new server.
   i. Select the name of the stopped server from the drop-down list.
   j. Click **Finish**.
6. Restart the stopped managed server to bring it up in the new engine tier cluster:
   a. Access the system on which the stopped engine tier server runs (for example, use a remote desktop on Windows, or secure shell (SSH) on Linux).
   b. Use the available Managed Server start script (startManagedWebLogic.cmd or startManagedWebLogic.sh) to start the Managed Server. For example:

      ```
      startManagedWebLogic.cmd engine-server1 t3://adminhost:7001
      ```
7. In the Administration Console, select **Environment**, then select the **Servers** node and verify that the Managed Server has started.
8. Repeat these steps to upgrade the remaining engine tier servers.

At this point, all running Managed Servers are using the new SIP Container implementation and are hosting your production SIP Servlets with the same SIP Servlet container settings as your old configuration.

Applying Software Patches and Updates

Oracle may occasionally release software patches and updates to address known bugs and limitations in the WebRTC Session Controller software. To update WebRTC Session Controller, you apply patches using the Oracle OPatch utility.

See "Patching with OPatch" in Oracle Fusion Middleware Documentation for more information on how to update your WebRTC Session Controller installation.

Part II

Monitoring and Troubleshooting

This part provides information on operating and maintaining Oracle Communications WebRTC Session Controller. It includes information on starting and stopping servers, logging, diagnostics, SNMP traps, upgrading WebRTC Session Controller software and deployed SIP applications, and avoiding and recovering from server failure.

This part contains the following chapters:

- Logging SIP Requests and Responses
- Avoiding and Recovering From Server Failures
- Tuning JVM Garbage Collection for Production Deployments
- Avoiding JVM Delays Caused By Random Number Generation

14

Logging SIP Requests and Responses

This chapter describes how to configure and manage logging for SIP requests and responses that Oracle Communications WebRTC Session Controller processes.

Overview of SIP Logging

WebRTC Session Controller enables you to perform Protocol Data Unit (PDU) logging for the SIP requests and responses it processes. Logged SIP messages are placed either in the domain-wide log file for WebRTC Session Controller, or in the log files for individual Managed Server instances. Because SIP messages share the same log files as WebRTC Session Controller instances, you can use advanced server logging features such as log rotation, domain log filtering, and maximum log size configuration when managing logged SIP messages.

Administrators configure SIP PDU logging by defining one or more SIP servlets using the `com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl` class. Logging criteria are then configured either as parameters to the defined servlet, or in separate XML files packaged with the application.

As SIP requests are processed or SIP responses generated, the logging servlet compares the message with the filtering patterns defined in a standalone XML configuration file or servlet parameter. WebRTC Session Controller writes SIP requests and responses that match the specified pattern to the log file along with the name of the logging servlet, the configured logging level, and other details. To avoid unnecessary pattern matching, the servlet marks new SIP Sessions when an initial pattern is matched and then logs subsequent requests and responses for that session automatically.

Logging criteria are defined either directly in **sip.xml** as parameters to a logging servlet, or in external XML configuration files. See "Specifying the Criteria for Logging Messages".

Note: Engineers can implement PDU logging functionality in their servlets either by creating a delegate with the `TraceMessageListenerFactory` in the servlet's `init()` method, or by using the tracing class in deployed Java applications. Using the delegate enables you to perform custom logging or manipulate incoming SIP messages using the default trace message listener implementation. See "Adding Tracing Functionality to SIP Servlet Code" for an example of using the factory in a servlet's `init()` method.

Defining Logging Servlets in sip.xml

Create logging servlets for SIP messages by defining servlets having the implementation class `com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl` in the **sip.xml** file. The definition for a sample `msgTraceLogger` is shown in Example 14–1.

Example 14–1 Sample Logging Servlet

```
<servlet>
    <servlet-name>msgTraceLogger</servlet-name>

<servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl</s
ervlet-class>
    <init-param>
      <param-name>domain</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>level</param-name>
      <param-value>full</param-value>
    </init-param>
    <load-on-startup/>
  </servlet>
```

Configuring the Logging Level and Destination

Logging attributes such as the level of logging detail and the destination log file for SIP messages are passed as initialization parameters to the logging servlet. Table 14–1, " Pattern-matching Variables and Sample Values" lists the parameters and parameter values that you can specify as **init-param** entries. Example 14–1 shows the sample init-param entries for a servlet that logs full SIP message information to the domain log file.

Specifying the Criteria for Logging Messages

The criteria for selecting SIP messages to log can be defined either in XML files that are packaged with the logging servlet's application, or as initialization parameters in the servlet's **sip.xml** deployment descriptor. The sections that follow describe each method.

Using XML Documents to Specify Logging Criteria

If you do not specify logging criteria as an initialization parameter to the logging servlet, the servlet looks for logging criteria in a pair of XML descriptor files in the top level of the logging application. These descriptor files, named **request-pattern.xml** and **response-pattern.xml**, define patterns that WebRTC Session Controller uses for selecting SIP requests and responses to place in the log file.

> **Note:** By default WebRTC Session Controller logs both requests and responses. If you do not want to log responses, you must define a **response-pattern.xml** file with empty matching criteria.

A typical pattern definition defines a condition for matching a particular value in a SIP message header. For example, the sample **response-pattern.xml** used by the

msgTraceLogger servlet matches all MESSAGE requests. The contents of this descriptor are shown in Example 14–2.

Example 14–2 Sample response-pattern.xml for msgTraceLogger Servlet

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pattern
   PUBLIC "Registration//Organization//Type Label//Definition Language"
   "trace-pattern.dtd">
<pattern>
  <equal>
    <var>response.method</var>
    <value>MESSAGE</value>
  </equal>
</pattern>
```

See "trace-pattern.dtd Reference" for descriptions of additional operators and conditions used for matching SIP messages. Most conditions, such as the **equal** condition shown in Example 14–2, require a variable (**var** element) that identifies the portion of the SIP message to evaluate. Table 14–1 lists some common variables and sample values. For additional variable names and examples, see *Section 16: Mapping Requests to Servlets* in the SIP servlet API 1.1 specification (http://jcp.org/en/jsr/detail?id=289); WebRTC Session Controller enables mapping of both request and response variables to logging servlets.

Table 14–1 Pattern-matching Variables and Sample Values

Variable	Sample Values
request.method, response.method	MESSAGE, INVITE, ACK, BYE, CANCEL
request.uri.user, response.uri.user	guest, admin, joe
request.to.host, response.to.host	server.mydomain.com

Both **request-pattern.xml** and **response-pattern.xml** use the same Document Type Definition (DTD). See "trace-pattern.dtd Reference" for more information.

Using Servlet Parameters to Specify Logging Criteria

Pattern-matching criteria can also be specified as initialization parameters to the logging servlet, rather than as separate XML documents. The parameter names used to specify matching criteria are **request-pattern-string** and **response-pattern-string**. They are defined along with the logging level and destination as described in "Configuring the Logging Level and Destination".

The value of each pattern-matching parameter must consist of a valid XML document that adheres to the DTD for standalone pattern definition documents (see "Using XML Documents to Specify Logging Criteria"). Because the XML documents that define the patterns and values must not be parsed as part of the **sip.xml** descriptor, you must enclose the contents within the **CDATA** tag. Example 14–3 shows the full **sip.xml** entry for the sample logging servlet, **invTraceLogger**. The final two init-param elements specify that the servlet log only **INVITE** request methods and **OPTIONS** response methods.

Example 14–3 Logging Criteria Specified as init-param Elements

```
<servlet>
      <servlet-name>invTraceLogger</servlet-name>
```

```
<servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl</s
ervlet-class>
      <init-param>
        <param-name>domain</param-name>
        <param-value>true</param-value>
      </init-param>
      <init-param>
        <param-name>level</param-name>
        <param-value>full</param-value>
      </init-param>
      <init-param>
        <param-name>request-pattern-string</param-name>
        <param-value>
            <![CDATA[
                <?xml version="1.0" encoding="UTF-8"?>
                <!DOCTYPE pattern
                   PUBLIC "Registration//Organization//Type Label//Definition
Language"
                   "trace-pattern.dtd">
                <pattern>
                  <equal>
                    <var>request.method</var>
                    <value>INVITE</value>
                  </equal>
                </pattern>
            ]]>
        </param-value>
      </init-param>
      <init-param>
        <param-name>response-pattern-string</param-name>
        <param-value>
            <![CDATA[
                <?xml version="1.0" encoding="UTF-8"?>
                <!DOCTYPE pattern
                   PUBLIC "Registration//Organization//Type Label//Definition
Language"
                   "trace-pattern.dtd">
                <pattern>
                  <equal>
                    <var>response.method</var>
                    <value>OPTIONS</value>
                  </equal>
                </pattern>
            ]]>
        </param-value>
      </init-param>
      <load-on-startup/>
  </servlet>
```

Specifying Content Types for Unencrypted Logging

By default WebRTC Session Controller uses String format (UTF-8 encoding) to log the content of SIP messages having a text or application/sdp Content-Type value. For all other Content-Type values, WebRTC Session Controller attempts to log the message content using the character set specified in the **charset** parameter of the message, if one is specified. If no charset parameter is specified, or if the charset value is invalid or unsupported, WebRTC Session Controller uses Base-64 encoding to encrypt the message content before logging the message.

To avoid encrypting the content of messages under these circumstances, specify a list of String-representable Content-Type values using the **string-rep** element in **sipserver.xml**. The string-rep element can contain one or more **content-type** elements to match. If a logged message matches one of the configured content-type elements, WebRTC Session Controller logs the content in String format using UTF-8 encoding, regardless of whether a charset parameter is included.

> **Note:** You do not need to specify text/* or application/sdp content types as these are logged in String format by default.

Example 14–4 shows a sample **message-debug** configuration that logs String content for three additional Content-Type values, in addition to text/* and application/sdp content.

Example 14–4 Logging String Content for Additional Content Types

```
<message-debug>
  <level>full</level>
  <string-rep>
    <content-type>application/msml+xml</content-type>
    <content-type>application/media_control+xml</content-type>
    <content-type>application/media_control</content-type>
  </string-rep>
</message-debug>
```

Enabling Log Rotation and Viewing Log Files

The WebRTC Session Controller logging infrastructure enables you to automatically write to a new log file when the existing log file reaches a specified size. You can also view log contents using the Administration Console or configure additional server-level events that are written to the log.

trace-pattern.dtd Reference

trace-pattern.dtd defines the required contents of the **request-pattern.xml** and **response-pattern.xml**, documents and the values for the **request-pattern-string** and **response-pattern-string** servlet **init-param** variables.

Example 14–5 trace-pattern.dtd

```
<!--
The different types of conditions supported.
- >

<!ENTITY % condition "and | or | not |
                     equal | contains | exists | subdomain-of">

<!--
A pattern is a condition: a predicate over the set of SIP requests.
- >

<!ELEMENT pattern (%condition;)>

<!--
An "and" condition is true if and only if all its constituent conditions
are true.
```

```
- >

<!ELEMENT and (%condition;)+>

<!--
An "or" condition is true if at least one of its constituent conditions
is true.
- >

<!ELEMENT or (%condition;)+>

<!--
Negates the value of the contained condition.
- >

<!ELEMENT not (%condition;)>

<!--
True if the value of the variable equals the specified literal value.
- >

<!ELEMENT equal (var, value)>

<!--
True if the value of the variable contains the specified literal value.
- >

<!ELEMENT contains (var, value)>

<!--
True if the specified variable exists.
- >

<!ELEMENT exists (var)>

<!--
- >

<!ELEMENT subdomain-of (var, value)>

<!--
Specifies a variable. Example:
  <var>request.uri.user</var>
- >

<!ELEMENT var (#PCDATA)>

<!--
Specifies a literal string value that is used to specify rules.
- >

<!ELEMENT value (#PCDATA)>

<!--
Specifies whether the "equal" test is case sensitive or not.
- >

<!ATTLIST equal ignore-case (true|false) "false">

<!--
```

```
Specifies whether the "contains" test is case sensitive or not.
- >

<!ATTLIST contains ignore-case (true|false) "false">

<!--
The ID mechanism is to allow tools to easily make tool-specific
references to the elements of the deployment descriptor. This allows
tools that produce additional deployment information (i.e information
beyond the standard deployment descriptor information) to store the
non-standard information in a separate file, and easily refer from
these tools-specific files to the information in the standard sip-app
deployment descriptor.
- >

<!ATTLIST pattern id ID #IMPLIED>
<!ATTLIST and id ID #IMPLIED>
<!ATTLIST or id ID #IMPLIED>
<!ATTLIST not id ID #IMPLIED>
<!ATTLIST equal id ID #IMPLIED>
<!ATTLIST contains id ID #IMPLIED>
<!ATTLIST exists id ID #IMPLIED>
<!ATTLIST subdomain-of id ID #IMPLIED>
<!ATTLIST var id ID #IMPLIED>
<!ATTLIST value id ID #IMPLIED>
```

Adding Tracing Functionality to SIP Servlet Code

Tracing functionality can be added to your own servlets or to Java code by using the **TraceMessageListenerFactory**. TraceMessageListenerFactory enables clients to reuse the default trace message listener implementation behaviors by creating an instance and then delegating to it. The factory implementation instance can be found in the servlet context for SIP servlets by looking up the value of the **TraceMessageListenerFactory.TRACE_MESSAGE_LISTENER_FACTORY** attribute.

> **Note:** Instances created by the factory are not registered with WebRTC Session Controller to receive callbacks upon SIP message arrival and departure.

To implement tracing in a servlet, you use the factory class to create a delegate in the servlet's **init()** method as shown in Example 14–6.

Example 14–6 Using the TraceMessageListenerFactory

```
public final class TraceMessageListenerImpl extends SipServlet implements
MessageListener {
  private MessageListener delegate;

  public void init() throws ServletException {
    ServletContext sc = (ServletContext) getServletContext();
    TraceMessageListenerFactory factory = (TraceMessageListenerFactory)
sc.getAttribute(TraceMessageListenerFactory.TRACE_MESSAGE_LISTENER_FACTORY);
    delegate = factory.createTraceMessageListener(getServletConfig());
  }
  public final void onRequest(SipServletRequest req, boolean incoming) {
    delegate.onRequest(req,incoming);
  }
```

```
  public final void onResponse(SipServletResponse resp, boolean incoming) {
    delegate.onResponse(resp,incoming);
  }
}
```

Order of Startup for Listeners and Logging Servlets

If you deploy both listeners and logging servlets, the listener classes are loaded first, followed by the servlets. Logging servlets are deployed in order according to the load order specified in their web application deployment descriptor.

15

Avoiding and Recovering From Server Failures

This chapter describes the Oracle Communications WebRTC Session Controller failure prevention and recovery features, and includes the configuration artifacts that are required to restore different portions of a WebRTC Session Controller domain.

Failure Prevention and Automatic Recovery Features

A variety of events can lead to the failure of a server instance. Often one failure condition leads to another. Loss of power, hardware malfunction, operating system malfunctions, network partitions, or unexpected application behavior may each contribute to the failure of a server instance.

WebRTC Session Controller uses a highly clustered architecture as the basis for minimizing the impact of failure events. However, even in a clustered environment it is important to prepare for a sound recovery process if an individual server fails.

WebRTC Session Controller, and the underlying WebLogic Server platform, provide many features that protect against server failures. In a production system, use all available features to ensure uninterrupted service.

Overload Protection

WebRTC Session Controller detects increases in system load that could affect the performance and stability of deployed SIP Servlets, and automatically throttles message processing at predefined load thresholds.

Using overload protection helps you avoid failures that could result from unanticipated levels of application traffic or resource utilization.

WebRTC Session Controller attempts to avoid failure when certain conditions occur:

- The rate at which SIP sessions are created reaches a configured value, or
- The size of the SIP timer and SIP request-processing execute queues reaches a configured length.

See "overload" in the chapter Chapter 18, "Engine Tier Configuration Reference (sipserver.xml)" for more information.

The underlying WebLogic Server platform also detects increases in system load that can affect deployed application performance and stability. WebLogic Server allows administrators to configure failure prevention actions that occur automatically at predefined load thresholds. Automatic overload protection helps you avoid failures that result from unanticipated levels of application traffic or resource utilization as indicated by:

- A workload manager's capacity being exceeded

- The HTTP session count increasing to a predefined threshold value
- Impending out of memory conditions

See "Avoiding and Managing Overload" in *Administering Server Environments for Oracle WebLogic Server* for more information.

Redundancy and Failover for Clustered Services

You can increase the reliability and availability of your applications by using multiple engine tier servers in a dedicated cluster and multiple SIP data tier servers (replicas) in a dedicated SIP data tier cluster.

Because engine tier clusters maintain no stateful information about applications, the failure of an engine tier server does not result in any data loss or dropped calls. Multiple replicas in a SIP data tier partition store redundant copies of call state information, and automatically failover to one another should a replica fail.

See *Oracle Communications WebRTC Session Controller Concepts* for more information.

Automatic Restart for Failed Server Instances

WebLogic Server self-health monitoring features improve the reliability and availability of server instances in a domain. Selected subsystems within each server instance monitor their health status based on criteria specific to the subsystem. (For example, the JMS subsystem monitors the condition of the JMS thread pool while the core server subsystem monitors default and user-defined execute queue statistics.) If an individual subsystem determines that it can no longer operate in a consistent and reliable manner, it registers its health state as failed with the host server.

Each WebLogic Server instance, in turn, checks the health state of its registered subsystems to determine its overall viability. If one or more of its critical subsystems have reached the FAILED state, the server instance marks its own health state FAILED to indicate that it cannot reliably host an application.

When used in combination with Node Manager, server self-health monitoring enables you to automatically restart servers that have failed. This improves the overall reliability of a domain, and requires no direct intervention from an administrator. For more information, see "Using Node Manager to Control Servers" in the Administering Node Manager for Oracle WebLogic Server.

Managed Server Independence Mode

Managed Servers maintain a local copy of the domain configuration. When a Managed Server starts, it contacts its Administration Server to retrieve any changes to the domain configuration that were made since the Managed Server was last shut down. If a Managed Server cannot connect to the Administration Server during startup, it can use its locally-cached configuration information—this is the configuration that was current at the time of the Managed Server's most recent shutdown. A Managed Server that starts without contacting its Administration Server to check for configuration updates is running in **Managed Server Independence (MSI)** mode. By default, MSI mode is enabled. See "Replicate domain config files for Managed Server Independence" in the *Administration Console Online Help* for more information.

Automatic Migration of Failed Managed Servers

When using Linux or UNIX operating systems, you can use WebLogic Server's server migration feature to automatically start a candidate (backup) server if a Network tier

server fails or becomes partitioned from the network. The server migration feature uses node manager, with the **wlsifconfig.sh** script, to automatically start candidate servers using a floating IP address. Candidate servers are started only if the primary server hosting a Network tier instance becomes unreachable. See the discussion on `"Whole Server Migration"` in *Administering Clusters for Oracle WebLogic Server* for more information about using the server migration feature.

Geographic Redundancy for Regional Site Failures

In addition to server-level redundancy and failover capabilities, WebRTC Session Controller enables you to configure peer sites to protect against catastrophic failures, such as power outages, that can affect an entire domain. This configuration enables you to failover from one geographical site to another, avoiding complete service outages.

Directory and File Backups for Failure Recovery

Recovery from the failure of a server instance requires access to the domain's configuration data. By default, the Administration Server stores a domain's primary configuration data in a file called *domain_home***/config/config.xml**, where *domain_home* is the root directory of the domain.

The primary configuration file may reference additional configuration files for specific WebLogic Server services, such as JDBC and JMS, and for WebRTC Session Controller services, such as SIP container properties and SIP data tier configuration. The configuration for specific services are stored in additional XML files in subdirectories of the *domain_home***/config** directory, such as *domain_home***/config/jms**, *domain_home***/config/jdbc**, and *domain_home***/config/custom** for WebRTC Session Controller configuration files.

The Administration Server can automatically archive multiple versions of the domain configuration (the entire **domain_home/config** directory). Use the configuration archives for system restoration in cases where accidental configuration changes need to be reversed. For example, if an administrator accidentally removes a configured resource, the prior configuration can be restored by using the last automated backup.

The Administration Server stores only a finite number of automated backups locally in *domain_home***/config**. For this reason, automated domain backups are limited in their ability to guard against data corruption, such as a failed hard disk. Automated backups also do not preserve certain configuration data that are required for full domain restoration, such as LDAP repository data and server start-up scripts. Oracle recommends that you also maintain multiple backup copies of the configuration and security offline, in a source control system.

This section describes file backups that WebRTC Session Controller performs automatically and manual backup procedures that an administrator should perform periodically.

Enabling Automatic Configuration Backups

Follow these steps to enable automatic domain configuration backups on the Administration Server for your domain:

1. Access the Administration Console for your domain.
2. In the left pane of the Administration Console, select the name of the domain.
3. In the right pane, click **Configuration**, and then select the **General** tab.

4. Select **Advanced** to display advanced options.
5. Select **Configuration Archive Enabled**.
6. In the **Archive Configuration Count** box, enter the maximum number of configuration file revisions to save.
7. Click **Save**.

When you enable configuration archiving, the Administration Server automatically creates a configuration JAR file archive. The JAR file contains a complete copy of the previous configuration (the complete contents of the *domain_home***config** directory). JAR file archive files are stored in the *domain_home***configArchive** directory. The files use the naming convention **config-number.jar**, where **number** is the sequential number of the archive.

When you save a change to a domain's configuration, the Administration Server saves the previous configuration in *domain_home***configArchive\config.xml#n**. Each time the Administration Server saves a file in the **configArchive** directory, it increments the value of the **#n** suffix, up to a configurable number of copies—5 by default. Thereafter, each time you change the domain configuration:

- The archived files are rotated so that the newest file has a suffix with the highest number,
- The previous archived files are renamed with a lower number, and
- The oldest file is deleted.

Be aware that configuration archives are stored locally within the domain directory, and they may be overwritten according to the maximum number of revisions you selected. For these reasons, you must also create your own off-line archives of the domain configuration, as described in "Storing the Domain Configuration Offline".

Storing the Domain Configuration Offline

Although automatic backups protect against accidental configuration changes, they do not protect against data loss caused by a failure of the hard disk that stores the domain configuration, or accidental deletion of the domain directory. To protect against these failures, you must also store a complete copy of the domain configuration offline, preferably in a source control system.

Oracle recommends storing a copy of the domain configuration at regular intervals. For example, backup a new revision of the configuration when:

- You first deploy the production system
- You add or remove deployed applications
- The configuration is tuned for performance
- Any other permanent change is made.

The domain configuration backup should contain the complete contents of the *domain_home*/**config** directory. For example, to make an offline archive of *mydomain* use the following command:

```
cd ~/ORACLE_HOME/Middleware/user_projects/domains/mydomain
tar cvf domain-backup-06-17-2007.jar config
```

Store the new archive in a source control system, preserving earlier versions should you need to restore the domain configuration to an earlier point in time.

Backing Up Server Start Scripts

In a WebRTC Session Controller deployment, the start scripts used to start engine and SIP data tier servers are generally customized to include domain-specific configuration information such as:

- JVM Garbage Collection parameters required to achieve throughput targets for SIP message processing (see "Modifying JVM Parameters in Server Start Scripts" in Chapter 16, "Tuning JVM Garbage Collection for Production Deployments." Different parameters (and therefore, different start scripts) are generally used to start engine and SIP data tier servers.
- Configuration parameters and startup information for the WebRTC Session Controller heartbeat mechanism. If you use the heartbeat mechanism, engine tier server start scripts should include startup options to enable and configure the heartbeat mechanism. SIP data tier server start scripts should include startup options to enable heartbeats and start the **WlssEchoServer** process.

Backup each distinct start script used to start engine tier, SIP data tier, or diameter relay servers in your domain.

Backing Up Logging Servlet Applications

If you use WebRTC Session Controller logging Servlets (see Chapter 14, "Logging SIP Requests and Responses") to perform regular logging or auditing of SIP messages, backup the complete application source files so that you can easily redeploy the applications should the staging server fail or the original deployment directory becomes corrupted.

Backing Up Security Data

The WebLogic Security service stores its configuration data **config.xml** file, and also in an LDAP repository and other files.

Backing Up the WebLogic LDAP Repository

The default Authentication, Authorization, Role Mapper, and Credential Mapper providers that are installed with WebRTC Session Controller store their data in an LDAP server. Each WebRTC Session Controller contains an embedded LDAP server. The Administration Server contains the master LDAP server, which is replicated on all Managed Servers. If any of your security realms use these installed providers, you should maintain an up-to-date backup of the following directory tree:

*domain_home***servers\AdminServer\data\ldap**

where *domain_home* is the domain's root directory and **servers\AdminServer\data\ldap** is the directory in which the Administration Server stores run-time and security data.

Each WebRTC Session Controller has an LDAP directory, but you only need to back up the LDAP data on the Administration Server—the master LDAP server replicates the LDAP data from each Managed Server when updates to security data are made. WebLogic security providers cannot modify security data while the domain's Administration Server is unavailable. The LDAP repositories on Managed Servers are replicas and cannot be modified.

The **ldap\ldapfiles** subdirectory contains the data files for the LDAP server. The files in this directory contain user, group, group membership, policies, and role information. Other subdirectories under the `ldap` directory contain LDAP server message logs and data about replicated LDAP servers.

Do not update the configuration of a security provider while a backup of LDAP data is in progress. If a change is made—for instance, if an administrator adds a user—while you are backing up the **ldap** directory tree, the backups in the **ldapfiles** subdirectory could become inconsistent. If this does occur, consistent, but potentially out-of-date, LDAP backups are available.

Once a day, a server suspends write operations and creates its own backup of the LDAP data. It archives this backup in a ZIP file below the **ldap\backup** directory and then resumes write operations. This backup is guaranteed to be consistent, but it might not contain the latest security data.

For information about configuring the LDAP backup, see the `"Back Up LDAP Repository"` section in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

Backing Up SerializedSystemIni.dat and Security Certificates

All servers create a file named **SerializedSystemIni.dat** and place it in the server's root directory. This file contains encrypted security data that must be present to start the server. You must back up this file.

If you configured a server to use SSL, also back up the security certificates and keys. The location of these files is user-configurable.

Backing Up Additional Operating System Configuration Files

Certain files maintained at the operating system level are also critical in helping you recover from system failures. Consider backing up the following information as necessary for your system:

- Load Balancer configuration scripts. For example, any automated scripts used to configure load balancer pools and virtual IP addresses for the engine tier cluster and NAT configuration settings.
- NTP client configuration scripts used to synchronize the system clocks of engine and SIP data tier servers.
- Host configuration files for each WebRTC Session Controller system (host names, virtual and real IP addresses for multi-homed machines, IP routing table information).

Restarting a Failed Administration Server

If an Administration Server fails, only configuration, deployment, and monitoring features are affected, but Managed Servers continue to operate and process client requests. Potential losses incurred due to an Administration Server failure include:

- Loss of in-progress management and deployment operations.
- Loss of ongoing logging functionality.
- Loss of SNMP trap generation for WebLogic Server instances (as opposed to WebRTC Session Controller instances). On Managed Servers, WebRTC Session Controller traps are generated even without the Administration Server.

To resume normal management activities, restart the failed Administration Server instance as soon as possible.

When you restart a failed Administration Server, no special steps are required. Start the Administration Server as you normally would.

If the Administration Server shuts down while Managed Servers continue to run, you do not need to restart the Managed Servers that are already running to recover management of the domain. The procedure for recovering management of an active domain depends upon whether you can restart the Administration Server on the same system it was running on when the domain was started.

Restarting an Administration Server on the Same System

If you restart the WebLogic Administration Server while Managed Servers continue to run, by default the Administration Server can discover the presence of the running Managed Servers.

> **Note:** Ensure that the startup command or startup script does not include `-Dweblogic.management.discover=false`, which disables an Administration Server from discovering its running Managed Servers.

The root directory for the domain contains a file, **running-managed-servers.xml**, which contains a list of the Managed Servers in the domain and describes their running state. When the Administration Server restarts, it checks this file to determine which Managed Servers were under its control before it stopped running.

When a Managed Server is gracefully or forcefully shut down, its status in **running-managed-servers.xml** is updated to "not-running." When an Administration Server restarts, it does not try to discover Managed Servers with the "not-running" status. A Managed Server that stops running because of a system malfunction, or that was stopped by killing the JVM or the command prompt (shell) in which it was running, will still have the status "running' in **running-managed-servers.xml**. The Administration Server will attempt to discover them, and will throw an exception when it determines that the Managed Server is no longer running.

Restarting the Administration Server does not cause Managed Servers to update the configuration of static attributes. **Static attributes** are those that a server refers to only during its startup process. Servers instances must be restarted to take account of changes to static configuration attributes. Discovery of the Managed Servers only enables the Administration Server to monitor the Managed Servers or make run-time changes to attributes configurable while a server is running (dynamic attributes).

Restarting an Administration Server on Another System

If a system malfunction prevents you from restarting the Administration Server on the same system, you can recover management of the running Managed Servers as follows:

1. Install the WebRTC Session Controller software on the new system (if this has not already been done).
2. Make your application files available to the new Administration Server by copying them from backups or by using a shared disk. Your application files should be available in the same relative location on the new file system as on the file system of the original Administration Server.
3. Make your configuration and security data available to the new administration system by copying them from backups or by using a shared disk. For more information, refer to "Storing the Domain Configuration Offline" and "Backing Up Security Data".
4. Restart the Administration Server on the new system.

Ensure that the startup command or startup script does not include `-Dweblogic.management.discover=false`, which disables an Administration Server from discovering its running Managed Servers.

When the Administration Server starts, it communicates with the Managed Servers and informs them that the Administration Server is now running on a different IP address.

Restarting Failed Managed Servers

If the system on which the failed Managed Server runs can contact the Administration Server for the domain, simply restart the Managed Server manually or automatically using Node Manager. You must configure Node Manager and the Managed Server to support automated restarts, as described in the discussion on `"How Node Manager Restarts a Managed Server"` in the *Administering Node Manager for Oracle WebLogic Server*.

If the Managed Server cannot connect to the Administration Server during startup, it can retrieve its configuration by reading locally-cached configuration data. A Managed Server that starts in this way is running in Managed Server Independence (MSI) mode.

For a description of MSI mode, and the files that a Managed Server must access to start in MSI mode, see `"Replicate domain config files for Managed Server independence"` in the Administration Console Online Help.

To start a Managed Server in MSI mode:

1. Ensure that the following files are available in the Managed Server's root directory:
   - **msi-config.xml**
   - **SerializedSystemIni.dat**
   - **boot.properties**

   If these files are not in the Managed Server's root directory:

   a. Copy the **config.xml** and **SerializedSystemIni.dat** file from the Administration Server's root directory (or from a backup) to the Managed Server's root directory.

   b. Rename the configuration file to **msi-config.xml**. When you start the server, it will use the copied configuration files.

   Note: Alternatively, use the `-Dweblogic.RootDirectory=path` startup option to specify a root directory that already contains these files.

   ---

2. Start the Managed Server at the command-line or using a script.

   The Managed Server will run in MSI mode until it is contacted by its Administration Server. For information about restarting the Administration Server in this scenario, see "Restarting a Failed Administration Server".

16

Tuning JVM Garbage Collection for Production Deployments

This chapter describes how to tune Java Virtual Machine (JVM) garbage collection performance for Oracle Communications WebRTC Session Controller engine tier servers.

Goals for Tuning Garbage Collection Performance

Production installations of WebRTC Session Controller generally require extremely small response times (under 50 milliseconds) for clients even under peak server loads. A key factor in maintaining brief response times is the proper selection and tuning of the JVM's Garbage Collection (GC) algorithm for WebRTC Session Controller instances in the engine tier.

Whereas certain tuning strategies are designed to yield the lowest average garbage collection times or to minimize the frequency of full GCs, those strategies can sometimes result in one or more very long periods of garbage collection (often several seconds long) that are offset by shorter GC intervals. With a production WebRTC Session Controller installation, all long GC intervals must be avoided to maintain response time goals.

The sections that follow describe GC tuning strategies for Oracle's JVM that generally result in best response time performance.

Modifying JVM Parameters in Server Start Scripts

If you use custom startup scripts to start WebRTC Session Controller engines and replicas, simply edit those scripts to include the recommended JVM options described in the sections that follow.

The Configuration Wizard also installs default startup scripts when you configure a new domain. by default, these scripts are installed in the *Middleware_Home***/user_projects/domains/***domain_name***/bin** directory, where *Middleware_Home* is where you installed the WebRTC Session Controller software and *domain_name* is the name of the domain's directory. The **/bin** directory includes:

- **startWebLogic.cmd**, **startWebLogic.sh**: These scripts start the Administration Server for the domain.
- **startManagedWebLogic.cmd**, **startManagedWebLogic.sh**: These scripts start managed engines and replicas in the domain.

If you use the Oracle-installed scripts to start engines and replicas, you can override JVM memory arguments by first setting the **USER_MEM_ARGS** environment variable in your command shell.

> **Note:** Setting the **USER_MEM_ARGS** environment variable overrides all default JVM memory arguments specified in the Oracle-installed scripts. Always set USER_MEM_ARGS to the full list of JVM memory arguments you intend to use. For example, when using the Sun JVM, always add **-XX:MaxPermSize=128m** to the USER_MEM_ARGS value, even if you only intend to change the default heap space (**-Xms, -Xmx**) parameters.

Tuning Garbage Collection with Oracle JDK

When using Oracle's JDK, the goal in tuning garbage collection performance is to reduce the time required to perform a full garbage collection cycle. You should not attempt to tune the JVM to minimize the frequency of full garbage collections, because this generally results in an eventual forced garbage collection cycle that may take up to several full seconds to complete.

The simplest and most reliable way to achieve short garbage collection times over the lifetime of a production server is to use a fixed heap size with the collector and the parallel young generation collector, restricting the new generation size to at most one third of the overall heap.

Oracle recommends using the Garbage-First (G1) garbage collector. See "Getting Started with the G1 Garbage Collector" for more information on using the Garbage-First collector.

The following example JVM settings are recommended for most production engine tier servers:

```
-server -Xms24G -Xmx24G -XX:PermSize=512m -XX:+UseG1GC -XX:MaxGCPauseMillis=200
-XX:ParallelGCThreads=20 -XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70
```

For production replica servers, use the example settings:

```
-server -Xms4G -Xmx4G -XX:PermSize=512m -XX:+UseG1GC -XX:MaxGCPauseMillis=200
-XX:ParallelGCThreads=20 -XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70
```

For standalone installations, use the example settings:

```
-server -Xms32G -Xmx32G -XX:PermSize=512m -XX:+UseG1GC -XX:MaxGCPauseMillis=200
-XX:ParallelGCThreads=20 -XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70
```

The above options have the following effect:

- **-Xms, -Xmx**: Places boundaries on the heap size to increase the predictability of garbage collection. The heap size is limited in replica servers so that even Full GCs do not trigger SIP retransmissions. **-Xms** sets the starting size to prevent pauses caused by heap expansion.
- **-XX:+UseG1GC**: Use the Garbage First (G1) Collector.
- **-XX:MaxGCPauseMillis**: Sets a target for the maximum GC pause time. This is a soft goal, and the JVM will make its best effort to achieve it.
- **-XX:ParallelGCThreads**: Sets the number of threads used during parallel phases of the garbage collectors. The default value varies with the platform on which the JVM is running.

- **-XX:ConcGCThreads**: Number of threads concurrent garbage collectors will use. The default value varies with the platform on which the JVM is running.
- **-XX:InitiatingHeapOccupancyPercent**: Percentage of the (entire) heap occupancy to start a concurrent GC cycle. GCs that trigger a concurrent GC cycle based on the occupancy of the entire heap and not just one of the generations, including G1, use this option. A value of 0 denotes 'do constant GC cycles'. The default value is 45.

17

Avoiding JVM Delays Caused By Random Number Generation

This chapter describes how to avoid Java Virtual Machine (JVM) delays in Oracle Communications WebRTC Session Controller processes caused by random number generation.

Avoiding JVM Delays Caused by Random Number Generation

The library used for random number generation in Oracle's JVM relies on **/dev/random** by default for UNIX platforms. This can potentially block the WebRTC Session Controller process because on some operating systems **/dev/random** waits for a certain amount of "noise" to be generated on the host system before returning a result. Although **/dev/random** is more secure, Oracle recommends using **/dev/urandom** if the default JVM configuration delays WebRTC Session Controller startup.

To determine if your operating system exhibits this behavior, try displaying a portion of the file from a shell prompt:

```
head -n 1 /dev/random
```

If the command returns immediately, you can use **/dev/random** as the default generator for Oracle's JVM. If the command does not return immediately, use these steps to configure the JVM to use **/dev/urandom**:

1. Open the *JAVA_HOME*/**jre/lib/security/java.security** file in a text editor where *JAVA_HOME* is the location of your java installation.
2. Change the line:

   ```
   securerandom.source=file:/dev/random
   ```

   to read:

   ```
   securerandom.source=file:/dev/urandom
   ```

3. Save your change and exit the text editor.

Part III

Reference

This part provides reference information on Oracle Communications WebRTC Session Controller XML configuration files and their entries. It also provides a list of startup configuration options.

This part contains the following chapters:

- Engine Tier Configuration Reference (sipserver.xml)
- SIP Data Tier Configuration Reference (datatier.xml)
- Diameter Configuration Reference (diameter.xml)

18

Engine Tier Configuration Reference (sipserver.xml)

This chapter describes the Oracle Communications WebRTC Session Controller engine tier configuration file, **sipserver.xml**.

Overview of sipserver.xml

The **sipserver.xml** file is an XML document that configures the SIP container features provided by a WebRTC Session Controller instance in the engine tier of a server installation. **sipserver.xml** is stored in the *domain_home*/**config/custom** subdirectory where *domain_home* is the root directory of the WebRTC Session Controller domain.

Editing sipserver.xml

You should never move, modify, or delete the **sipserver.xml** file during normal operations.

Oracle recommends using the Administration Console to modify **sipserver.xml** indirectly, rather than editing the file manually with a text editor. Using the Administration Console ensures that the **sipserver.xml** document always contains valid XML.

You may need to manually view or edit **sipserver.xml** to troubleshoot problem configurations, repair corrupted files, or to roll out custom configurations to many systems when installing or upgrading WebRTC Session Controller. When you manually edit **sipserver.xml**, you must restart WebRTC Session Controller instances to apply your changes.

Caution: Always use the **SipServer** node in the Administration Console or the WLST utility to make changes to a running WebRTC Session Controller deployment. See Chapter 7, "Configuring WebRTC Session Controller Container Properties."

Steps for Editing sipserver.xml

If you need to modify **sipserver.xml** on a production system, follow these steps:

1. Use a text editor to open the *domain_home*/**config/custom/sipserver.xml** file, where *domain_home* is the root directory of the WebRTC Session Controller domain.
2. Modify the **sipserver.xml** file as necessary. See "XML Schema" for a full description of the XML elements.

3. Save your changes and exit the text editor.
4. Restart or start servers to have your changes take effect:

> **Caution:** Always use the SipServer node in the Administration Console or the WLST utility to make changes to a running WebRTC Session Controller deployment. See Chapter 7, "Configuring WebRTC Session Controller Container Properties" for more information.

5. Test the updated system to validate the configuration.

XML Schema

The schema file for **sipserver.xml** (**wcp-sipserver.xsd**) is installed inside the **wlss-descriptor-binding.jar** library, located in *WL_home*/**wlserver/sip/server/lib**, where *WL_home* is the path to the directory where WebLogic Server is installed.

Example sipserver.xml File

The following shows a simple example of a **sipserver.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<sip-server xmlns="http://www.bea.com/ns/wlcp/wlss/300">
  <overload>
    <threshold-policy>queue-length</threshold-policy>
    <threshold-value>200</threshold-value>
    <release-value>150</release-value>
  </overload>
</sip-server>
```

XML Element Description

The following sections describe each element used in the **sipserver.xml** configuration file. Each section describes an XML element that is contained within the main **sip-server** element.

enable-timer-affinity

The **enable-timer-affinity** element determines the way in which engine tier servers process expired timers. By default (when enable-timer-affinity is omitted from **sipserver.xml**, or is set to **false**), an engine tier server that polls the SIP data tier for expired timers processes all available expired timers. When enable-timer-affinity is set to **true**, engine tier servers polling the SIP data tier process only those expired timers that are associated with call states that the engine last modified (or expired timers for call states that have no owner).

See "Configuring Timer Processing" for more information.

overload

The **overload** element enables you to throttle incoming SIP requests according to a configured overload condition. When an overload condition occurs, WebRTC Session Controller destroys new SIP requests by responding with **503 Service Unavailable** until the configured release value is observed, or until the size of the server's capacity constraints is reduced (see "Overload Control Based on Capacity Constraints").

User-configured overload controls are applied only to initial SIP requests; SIP dialogues that are already active when an overload condition occurs may generate additional SIP requests that are not throttled.

To configure an overload control, you define the three elements described in Table 18–1.

Table 18–1 Nested overload Elements

Element	Description
threshold-policy	A String value that identifies the type of measurement used to monitor overload conditions: ■ **session-rate** measures the rate at which new SIP requests are generated. WebRTC Session Controller determines the session rate by calculating the number of new SIP application connections that were created in the last 5 seconds of operation. See "Overload Control Based on Session Generation Rate". ■ **queue-length** measures the sum of the sizes of the capacity constraint work manager components that processes SIP requests and SIP timers. See "Overload Control Based on Capacity Constraints". **Note:** Execute queues are deprecated and no longer used in WebRTC Session Controller. Capacity constraints are used for execute queues. The policy name **queue-length** was kept for backward compatibility. You must use only one of the above policies to define an overload control. See "Selecting an Appropriate Overload Policy" for more information.
threshold-value	Specifies the measured value that causes WebRTC Session Controller to recognize an overload condition and *start* throttling new SIP requests: ■ When using the **session-rate** threshold policy, **threshold-value** specifies the number of new SIP requests per second that trigger an overload condition. See "Overload Control Based on Session Generation Rate". ■ When using the **queue-length** threshold policy, **threshold-value** specifies the size of the combined number of requests in the SIP transport and SIP timer capacity constraint components that triggers an overload condition. See "Overload Control Based on Capacity Constraints". ■ After the **threshold-value** is observed, WebRTC Session Controller recognizes an overload condition for a minimum of 512 milliseconds during which time new SIP requests are throttled. If multiple overloads occur over a short period, the minimum overload of 512 ms is dynamically increased to avoid repeated overloads. ■ After the minimum overload recognition period expires, the overload condition is terminated only after the configured **release-value** is observed.

Table 18–1 (Cont.) Nested overload Elements

Element	Description
release-value	Specifies the measured value that causes WebRTC Session Controller to end an overload condition and *stop* throttling new SIP requests: ■ When using the **session-rate** threshold policy, **release-value** specifies the number of new SIP requests per second that terminates session throttling. See "Overload Control Based on Session Generation Rate". ■ When using the **queue-length** threshold policy, **release-value** specifies the combined number of requests in the capacity constraints that terminates session throttling. See "Overload Control Based on Capacity Constraints".

Selecting an Appropriate Overload Policy

WebRTC Session Controller provides two different policies for throttling SIP requests:

- The **session-rate** policy throttles sessions when the volume new SIP sessions reaches a configured rate (a specified number of sessions per second).
- The **queue-length** policy throttles requests after the sum of the requests in the **wlss.trasnport** work manager and **wlss.timer.capacity** capacity constraint components reaches a configured size.

You must select only one of the available overload policies. You cannot use both policies simultaneously.

The **session-rate** policy is generally used when a back-end resource having a known maximum throughput (for example, an RDBMS) is used when setting up SIP calls. In this case, the **session-rate** policy enables you to tie the WebRTC Session Controller overload policy to the known throughput capabilities of the back-end resource.

With the **queue-length** policy, WebRTC Session Controller monitors both CPU and I/O bottlenecks to diagnose an overload condition. The **queue-length** policy is generally used with CPU-intensive SIP applications in systems that have no predictable upper bound associated with the call rate.

The following sections describe each policy in detail.

Overload Control Based on Session Generation Rate

WebRTC Session Controller calculates the session generation rate (sessions per second) by monitoring the number of application sessions created in the last 5 seconds. When the session generation rate exceeds the rate specified in the **threshold-value** element, WebRTC Session Controller throttles initial SIP requests until the session generation rate becomes smaller than the configured **release-value**.

The following example configures WebRTC Session Controller to begin throttling SIP requests when the new sessions are created at a rate higher than 50 sessions per second. Throttling is discontinued when the session rate drops to 40 sessions per second:

```
<overload>
  <threshold-policy>session-rate</threshold-policy>
  <threshold-value>50</threshold-value>
  <release-value>40</release-value>
</overload>
```

Overload Control Based on Capacity Constraints

By default, SIP messages are handled by a work manager named **wlss.transport** and SIP timers are processed by a work manager named **wlss.timer**. Each work manager has an associated capacity constraint component that sets the number of requests allotted for SIP message handling and timer processing. Work managers are configured in the **config.xml** file for your WebRTC Session Controller. Work managers allocate threads automatically, as described in the Oracle WebLogic Server documentation. You can also allocate additional threads to the server at start time using the startup option `-Dweblogic.threadpool.MinPoolSize=number_of_threads`.

WebRTC Session Controller performs **queue-length** overload control by monitoring the combined lengths of the configured capacity constraints. When the sum of the requests in the two constraints exceeds the length specified in the **threshold-value** element, WebRTC Session Controller throttles initial SIP requests until the total requests are reduced to the configured **release-value**.

Example 18–1 shows a sample **overload** configuration from **sipserver.xml**. Here, WebRTC Session Controller begins throttling SIP requests when the combined size of the constraints exceeds 200 requests. Throttling is discontinued when the combined length returns to 200 or fewer simultaneous requests.

Example 18–1 Sample overload Definition

```
<overload>
  <threshold-policy>queue-length</threshold-policy>
  <threshold-value>200</threshold-value>
  <release-value>150</release-value>
</overload>
```

Two Levels of Overload Protection

User-configured overload controls (defined in **sipserver.xml**) represent the first level of overload protection provided by WebRTC Session Controller. They mark the onset of an overload condition and initiate simple measures to avoid dropped calls (generating 503 responses for new requests).

If the condition that caused the overload persists or worsens, then the work manager component used to perform work in the SIP Servlet container may itself become overloaded. At this point, the server no longer uses threads to generate 503 responses, but instead begins to drop messages. In this way, the configured size of the SIP container's work manager components represent the second and final level of overload protection employed by the server.

Always configure overload controls in **sipserver.xml** conservatively, and resolve the circumstances that caused the overload in a timely fashion.

message-debug

The **message-debug** element enables and configures access logging with log rotation for WebRTC Session Controller. Use this element only in a development environment, because access logging logs all SIP requests and responses.

To perform more selective logging in a production environment, see Chapter 14, "Logging SIP Requests and Responses."

proxy—Setting Up an Outbound Proxy Server

RFC 3261 defines an outbound proxy as "A proxy that receives requests from a client, even though it may not be the server resolved by the Request-URI. Typically, a UA is

manually configured with an outbound proxy, or can learn about one through auto-configuration protocols."

In WebRTC Session Controller an outbound proxy server is specified using the **proxy** element in **sipserver.xml**. The proxy element defines one or more proxy server URIs. You can change the behavior of the proxy process by setting a proxy policy with the **proxy-policy** tag. Table 18–2, " Nested proxy Elements" describes the possible values for the **proxy** elements.

The default behavior is as if **proxy** policy is in effect. The **proxy** policy means that the request is sent out to the configured outbound Proxy and the Route headers in the request preserving any routing decision taken by WebRTC Session Controller. This configuration enables the outbound proxy to send the request over to the intended recipient after it has performed its actions on the request. The **proxy** policy comes into effect only for the initial requests. As for the subsequent request the Route Set takes precedence over any policy in a dialog. (If the outbound proxy wants to be in the Route Set it can turn record routing on).

Also if a proxy application written on WebRTC Session Controller wishes to override the configured behavior of outbound proxy traversal, then it can add a special header with name X-BEA-Proxy-Policy with the value **domain**. This header is stripped from the request while sending, but the effect is to ignore the configured outbound proxy. Applications use the X-BEA-Proxy-Policy custom header to override the configured policy on a request-by-request basis. The value of the header can be **domain** or **proxy**. Note, however, that if the policy is overridden to **proxy**, the configuration must still have the outbound proxy URIs to route to the outbound proxy.

Table 18–2 Nested proxy Elements

Element	Description
routing-policy	An optional element that configures the behavior of the proxy. Valid values are: ■ **domain**: Proxies messages using the routing rule defined by RFC 3261, ignoring any outbound proxy that is specified. ■ **proxy**: Sends the message to the downstream proxy specified in the default proxy URI. If there are multiple proxy specifications they are tried in the order in which they are specified. However, if the transport tries a UDP proxy, the settings for subsequent proxies are ignored.
uri	The TCP or UDP URI of the proxy server. You must specify at least one URI for a `proxy` element. Place multiple URIs in multiple `uri` elements within the `proxy` element.

Example 18–2 shows the default proxy configuration for WebRTC Session Controller domains. The request in this case is created in accordance with the SIP routing rules, and finally the request is sent to the outbound proxy **sipoutbound.oracle.com**.

Example 18–2 Sample proxy Definition

```
<proxy>
     <routing-policy>proxy</routing-policy>
     <uri>sip:sipoutbound.oracle.com:5060</uri>
     <!-- Other proxy uri tags can be added. - >
</proxy>
```

t1-timeout-interval

This element sets the value of the SIP protocol T1 timer, in milliseconds. Timer T1 also specifies the initial values of Timers A, E, and G, which control the retransmit interval for INVITE requests and responses over UDP.

Timer T1 also affects the values of timers F, H, and J, which control retransmit intervals for INVITE responses and requests; these timers are set to a value of 64*T1 milliseconds. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" for more information.

If **t1-timeout-interval** is not configured, WebRTC Session Controller uses the SIP protocol default value of 500 milliseconds.

t2-timeout-interval

This elements sets the value of the SIP protocol T2 timer, in milliseconds. Timer T2 defines the retransmit interval for INVITE responses and non-INVITE requests. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" for more information.

If **t2-timeout-interval** is not configured, WebRTC Session Controller uses the SIP protocol default value of 4 seconds.

t4-timeout-interval

This elements sets the value of the SIP protocol T4 timer, in milliseconds. Timer T4 specifies the maximum length of time that a message remains in the network. Timer T4 also specifies the initial values of Timers I and K, which control the wait times for retransmitting ACKs and responses over UDP. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" for more information.

If **t4-timeout-interval** is not configured, WebRTC Session Controller uses the SIP protocol default value of 5 seconds.

timer-b-timeout-interval

This elements sets the value of the SIP protocol Timer B, in milliseconds. Timer B specifies the length of time a client transaction attempts to retry sending a request. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" for more information.

If **timer-b-timeout-interval** is not configured, the Timer B value is derived from timer T1 (64*T1, or 32000 milliseconds by default).

timer-f-timeout-interval

This elements sets the value of the SIP protocol Timer F, in milliseconds. Timer F specifies the timeout interval for retransmitting non-INVITE requests. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" for more information.

If **timer-f-timeout-interval** is not configured, the Timer F value is derived from timer T1 (64*T1, or 32000 milliseconds by default).

max-application-session-lifetime

This element sets the maximum amount of time, in minutes, that a SIP application session can exist before WebRTC Session Controller invalidates the session. **max-application-session-lifetime** acts as an upper bound for any timeout value specified using the **session-timeout** element in a **sip.xml** file, or using the **setExpires** API.

A value of **-1** (the default) specifies that there is no upper bound to application-configured timeout values.

enable-local-dispatch

enable-local-dispatch is a server optimization that helps avoid unnecessary network traffic when sending and forwarding messages. You enable the optimization by setting this element **true**. When **enable-local-dispatch** enabled, if a server instance needs to send or forward a message and the message destination is the engine tier's cluster address or the local server address, then the message is routed internally to the local server instead of being sent through the network.

You may want to disable this optimization if you feel that routing internal messages could skew the load on servers in the engine tier, and you prefer to route all requests through a configured load balancer.

By default **enable-local-dispatch** is set to **false**.

cluster-loadbalancer-map

The **cluster-loadbalancer-map** element is used only when upgrading WebRTC Session Controller software, or when upgrading a production SIP Servlet to a new version. It is not required or used during normal server operations.

During a software upgrade, multiple engine tier clusters are defined to host the older and newer software versions. **A cluster-loadbalancer-map** defines the virtual IP address (defined on your load balancer) that correspond to an engine tier cluster configured for an upgrade. WebRTC Session Controller uses this mapping to ensure that engine tier requests for timers and call state data are received from the correct "version" of the cluster. If a request comes from an incorrect version of the software, WebRTC Session Controller uses the **cluster-loadbalancer-map** to forward the request to the correct cluster.

Each **cluster-loadbalancer-map** entry contains the two elements described in Table 18–3.

Table 18–3 Nested cluster-loadbalancer-map Elements

Element	Description
cluster-name	The configured name of an engine tier cluster.
sip-uri	The internal SIP URI that maps to the engine tier cluster. This corresponds to a virtual IP address that you have configured in your load balancer. The internal URI forwards requests to the correct cluster version during an upgrade.

Example 18–3 shows a sample **cluster-loadbalancer-map** entry used during an upgrade.

Example 18–3 Sample cluster-loadbalancer-map Entry

```
<cluster-loadbalancer-map>
```

```
   <cluster-name>EngineCluster</cluster-name>
   <sip-uri>sip:172.17.0.1:5060</sip-uri>
</cluster-loadbalancer-map>
<cluster-loadbalancer-map>
   <cluster-name>EngineCluster2</cluster-name>
   <sip-uri>sip:172.17.0.2:5060</sip-uri>
</cluster-loadbalancer-map>
```

See Chapter 13, "Upgrading Production WebRTC Session Controller Software," for more information.

default-behavior

This element defines the default behavior of the WebRTC Session Controller instance if the server cannot match an incoming SIP request to a deployed SIP Servlet (or if the matching application has been invalidated or timed out). Valid values are:

- **proxy**: Act as a proxy server.
- **ua**: Act as a User Agent.

proxy is used as the default if you do not specify a value.

When acting as a User Agent (UA), WebRTC Session Controller acts in the following way in response to SIP requests:

- ACK requests are discarded without notice.
- CANCEL or BYE requests receive response code 481 - Transaction does not exist.
- All other requests receive response code 500 - Internal server error.

When acting as a proxy requests are automatically forwarded to an outbound proxy (see "proxy—Setting Up an Outbound Proxy Server") if one is configured. If no proxy is defined, WebRTC Session Controller proxies to a specified Request URI only if the Request URI does not match the IP and port number of a known local address for a SIP Servlet container, or a load balancer address configured for the server. This ensures that the request does not constantly loop to the same servers. When the Request URI matches a local container address or load balancer address, WebRTC Session Controller instead acts as a UA.

default-servlet-name

This element specifies the name of a default SIP Servlet to call if an incoming initial request cannot be matched to a deployed Servlet (using standard **servlet-mapping** definitions in **sip.xml**). The name specified in the **default-servlet-name** element must match the **servlet-name** value of a deployed SIP Servlet. For example:

```
<default-servlet-name>myServlet</default-servlet-name>
```

If the name defined in **default-servlet-name** does not match a deployed Servlet, or no value is supplied (the default configuration), WebRTC Session Controller registers the name `com.bea.wcp.sip.engine.BlankServlet` as the default Servlet. The **BlankServlet** name is also used if a deployed Servlet registered as the **default-servlet-name** is undeployed from the container.

BlankServlet's behavior is configured with the **default-behavior** element. By default the Servlet proxies all unmatched requests. However, if the **default-behavior** element is set to **ua** mode, **BlankServlet** is responsible for returning 481 responses for CANCEL and BYE requests, and 500/416 responses in all other cases. **BlankServlet** does not respond to ACK, and it always invalidates the application session.

retry-after-value

Specifies the number of seconds used in the **Retry-After** header for 5xx response codes. This value can also include a parameter or a reason code, such as "Retry-After: 18000;duration=3600" or "Retry-After: 120 (I'm in a meeting)."

If the this value is not configured, WebRTC Session Controller uses the default value of 180 seconds.

sip-security

WebRTC Session Controller enables you to configure one or more trusted hosts for which authentication is not performed. When WebRTC Session Controller receives a SIP message, it calls **getRemoteAddress()** on the SIP Servlet message. If this address matches an address defined in the server's trusted host list, no further authentication is performed for the message.

The **sip-security** element defines one or more trusted hosts, for which authentication is not performed. The **sip-security** element contains one or more **trusted-authentication-host** or **trusted-charging-host** elements, each of which contains a trusted host definition. A trusted host definition can consist of an IP address (with or without wildcard placeholders) or a DNS name. Example 18–4 shows a sample **sip-security** configuration.

Example 18–4 Sample Trusted Host Configuration

```
<sip-security>

<trusted-authentication-host>myhost1.mycompany.com</trusted-authentication-host>
   <trusted-authentication-host>172.*</trusted-authentication-host>
</sip-security>
```

route-header

3GPP TS 24.229 Version 7.0.0 :

http://www.3gpp.org/ftp/Specs/archive/24_series/24.229/24229-700.zip requires that IMS Application Servers generating new requests (for example, as a B2BUA) include the S-CSCF route header. In WebRTC Session Controller, the S-CSCF route header must be statically defined as the value of the **route-header** element in **sipserver.xml**. For example:

```
<route-header>
   <uri>Route: sip:wlss1.bea.com</uri>
</route-header>
```

engine-call-state-cache-enabled

WebRTC Session Controller provides the option for engine tier servers to cache a portion of the call state data locally or in the SIP data tier, to improve performance with SIP-aware load balancers. When a local cache is used, an engine tier server first checks its local cache for existing call state data. If the cache contains the required data, and the local copy of the data is up-to-date (compared to the SIP data tier copy), the engine locks the call state in the SIP data tier but reads directly from its cache.

By default the engine tier cache is enabled. To disable caching, set **engine-call-state-cache-enabled** to **false**:

```
<engine-call-state-cache-enabled>false</engine-call-state-cache-enabled>
```

See Chapter 11, "Using the Engine Tier Cache" for more information.

server-header

WebRTC Session Controller enables you to control when a Server header is inserted into SIP messages. You can use this functionality to limit or eliminate Server headers to reduce the message size for wireless networks, or to increase security.

By default, WebRTC Session Controller inserts no Server header into SIP messages. Set the **server-header** to one of the following string values to configure this behavior:

- **none** (the default) inserts no Server header.
- **request** inserts the Server header only for SIP requests generated by the server.
- **response** inserts the Server header only for SIP responses generated by the server.
- **all** inserts the Server header for all SIP requests and responses.

For example, the following element configures WebRTC Session Controller to insert a Server header for all generated SIP messages:

```
<server-header>all</server-header>
```

See also "server-header-value".

server-header-value

WebRTC Session Controller enables you to control the text that is inserted into the Server header of generated messages. This provides additional control over the size of SIP messages and also enables you to mask the server entity for security purposes. By default, WebRTC Session Controller does not insert a Server header into generated SIP messages (see "server-header"). If Server header insertion is enabled but no **server-header-value** is specified, WebRTC Session Controller inserts the value **WebLogic SIP Server**. To configure the header contents, enter a string value. For example:

```
<server-header-value>MyCompany Application Server</server-header-value>
```

persistence

The **persistence** element enables or disables writing call state data to an RDBMS, or to a remote, geographically-redundant WebRTC Session Controller installation. For sites that use geographically-redundant replication features, the **persistence** element also defines the site ID and the URL at which to persist call state data.

The `persistence` element contains the sub-elements described in Table 18–4.

Table 18–4 Nested persistence Elements

Element	Description
default-handling	Determines whether WebRTC Session Controller observes persistence hints for RDBMS persistence or geographical-redundancy. This element can have one of the following values: ■ **all**: Specifies that call state data may be persisted to both an RDBMS store and to a geographically-redundant WebRTC Session Controller installation. This is the default behavior. Replication to either destination also requires that the available resources (JDBC datasource and remote JMS queue) are available. ■ **db**: Specifies that long-lived call state data is replicated to an RDBMS if the required JDBC datasource and schema are available. ■ **geo**: Specifies that call state data is persisted to a remote, geographically-redundant site if the configured site URL contains the necessary JMS resources. ■ **none**: Specifies that only in-memory replication is performed to other replicas in the SIP data tier cluster. Call state data is not persisted in an RDBMS or to an external site.
geo-site-id	Specifies the site ID of this installation. All installations that participate in geographically-redundant replication require a unique site ID.
geo-remote-t3-url	Specifies the remote WebRTC Session Controller installation to which this site replicates call state data. You can specify a single URL corresponding to the engine tier cluster of the remote installation. You can also specify a comma-delimited list of addresses corresponding to each engine tier server. The URLs must specify the t3 protocol.

Example 18–5 shows a sample configuration that uses RDBMS storage for long-lived call state and geographically-redundant replication. Call states are replicated to two engine tier servers in a remote location.

Example 18–5 Sample persistence Configuration

```
<persistence>
  <default-handling>all</default-handling>
  <geo-site-id>1</geo-site-id>

<geo-remote-t3-url>t3://remoteEngine1:7050,t3://remoteEngine2:7051</geo-remote-t3-
url>
</persistence>
```

use-header-form

This element configures the server-wide, default behavior for using or preserving compact headers in SIP messages. You can set this element to one of the following values:

- **compact**: WebRTC Session Controller uses the compact form for all system-generated headers. However, any headers that are copied from an originating message (rather than generated) use their original form.
- **force compact**: WebRTC Session Controller uses the compact form for all headers, converting long headers in existing messages into compact headers as necessary.
- **long**: WebRTC Session Controller uses the long form for all system-generated headers. However, any headers that are copied from an originating message (rather than generated) use their original form.

- **force long**: WebRTC Session Controller uses the long form for all headers, converting compact headers in existing messages into long headers as necessary.

enable-dns-srv-lookup

This element enables or disables WebRTC Session Controller DNS lookup capabilities. If you set the element to **true**, then the server can use DNS to:

- Discover a proxy server's transport, IP address, and port number when a request is sent to a SIP URI.
- Resolve an IP address and port number during response routing, depending on the contents of the Sent-by field.

For proxy discovery, WebRTC Session Controller uses DNS resolution only once per SIP transaction to determine transport, IP, and port number information. All retransmissions, ACKs, or CANCEL requests are delivered to the same address and port using the same transport. For details about how DNS resolution takes place, see *RFC 3263: Session Initiation Protocol (SIP): Locating SIP Servers* (`http://www.ietf.org/rfc/rfc3263.txt`).

When a proxy needs to send a response message, WebRTC Session Controller uses DNS lookup to determine the IP address and port number of the destination, depending on the information provided in the **sent-by** field and **Via** header.

By default, DNS resolution is not used (**false**).

Note: Because DNS resolution is performed within the context of SIP message processing, any DNS performance problems result in increased latency performance. Oracle recommends using a caching DNS server in a production environment to minimize potential performance problems.

---

connection-reuse-pool

WebRTC Session Controller includes a connection pooling mechanism that minimizes communication overhead with a Session Border Control (SBC) function or Serving Call Session Control Function (S-CSCF). You can configure multiple, fixed pools of connections to different addresses.

WebRTC Session Controller opens new connections from the connection pool on demand as the server makes requests to a configured address. The server then multiplexes new SIP requests to the address using the already-opened connections, rather than repeatedly terminating and re-creating new connections. Opened connections are reused in a round-robin fashion. Opened connections remain open until they are explicitly closed by the remote address.

Connection reuse pools are not used for incoming requests from a configured address.

To configure a connection reuse pool, you define the four nested elements described in Table 18–5.

Table 18–5 Nested connection-reuse-pool Elements

Element	Description
pool-name	A String value that identifies the name of this pool. All configured `pool-name` elements must be unique to the domain.

Table 18–5 (Cont.) Nested connection-reuse-pool Elements

Element	Description
destination	Specifies the IP address or host name of the destination SBC or S-CSCF. WebRTC Session Controller opens or reuses connection in this pool only when making requests to the configured address.
destination-port	Specifies the port number of the destination SBC or S-CSCF.
maximum-connections	Specifies the maximum number of opened connections to maintain in this pool.

Example 18–6 shows a sample connection-reuse-pool configuration having two pools.

Example 18–6 Sample connection-reuse-pool Configuration

```
<connection-reuse-pool>
   <pool-name>SBCPool</pool-name>
   <destination>MySBC</destination>
   <destination-port>7070</destination-port>
   <maximum-connections>10</maximum-connections>
</connection-reuse-pool>
<connection-reuse-pool>
   <pool-name>SCSFPool</pool-name>
   <destination>192.168.1.6</destination>
   <destination-port>7071</destination-port>
   <maximum-connections>10</maximum-connections>
</connection-reuse-pool>
```

globally-routable-uri

This element enables you to specify a Globally-Routable User Agent URI (GRUU) that WebRTC Session Controller automatically inserts into Contact and Route-Set headers when communicating with network elements. The URI specified in this element should be the GRUU for the entire WebRTC Session Controller cluster. (In a single-server domain, use a GRUU for the server itself.)

User Agents (UAs) deployed on WebRTC Session Controller typically obtain GRUUs through a registration request. In this case, the application code is responsible both for requesting and subsequently handling the GRUU. To request a GRUU, the UA includes the `+sip.instance` field parameter in the Contact header in each Contact for which GRUU is required. Upon receiving a GRUU, the UA uses the GRUU as the URI for the Contact header field when generating new requests.

domain-alias-name

This element defines one or more domains for which WebRTC Session Controller is responsible. If a message has a destination domain that matches a domain specified with a **domain-alias-name** element, WebRTC Session Controller processes the message locally, rather than forwarding it.

The **sipserver.xml** configuration file can have multiple **main-alias-name** elements. Each element can specify either:

- an individual, fully-qualified domain name, such as **myserver.mycompany.com**, or
- a domain name starting with an initial wildcard character, such as ***.mycompany.com**, used to represent all matching domains. Only a single

wildcard character is supported, and it must be used as the first element of the domain name.

> **Note:** You can also identify these domain names using the Domain Aliases field in the Configuration > General tab of the SipServer Administration Console extension.

enable-rport

This element determines whether WebRTC Session Controller automatically adds an **rport** parameter to **Via** headers when acting as a UAC. By default, the server does not add the **rport** parameter; set the element to **true** to automatically add **rport** to requests generated by the server.

> **Note:** You can also set this parameter to **true** by selecting the Symmetric Response Routing option in the Administration Console. In the Administration Console, select **Configuration**, then select the **General** tab of the SipServer Administration console extension.

The `rport` parameter is used for symmetric response routing as described in RFC 3581 (`http://www.ietf.org/rfc/rfc3581.txt`). When a message is received by an RFC 3581-compliant server, such as WebRTC Session Controller, the server responds using the remote UDP port number from which the message was received, rather than the port number specified in the **Via** header. This behavior is frequently used when servers reside behind gateway devices that perform Network Address Translation (NAT). The NAT devices maintain a binding between the internal and external port numbers, and all communication must be initiated through the gateway port.

WebRTC Session Controller is compliant with RFC 3581, and will honor the **rport** parameter even if you set the **enable-rport** element to **false**. The **enable-rport** element only specifies whether the server automatically adds `rport` to the requests it generates when acting as a UAC. To disable **rport** handling completely (disable RFC 3581 support), you must start the server with the command-line option, `-Dwlss.udp.uas.rport=false`.

> **Note:** `rport` support as described in RFC 3581 requires that SIP responses include the source port of the original SIP request. Because source port information is frequently treated as sensitive data, Oracle recommends using the TLS transport.

image-dump-level

This element specifies the level of detail to record in WebRTC Session Controller diagnostic image files. You can set this element to one of two values:

- **basic**: Records all diagnostic data except for call state data.
- **full**: Records all diagnostic data including call state data.

Note: Recording call state data in the image file can be time consuming. By default, image dump files are recorded using the `basic` option.

You can also set this parameter using the **Configuration** > **General** tab of the **SipServer** Administration Console extension.

stale-session-handling

WebRTC Session Controller uses encoded URIs to identify the call states and application sessions associated with a message. When an application is undeployed or upgraded to a new version, incoming requests may have encoded URIs that specify "stale" or nonexistent call or session IDs. The **stale-session-handling** element enables you to configure the action that WebRTC Session Controller takes when it encounters stale session data in a request. The following actions are possible:

- **drop**: Drops the message without logging an error. This setting is desirable for systems that frequently upgrade applications using WebRTC Session Controller's in-place upgrade feature. Using the **drop** action ensures that messages intended for older, incompatible versions of a deployed application are dropped.
- **error**: Responds with an error, so that a UAC might correct the problem. This is the default action. Messages having a **To:** tag cause a **481 Call/Transaction Does Not Exist** error, while those without the tag cause a **404 Not Found** error.
- **continue**: Ignores the stale session data and continues processing the request.

Note: When it encounters stale session data, WebRTC Session Controller applies the action specified by **stale-session-handling** before considering the value of the **default-behavior** element. The **default-behavior** is performed only when you have configured **stale-session-handling** to perform the **continue** action.

enable-contact-provisional-response

By default WebRTC Session Controller does not place a Contact header in non-reliable provisional (1xx) responses that have a To header. If you deploy applications that expect the Contact header to be present in such 1xx responses, set this element to true:

```
<enable-contact-provisional-response>true</enable-contact-provisional-response>
```

Setting this element to **true** does not affect **100 Trying** responses.

19

SIP Data Tier Configuration Reference (datatier.xml)

The chapter describes the Oracle Communications WebRTC Session Controller SIP data tier configuration file, **datatier.xml**:

Overview of datatier.xml

The **datatier.xml** configuration file identifies servers that manage the concurrent call state for SIP applications, and defines how those servers are arranged into SIP data tier *partitions*. A *partition* refers to one or more SIP data tier server instances that manage the same portion of the call state. Multiple servers in the same partition are referred to as *replicas* because they all manage a copy of the same portion of the call state.

datatier.xml is stored in the *domain_home*/**config/custom** subdirectory where *domain_home* is the root directory of WebRTC Session Controller domain.

Editing datatier.xml

You can edit **datatier.xml** using either the Administration Console or a text editor. Changes to the SIP data tier configuration cannot be applied to servers dynamically; you must restart servers to change SIP data tier membership or reconfigure partitions.

XML Schema

This schema file is bundled within the **wlss-descriptor-binding.jar** library, installed in the *Middleware_Home*/**wlserver/sip/server/lib** directory where *Middleware_Home* is the path to the directory where WebLogic Server is installed

Example datatier.xml File

Example 19–1 shows the template **datatier.xml** file created using the Configuration Wizard. See also "Example SIP Data Tier Configurations and Configuration Files" in Chapter 8, "Configuring SIP Data Tier Partitions and Replicas."

Example 19–1 Default datatier.xml File

```
<st:data-tier xmlns:st="http://bea.com/wcp/sip/management/internal/webapp">
 <st:partition>
   <st:name>partition-0</st:name>
   <st:server-name>replica1</st:server-name>
   <st:server-name>replica2</st:server-name>
 </st:partition>
```

```
</st:data-tier>
```

XML Element Description

datatier.xml contains one or more `partition` elements that define servers' membership in a SIP data tier partition. All SIP data tier clusters must have at least one `partition`. Each partition contains the XML elements described in Table 19–1.

Table 19–1 Nested partition Elements

Element	Description
name	A String value that identifies the name of the partition. Oracle recommends including the number of the partition (starting at 0) in the text of the name for administrative purposes. For example, "partition-0."
server-name	Specifies the name of a WebRTC Session Controller instance that manages call state in this partition. You can define up two three servers per `partition` element. Multiple servers in the same partition maintain the same call state data, and are referred to as *replicas*. Oracle recommends including the number of the server (starting with 0) and the number of the partition in the server name for administrative purposes. For example, "replica-0-0."

20

Diameter Configuration Reference (diameter.xml)

This chapter describes the Oracle Communications WebRTC Session Controller Diameter configuration file, **diameter.xml**.

Overview of diameter.xml

The **diameter.xml** file configures attributes of a Diameter node, such as:

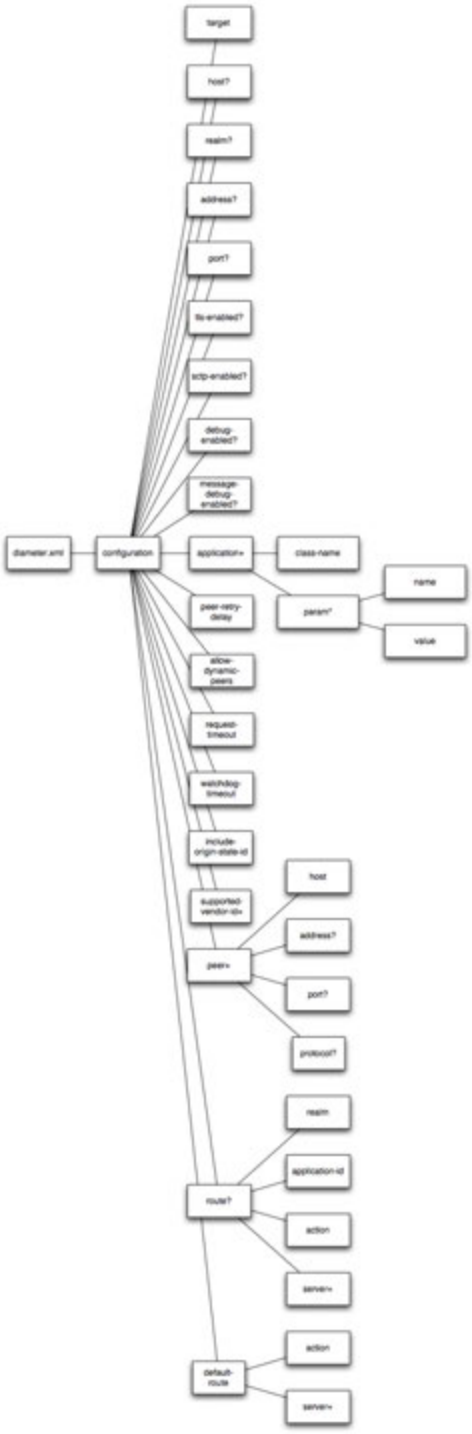
- The host identity of the Diameter node
- The Diameter applications that are deployed on the node
- Connection information for Diameter peer nodes
- Routing information and default routes for handling Diameter messages.

The Diameter protocol implementation reads the configuration file at start time. **diameter.xml** is stored in the domain_home**/config/custom** subdirectory where *domain_home* is the root directory of the WebRTC Session Controller domain.

Graphical Representation

Figure 20–1 shows the element hierarchy of the **diameter.xml** file.

Figure 20–1 Element Hierarchy of diameter.xml



Editing diameter.xml

WARNING: You should never move, modify, or delete the diameter.xml file during normal operations.

Oracle recommends using the Administration Console to modify **diameter.xml** indirectly, rather than editing the manually with a text editor. Using the Administration Console ensures that the **diameter.xml** document always contains valid XML.

You may need to manually view or edit **diameter.xml** to troubleshoot problem configurations, repair corrupted files, or to roll out custom Diameter node configurations to a large number of machines when installing or upgrading WebRTC Session Controller. When you manually edit **diameter.xml**, you must restart Diameter nodes to apply your changes.

Caution: Always use the Diameter node in the Administration Console or the WLST utility, as described in Chapter 7, "Configuring WebRTC Session Controller Container Properties" to make changes to a running WebRTC Session Controller deployment.

Steps for Editing diameter.xml

If you need to modify **diameter.xml** on a production system, follow these steps:

1. Use a text editor to open the *WSC_home*/**config/custom/diameter.xml** file, where *WSC_home* is the root directory of the WebRTC Session Controller domain.
2. Modify the **diameter.xml** file as necessary. See "XML Element Description" for a full description of the XML elements.
3. Restart or start servers to have your changes take effect.
4. Test the updated system to validate the configuration.

XML Schema

The XML schema file (**wcp-diameter.xsd**) is bundled within the **wlssdiameter.jar** library, installed in *WL_home*/**wlserver/sip/server/lib**, where *WL_home* is the path to the directory where WebLogic Server is installed.

Example diameter.xml File

See Chapter 5, "Configuring WebRTC Session Controller Diameter Rx to PCRF Integration" for examples of **diameter.xml** configuration files.

XML Element Description

The following sections describe each XML element in **diameter.xml**.

configuration

The top level **configuration** element contains the entire diameter node configuration.

target

Specifies one or more target WebRTC Session Controller instances to which the node configuration is applied. The target servers must be defined in the **config.xml** file for your domain.

host

Specifies the host identity for this Diameter node. If no **host** element is specified, the identity is taken from the local server's host name. The host identity may or may not match the DNS name.

> **Note:** When configuring Diameter support for multiple Sh client nodes, it is best to omit the `host` element from the **diameter.xml** file. This omission enables you to deploy the same Diameter web application to all servers in the engine tier cluster, and the host name is dynamically obtained for each server instance.

realm

Specifies the realm name for which this Diameter node has responsibility. You can run multiple Diameter nodes on a single host using different realms and listen port numbers. The HSS, Application Server, and relay agents must all agree on a realm name or names. The realm name for the HSS and Application Server need not match.

If you omit the **realm** element, the realm named is derived using the domain name portion of the host name, if the host name is fully-qualified (for example, host@oracle.com).

address

Specifies the listen address for this Diameter node, using either the DNS name or IP address. If you do not specify an address, the node uses the **host** identity as the listen address.

> **Note:** The `host` identity may or may not match the DNS name of the Diameter node. Oracle recommends configuring the **address** element with an explicit DNS name or IP address to avoid configuration errors.

port

Specifies the TCP or TLS listen port for this Diameter node. The default port is 3868.

tls-enabled

This element is used only for standalone node operation to advertise TLS capabilities.

WebRTC Session Controller ignores the **tls-enabled** element for nodes running within a server instance. Instead, TLS transport is reported as enabled if the server instance has configured a Network Channel having TLS support (a diameters channel). See "Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol" in Chapter 5, "Configuring WebRTC Session Controller Diameter Rx to PCRF Integration."

sctp-enabled

This element is used only for standalone node operation to advertise SCTP capabilities.

WebRTC Session Controller ignores the **sctp-enabled** element for nodes running within a server instance. Instead, SCTP transport is reported as enabled if the server

instance has configured a Network Channel having SCTP support (a diameter-sctp channel). See "Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol" in Chapter 5, "Configuring WebRTC Session Controller Diameter Rx to PCRF Integration."

debug-enabled

Specifies a boolean value to enable or disable debug message output. Debug messages are disabled by default.

message-debug-enabled

Specifies a boolean value to enable or disable tracing of Diameter messages. This element is disabled by default.

application

Configures a particular Diameter application to run on the selected node. WebRTC Session Controller includes applications to support nodes that act as Diameter Rx clients, Diameter relay agents, or Home Subscriber Servers (HSS). The HSS application is a simulator that is provided only for development or testing purposes.

class-name

Specifies the application class file to load.

param*

Specifies one or more optional parameters to pass to the application class.

name Specifies the name of the application parameter.

value Specifies the value of the parameter.

peer-retry-delay

Specifies the number of seconds this node waits between retries to Diameter peers. The default value is 30 seconds.

allow-dynamic-peers

Specifies a boolean value that enables or disables dynamic peer configuration. Dynamic peer support is disabled by default. Oracle recommends enabling dynamic peers only when using the TLS transport, because no access control mechanism is available to restrict hosts from becoming peers.

request-timeout

Specifies the number of milliseconds to wait for an answer from a peer before timing out.

watchdog-timeout

Specifies the number of seconds used for the Diameter Tw watchdog timer.

include-origin-state-id

Specifies whether the node should include the origin state AVP in requests and answers.

supported-vendor-id+

Specifies one or more vendor IDs to be added to the **Supported-Version-Ids** AVP in the capabilities exchange.

peer+

Specifies connection information for an individual Diameter peer. You can choose to configure connection information for individual peer nodes, or allow any node to be dynamically added as a peer. Oracle recommends using dynamic peers only if you are using the TLS transport, because there is no way to filter or restrict hosts from becoming peers when dynamic peers are enabled.

When configuring Sh client nodes, the **peers** element should contain peer definitions for each Diameter relay agent deployed to your system. If your system does not use relay agents, you must include a peer entry for the Home Subscriber Server (HSS) in the system and for all other engine tier nodes that act as Sh client nodes.

When configuring Diameter relay agent nodes, the **peers** element should contain peer entries for all Diameter client nodes that access the peer and the HSS.

host

Specifies the host identity for a Diameter peer.

address

Specifies the listen address for a Diameter peer. If you do not specify an address, the host identity is used.

port

Specifies the TCP or TLS port number for this Diameter peer. The default port is 3868.

protocol

Specifies the protocol used by the peer. This element may be one of **tcp** or **sctp**.

route

Defines a realm-based route that this node uses when resolving messages.

When configuring Sh client nodes, you should specify a route to each Diameter relay agent node deployed in the system and a **default-route** to a selected relay. If your system does not use relay agents, simply configure a single **default-route** to the HSS.

When configuring Diameter relay agent nodes, specify a single **default-route** to the HSS.

realm

The target realm used by this route.

application-id

The target application ID for the route.

action

An action type that describes the role of the Diameter node when using this route. The value of this element can be one of the following:

- none
- local
- relay
- proxy
- redirect

server+

Specifies one or more target servers for this route. Any server specified in the **server** element must also be defined as a **peer** to this Diameter node, or dynamic peer support must be enabled.

default-route

Defines a default route to use when a request cannot be matched to a configured route.

action

Specifies the default routing action for the Diameter node. See "route" for more information.

server+

Specifies one or more target servers for the default route. Any server you include in this element must also be defined as a **peer** to this Diameter node, or dynamic peer support must be enabled.