Oracle® Communications WebRTC Session Controller

System Administrator's Guide

Release 7.2

E69506-01

May 2016

ORACLE®

Oracle Communications WebRTC Session Controller System Administrator's Guide, Release 7.2

E69506-01

Contents

3 Using the Administration Console and WLST

4 Configuring WebRTC Session Controller Authentication

13 Monitoring Statistics and Resource Limits

14 Avoiding and Recovering From Server Failures

15 Tuning JVM Garbage Collection for Production Deployments

16 Avoiding JVM Delays Caused By Random Number Generation

Part III Reference

17 Engine Server Configuration Reference (sipserver.xml)

18 SIP Coherence Configuration Reference (coherence.xml)

19 Diameter Configuration Reference (diameter.xml)

Part IV WebRTC Session Controller Media Engine Administration

20 Managing and Administering ME Systems

21 Configuring Permissions, Users, and Authorization

22 Enabling ME Interfaces and Protocols

23 Enabling ME Services

24 Configuring ME Accounting and Archiving

25 Configuring Domain Name Systems (DNS)

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 Installation Guide*
- *Oracle Communications WebRTC Session Controller Security Guide*
- *Oracle Communications WebRTC Session Controller Extension Developer's Guide*
- *Oracle Communications WebRTC Session Controller Application Developer's Guide*
- *Oracle Communications WebRTC Session Controller Media Engine Object Reference*
- *Oracle Communications WebRTC Session Controller Release Notes*

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 that have purchased support 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:

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 state storage.

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 server for the change to take effect. Configuration for SIP state-storage nodes cannot be changed dynamically, so you must restart SIP Coherence servers to change the number of partitions.

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 functionality. This extension also enables you to access SIP state storage properties and runtime statistics.

- **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" 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 configuration settings.
- **coherence.xml** identifies servers that participate in SIP state storage, and also defines the number of threads and partitions available in the state storage service.
- **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 state-storage nodes or engine 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.
- Configure Coherence call-state storage servers and specify distributed cache service parameters. See "Configuring Coherence" 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 and EDRs" for more information.

2

Configuring WebRTC Session Controller

This chapter describes how to configure application profiles, the Signaling Engine, the Media Engine, and the Notification Service for multitenancy in the WebRTC Session Controller web console.

About Multitenancy

WebRTC Session Controller 7.2 is a platform that can support multi-tenant Software as a service (SaaS) applications that you host elsewhere. An in-premise installation of WebRTC Session Controller enables multiple departments of a customer to use WebRTC Session Controller for their specific applications. The main benefits of multi-tenancy in WebRTC Session Controller are increased density, tenant isolation, and simplified cloud configuration and management.

In your WebRTC Session Controller installation, a **tenant** represents a configuration scope for a customer or a department that is authenticated to use its services. Multiple tenants can access a single WebRTC Session Controller installation or it can be configured as a single tenant installation. A tenant is associated with a tenant key that allows you to manage and track all usage of the installation by that customer or department. Every user account that is authenticated to access WebRTC Session Controller is associated with a tenant.

About Tenants

As the administrator of WebRTC Session Controller, you set up and manage the configuration for one or more tenants. These configurations allow the tenants to utilize the WebRTC Session Controller installation for *their* users. WebRTC Session Controller establishes the tenant profile with some default resource limits and allots an equal slice of the available resource limits to the tenant profile. In a multitenancy scenario, the tenant profile configuration WebRTC Session Controller enables a partitioning that isolate each tenant.

As a system administrator for the tenant, you set up the global configuration of the runtime environment for the tenant. To do so, you first review these default resource limits and usage parameters at the global level and update them for your environment. WebRTC Session Controller displays these global settings as selections for configuring the profiles for applications that you own in the environment.

When you have configured the global settings for your WebRTC Session Controller environment you register the individual applications for your customers to use. You create application profiles to associate with each tenant. To create the application profile, you can select from the available (global) values and enter others specific to the application.

About the Tenant Key

When a tenant profile is created, WebRTC Session Controller generates a key as an identifier for the tenant. WebRTC Session Controller uses this key to associate the request with a tenant and identify the tenant profile. It is recommended that key is not hard coded in the applications. For example, SaaS application could store this key in a database as a way for the application to access the key.

About Managing Tenant and Application Profiles

Tenants are registered with specific SaaS applications using the tools within the SaaS applications or in the cloud infrastructure. Both application profiles and tenant profiles contain the following information:

- Security realm to which the tenant or application belongs.
- Resource limits.
- Statistics to collect on the tenant or application.
- Groovy properties defined for the use of that tenant.

Application profiles contain in addition, information on any packages and scripts that are defined for them.

How Multitenancy Works

A websocket connection belongs to a tenant profile. WebRTC Session Controller associates the HTTP request that is used for the login and the request used for the websocket handshake with the tenant profile.

Web and mobile applications include the tenant key as the HTTP request parameter named **tenant_profile_key**. When a single user identity is associated with multiple tenant profiles, a request from each tenant can reach different SIP proxy registrars. WebRTC Session Controller Signaling engine stores the tenant key as the value of the **tenantToken** parameter in the SipURI of the contact header in the requests it sends to lightweight proxy registrar. In turn, the lightweight proxy registrar incorporates this information in its decision making process and sends each request from a tenant to the corresponding proxies.

In the runtime environment, WebRTC Session Controller gathers and stores the statistics for each tenant profile and each application profile. Any notification configuration contains the associated tenant identification information. And the resource limit profiles contains the resource limit for maximum number of notifications.

WebRTC Session Controller allows traffic to domains for a tenant based on the domains configured in the tenant profile. If your installation supports SaaS applications, configure the domain name in the tenant profile. Doing so enables your SaaS applications to provide cross-domain access.

About Service Level Agreements

Service level agreements (SLAs) are set between a SaaS application and its tenant. These SLAs translate the defined quantities of resources within the services used by the SaaS application. For example, an SLA may specify usage parameters for services such as the database or messaging server. In addition, where necessary, SaaS applications can allot resources to their tenants and monitor those resources within WebRTC Session Controller. The users of the tenants use a "slice" of the SaaS application.

About Managing Tenant and Application Profiles

The following sections in this chapter describe how you configure tenant profiles and manage global- and application-level configurations in WebRTC Session Controller Administration Console.

You can also use MBeans to create and remove tenant profiles, configure and manage application profiles in the runtime environment using MBeans. See "Managing Application and Tenant Profiles Using WebLogic Scripting Tool".

WebRTC Session Controller provides application level statistics for each tenant. For information on monitoring the tenancy and applications, see "Monitoring Statistics and Resource Limits". For an explanation on the MBeans, see *WebRTC Session Controller Configuration API Reference*.

About Secure Connections

This section describes some of the security-related features in WebRTC Session Controller.

About Security for Connections Between the Signaling and Media Engine

All communication between the WebRTC Signaling Engine and any Media engine uses the HTTPS protocol. For all media engines that you add to WebRTC Session Controller, ensure that the required SSL certificates are configured and stored in WebLogic Server.

Storing and Managing Certificates in WebLogic Server

When you configure SSL, Oracle recommends using separate keystores for both identity and trust because the identity keystore (holding the private key and associated digital certificate) and the trust keystore (trusted CA certificates) may have different security requirements. It also provides separate tools and procedures for using keystores and certificates in a development environment and a production environment. For more information, see the description about "Configuring Keystores" in *Oracle Fusion Middleware Administering Security for Oracle WebLogic Server*.

For information about managing a new WebRTC Session Controller Media Engine (ME) system, see "Managing and Administering ME Systems".

Disabling the HTTPS Setting in WebLogic Server

If your installation uses a hardened network configuration and need to disable the HTTPS protocol setting, then, use the following system property when you start the WebLogic server, **-Doracle.wsc.se-me-http=true**.

About Security for Connections to Cloud Messaging Providers

WebRTC Session Controller connects to Google Cloud Messaging and Apple Push Notification Service over secure channels.

If you are providing services using APNS, ensure that you access Apple Development Center to create appropriate certificates and install them for use in your WebRTC Session Controller system. The APNS certificates are stored in the **wsc-config.xml** file using Base64 encoding. Passphrases are encrypted and stored in this file using Weblogic encryption mechanism.

When you register your applications, you enter these certificates in WebRTC Session Controller. See "Creating Applications for the Notification Service".

About Security for WebRTC Application Features

This section describes some of the security features in the WebRTC-enabled applications.

RTCDataChannel Interface

The RTCDataChannel interface in WebRTC-enabled applications is secured with Datagram Transport Layer Security (DTLS). DTLS is a derivation of SSL protocol and used to provide communications privacy for datagram protocols and is built-in element of WebRTC Session Controller.

Device Handover

By default, WebRTC Session Controller does not allow session transfers. To support device handovers in your applications, you should enable session transfer validation when starting the WebRTC Session Controller server. Your application must be in charge of session transfer information security, because this data is transmitted between customer application network.

TURN Authorization

The Signaling Engine stores the secret key provisioned on the media engine in the **wsc-config** file using Weblogic encryption mechanism. The credentials are passed to the client over WebSocket. Use Web Services Security (wss) so that these credentials are not sniffed on the network.

About WebRTC Session Controller Console Configuration

The **Home**, **Packages**, and **Script Library** tabs provide access to the configuration settings at the global level. As Table 2–1 shows, the **Home** tab enables you to configure the **Signaling Engine**, **Media Engine**, and the **Notification Service** for WebRTC Session Controller.

Figure 2–1 WebRTC Session Controller Administration Console Configuration Tabs

The values configured in the **Home** and **Packages** tab become the default settings for the individual applications you create in the **Application Profiles** tab.

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 about using these MBeans in *WebRTC Session Controller Configuration API Reference*.

About the Administration Console Configuration Process

To manage WebRTC Session Controller for messaging applications associated with an access account, do the following:

1. Access the configuration tabs in WebRTC Session Controller administration console. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
2. Configure and update the general parameters for WebRTC Session Controller described in each of the following sections:
   - Configuring Default Parameters for WebRTC Session Controller Applications.
   - Configuring Messaging Packages.
   - About the WebRTC Session Controller Global Script Library.
3. Configure and update the specific parameters for each application your account creates, as described in "Managing WebRTC Session Controller Application Profiles".

Accessing the WebRTC Session Controller Console Configuration Tabs

The WebRTC Session Controller console resides in the same domain as your WebRTC Session Controller installation. When you start your domain, both the Oracle WebLogic administration console and the WebRTC Session Controller console become available.

The following procedure requires a running WebLogic server, and that you know the WebLogic user name and password that you created for the domain. See the discussion about "creating and starting a WebLogic domain" in *WebRTC Session Controller Installation Guide*.

To access the WebRTC Session Controller console configuration tabs:

1. Start the WebRTC Session Controller domain server.
2. Open a web browser.
3. Access one of the following URLs, as appropriate.
   - **http://***localhost***:***port***/wsc-console**
   - If HTTP security is configured:

     https://*localhost***:***port***/wsc-console**
   - To start the WebRTC Session Controller console on a local system using the default port:

     http://localhost:7001/wsc-console

   localhost is the IP address of the system running the WebLogic domain or the value **localhost** and *port* is the port of the domain. The default port is **7001**.
4. The WebLogic user login screen appears. Enter the **Username** and **Password** you set when creating the WebLogic domain.
5. Click **Login**.

   The WebRTC Session Controller console window appears. It displays the **Home**, **Packages**, **Script Library**, and **Application Profiles** configuration tabs as seen in Figure 2–1.

About Templates for Message Notifications

Review this section before you configure the notification service supported in your WebRTC Session Controller installation.

As a WebRTC Session Controller system administrator, you can configure application-specific templates that specify the default parameters for the push notification payload for each application. The push notification is made up by combining the message payload provided with the application settings with the message payload received from the application.

A template entry allows WebRTC Session Controller to construct the message correctly based on the cloud messaging provider that the message is targeted for. WebRTC Session Controller determines the format based on the application-id that the browser application sends along with the message.

In the WebRTC Session Controller administration console, there is a **Template** entry for each application profile configured in the **Notification Service tab.** Provide a JSON message as a template for the push payloads, by using this entry.

About the Push Payload Construction for Android Notifications

Example 2–1 shows the contents of the **Template** field for an Android application with the application ID *oracle.mywsc.myandroidapp*:

Example 2–1 Application Template Entry for Android Notification

```
{
  "collapse_key" : "app1",
  "delay_while_idle" : true,
  "time_to_live" : 60,
  "data" : {
      "wsc_time": "$data.time$",
      "wsc_from": "from $data.from$",
      "wsc_event": "Got $data.message$"
   }
}
```

Example 2–2 shows an example message payload that the GCM server sends for the Android application with that application ID *oracle.mywsc.myandroidapp*:

Example 2–2 The Message Payload Received from WebRTC Session Controller

```
{
  "data" : {
    "time": "15:16.2342",
    "from": "alice@example.com",
    "message": "Incoming Call"
  }
}
```

The runtime message payload received from WebRTC Session Controller contains the values for the time (*15:16.2342*), the calling party information (*alice@example.com*), and the event (*Incoming Call*). Any payload from the Groovy layer is only used for merging with the template if any dynamic parameters are specified.

The resulting payload in the push notification received by the called party shows the information merged in the following way:

Example 2–3 The Merged Push Notification (Android)

```
{
  "collapse_key" : "app1",
  "delay_while_idle" : true,
  "to" : "APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...",
```

```
  "time_to_live" : 3,
  "data" : {
    "wsc_time": "15:16.2342",
    "wsc_from": "alice@example.com",
    "wsc_message": "Incoming Call"
  }
}
```

The value for the **"to" :** attribute can be the **registration ID** or the **device token** that receives the notification. In Example 2–3, the value for the **"to" :** attribute is the **registration ID** of the application that was sent to the application when it registered with the GCM server.

About the Push Payload Construction for iOS Notifications

Example 2–4 shows the contents of the **Template** field for an iOS application with the application ID *oracle.mywsc.myiOSapp*:

Example 2–4 Application Template Entry for iOS Notification

```
{"aps" : {
    "alert" : "$data.message$ from $data.from$",
    "badge" : 1,
    "sound" : "chime.aiff"
    }
  }
```

Example 2–5 shows an example message payload that APNs server sends for the iOS application with that application ID *oracle.mywsc.myiOSapp*:

Example 2–5 Message Payload Received from WebRTC Session Controller

```
{
  "data" : {
    "time": "15:16.2342",
    "from": "alice@example.com",
    "message": "Incoming Call"
  }
}
```

The runtime message payload received from WebRTC Session Controller contains the values for the time (*15:16.2342*), the calling party information (*alice@example.com*), and the event (*Incoming Call*). Any payload from the Groovy layer is only used for merging with the template if any dynamic parameters are specified.

Example 2–6 shows the resulting payload in the push notification received by the called party:

Example 2–6 The Merged Push Notification (iOS)

```
"aps" : {
    "alert" : "Incoming Call from alice@example.com"
    "badge" : 1,
    "sound" : "chime.aiff"
  },
  "data" : {
    "time": "15:16.2342",
    "from": "alice@example.com",
    "message": "Incoming Call"
  }
```

Handling Silent Notifications

iOS7.0 and later versions support silent remote notifications where the silent notification wakes up the application in the background so that the application can get new data from the server.

> **Tip:** Use a template that contains application-specific data only. Do not set sound, badge, or alert in the template if you require silent notifications.

Configuring Default Parameters for WebRTC Session Controller Applications

Configure the global (default) parameters for WebRTC Session Controller through the **Home** tab seen in Figure 2–1. The **Home** tab contains three sub tabs, **Signaling Engine**, **Media Engine**, and **Notification Service**.

You can override these defaults when you register your applications using the Application Profile tab and configure the application profiles for your Web, Android of iOS applications.

To configure the default parameters for all the applications, complete the following tasks:

1. Configuring Global Properties for the Signaling Engine.
2. Managing Media Engine Nodes Configuration and Status.
3. Managing WebRTC Session Controller Notification Service.

Configuring Global Properties for the Signaling Engine

The **Signaling Engine** tab displays configuration parameters grouped under **Integration Parameters**, **Runtime Parameters**, **Resource Limits**, and **Log Settings** headings.

To configure signaling engine parameters:

1. If you are not already viewing the **Signaling Engine** tab, access the **Home** tab. See "Accessing the WebRTC Session Controller Console Configuration Tabs".

   By default, **Signaling Engine** is the selected tab.
2. Click **Edit** in the upper right corner of the screen.
3. Alter the contents of each of the following as needed for your environment:
   a. Integration Parameters. For a description of the integration parameters, see Table 2–1.
   b. Runtime parameters. For a description of the runtime parameters, see Table 2–8.
   c. Resource Limits. For a description of the resource limits, see Table 2–3.
   d. Default Log levels. For a description of the default log levels, see "Configuring the Default Logging Level for the Signaling Engine".
   e. Logging levels for a signaling engines in a cluster. For a description of the fields, see "Logging for Single Engines in a Cluster".

4. Click **Save**.

Global Integration Parameters of the Signaling Engine

Table 2–1 describes the signaling engine parameters for integration with the Media Engine.

Table 2–1 Configurable Signaling Engine Integration Parameters

Parameter	Description
Proxy Registrar URI	Enter a SIP proxy server/Registrar URI. The value you enter in this field becomes the default SIP proxy server/Registrar URI for any new application you create. Access this parameter in Groovy as **context.properties.proxyRegistrar** using **SipContext**, **AuthenticationContext**, **TemplateContext**, or **WebContext**.
Dynamic Media Anchoring Type	Select a media anchoring option supported by WebRTC Session Controller. The possible selections are: ■ Disabled The application should not connect to the Media Engine. ■ web-to-web-anchor-conditional web to web conditional anchoring is used in a session when WebRTC-enabled browsers are allowed to communicate directly. If for some reason the browsers cannot communicate directly, they can communicate through WebRTC Session Controller. ■ web-to-web-anchored web to web forced anchoring is used in a session when all media flows through Media Engine. The supported Media Engine session type, is assigned to the Groovy constant ME_CONFIG_NAME_DMA, in the Groovy library
Media Engine MSRP	Select a message Session Relay Protocol (MSRP) to **Media Engine** from **Signaling Engine**. The possible selections are: ■ msrpwss-to-msrptcp ■ msrpws-to-msrptcp Access this parameter in Groovy as **context.properties.webMSRP** using **SipContext**, **AuthenticationContext**, **TemplateContext**, or **WebContext**.
Signaling Engine MSRP	Select an MSRP to **Signaling Engine** from Media Engine. ■ msrptcp-to-msrpwss ■ msrptcp-to-msrpws Access this parameter in Groovy as **context.properties.netMSRP** using **SipContext**, **AuthenticationContext**, **TemplateContext**, or **WebContext**.
File Transfer	Enter a pattern for resolving file_transfer packages by SDP. Access this parameter in Groovy as **context.properties.fileTransferPattern** using **SipContext**, **AuthenticationContext**, **TemplateContext**, or **WebContext**.

Table 2–1 (Cont.) Configurable Signaling Engine Integration Parameters

Parameter	Description
MSRP	Enter a pattern for resolving MSRP packages by the Session Description Protocol (SDP). MSRP signaling is carried in SIP INVITE requests. When WebRTC Session Controller receives a SIP INVITE, it determines whether the request should be processed as a call, msrp chat or msrp file transfer. To do so, it looks at these regex expressions. Access this parameter in Groovy as **context.properties.msrpPattern** using **SipContext**, **AuthenticationContext**, **TemplateContext**, or **WebContext**.

For information about accessing the parameters in Groovy scripts, see "Accessing Integration Parameters and Package Filters Using Groovy Scripts" in *WebRTC Session Controller Extension Developer's Guide*.

Global Runtime Parameters of the Signaling Engine

Table 2–2 describes the configurable signaling engine runtime parameters.

Table 2–2 Configurable Signaling Engine Runtime Parameters

Parameter	Description
Glare Handling	By default, glare handling is selected and enabled. To avoid race conditions that arise when a caller and callee send simultaneous invitations, reinvitations, or session update, select glare handling for the signaling engine.
Sip Session Default Time	Enter the default SIP session time (in seconds). The default value is 3600 seconds.
Sip Session Minimum Time	Enter the minimum SIP session time (in seconds). The default value is 90 seconds.
WebSocket Disconnect Time Limit	Enter the time interval after which the WebSocket times out (in milliseconds). The default value is 60,000 milliseconds.
WebSocket Idle Time Limit	Enter the idle time interval after which the WebSocket times out (in seconds). The default value is 30 seconds.
WebSocket Maximum Connections	Enter the maximum number of WebSocket connections allowed. The default value is **-1**, allowing unlimited connections.

Global Resource Limit Parameters of the Signaling Engine

Set the global resource limits for the resource parameters you include in the **Signaling Engine**.

First, add a resource limit entry for the signaling engine:

1. Click **Edit** in the upper right corner of the screen.
2. In the **Resource Limits** section of the **Signaling Engine** tab, click **Create**.
3. In the entry field within the **Create a resource limit** dialog, enter a resource name.
4. Click **OK**.
5. Click **Save**.

Then, update the resource limit entry for the signaling engine:

1. Click **Edit** in the upper right corner of the screen.

2. In the **Resource Limits** table of the **Signaling Engine** tab, select the row entry for the required resource name.
3. Update the row as described in Table 2–3.

   The numeric entries seen in this table represent the maximum values for each resource limit parameter. The default value is **-1**, indicating that the entry does not have an upper limit.
4. Click **Save**.

Table 2–3 Configurable Signaling Engine Resource Limit Entry Parameters

Resource Parameter	Description
Name	Enter the name of the resource limit.
Sessions	Enter the maximum number of sessions for this resource limit.
SessPerUser	Enter the maximum number of sessions per user for this resource limit.
SubSessPerSess	Enter the maximum number of sub sessions per session for this resource limit.
SubSessPerUser	Enter the maximum number of sub sessions per user for this resource limit.

To delete a resource limit entry for the signaling engine:

Note: The *default* entry cannot be deleted.

1. Click **Edit** in the upper right corner of the screen.
2. In the **Resource Limits** table of the **Signaling Engine** tab, select the row entry for the required resource name.
3. Click **Delete**.
4. Click **Save**.

Configuring the Default Logging Level for the Signaling Engine

You can specify the default logging level for each of the following logging components of the Signalling Engine: Diameter protocol, Groovy scripts, HTTP/WebSocket, JSON, Media, Others, Security, and SIP.

Table 2–4 lists the output associated with each logging level.

Table 2–4 Logging Levels for Signaling Engine Components

Logging Level	Output
Trace	Logs fine-grained events that are useful for tracing the actions of the application.
Debug	Logs fine-grained events that would be useful for debugging the application.
Info	Logs high-level information that indicates the progress of the application.
Warn	Logs messages that describe situations that are potentially harmful.

Table 2–4 (Cont.) Logging Levels for Signaling Engine Components

Logging Level	Output
Default	Assigns the default log level for the component that is specified in the Default log level panel. For example, you set the log level for Groovy to **Default**. If the default log level for Groovy is **Info**, the log level for Groovy is set to **Info**.

Signaling Engine writes the log records to the *domain_home*/servers/*server_name*/**logs/wsc.log** file. Here, *domain_home* is the name of the WebRTC Session Controller domain and *server_name* is the name of the service.

When you select a level heading, all the sliders move to that heading indicating that WebRTC Session Controller logs all the components at that level. To set the logging output for any individual component, move the corresponding slider to the required level. Select from one of the following levels:

To set *default* logging levels for each of the logging components of an engine, use the **Default log level** pane. The engine log settings pane allows you to set the log level for each of its logging components.

To set the logging level for an engine logging component:

1. Click **Edit** in the upper right corner of the screen.
2. In the *enginename* panel, move the slider for the logging component to one of the logging levels listed in Table 2–4.

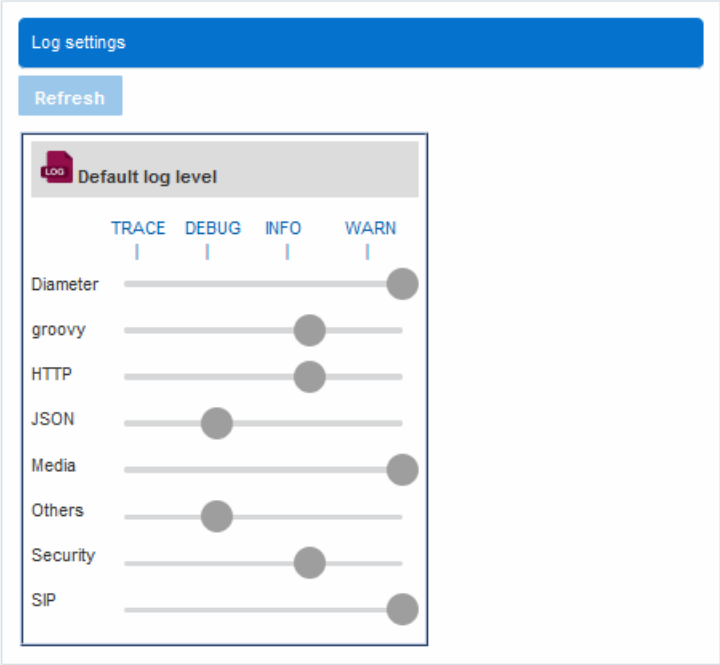
   Repeat this step for each logging component that you wish to set.
3. Click **Save**.

Logging for Single Engines in a Cluster

When signaling engines are in a cluster, the administration console displays the log for each signalling engine adjacent to the **Default log level** pane. Figure 2–2 shows the default logging level as the selection for all the components of *engine1*.

In the *enginename* panel, move the slider for the logging component to one of the logging levels listed in Table 2–4. The **Refresh** button refreshes the engine status for each engine in the cluster.

Figure 2–2 Log Settings Pane



Managing Media Engine Nodes Configuration and Status

Managing Media Engine nodes consists of configuring the media nodes, blocking and unblocking WebRTC network traffic to media nodes, monitoring their availability, and ensuring that their load factor remains within accepted limits. And removing a media node after blocking all traffic to the node.

The **Media Engine** tab displays the **Credentials** section followed by **Cluster Nodes**. Each row in the **Cluster Nodes** table displays the settings of a Media Engine node currently configured in WebRTC Session Controller. The row entries for a Media Engine node display the engine access details, whether it is enabled or not, its running status, and the load factor. An upward-pointing green arrow indicates that the engine is in a running state and a downward-pointing red arrow indicates that it is not.

Configuring the Media Engine

To configure Media Engine parameters:

1. If you are not already viewing the Home tab, access the **Home** tab. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
2. Click **Media Engine**.
3. Click **Edit** in the upper right corner of the screen.
4. In the **Credentials** section of the **Media Engine** tab, enter the information for the account authorized to connect to the Media Engine server:
   a. In the **User** field, enter the user name for the account.
   b. In the **Password** field, enter the password for the account.
   c. Click **Save**.
5. In the **Credentials** section of the **Media Engine** window, manage the Media Engine by doing the following:

a. Adding Media Engine Nodes

b. Blocking and Unblocking Media Node Traffic

c. Removing Media Engine Nodes

d. Refreshing Media Node Information

6. Click **Save**.

Providing Credentials for the Media Server

In the **Credentials** section of the **Media Engine** window, enter the user name and password for accessing the Media Engine server. Click **Save**.

Adding Media Engine Nodes

To add a Media Engine node:

1. Click the **Home** tab.
2. Click the **Media Engine** tab.
3. Click **Edit** in the upper right corner of the screen.
4. In the **Credentials** section of the **Media Engine** tab, enter the information for the account authorized to connect to the Media Engine server:
   a. In the **User** field, enter the user name for the account.
   b. In the **Password** field, enter the password for the account.
   c. Click **Save**.
5. In the **Cluster Nodes** section, click **Add**.
6. In the **Add a Media Engine** dialog:
   a. In the **Address** field, enter the address of the media server node.
   b. In the **Port** field, enter the port number.
   c. Click **OK**.

      The Cluster Nodes table displays a row with the address and port you provided.
7. In that row, configure the Media Engine with the information required.

   See Table 2–5 for a description of the configurable and viewable media node properties.
8. Click **Save**.

Table 2–5 Media Node Properties

Property	Description
User	Enter the user name required to connect to the Media Engine server.
Password	Enter the password require to connect to the Media Engine server.
Address	Enter the IP address of the node associated with the Media Engine.
Port	Enter the port number of the node connection.

Table 2–5 (Cont.) Media Node Properties

Property	Description
Media Node Traffic Enabled	Specify whether traffic is enabled to the node associated with the Media Engine.
Media Node Status	Specify whether a connection to the 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 server nodes. WebRTC Session Controller stops sending requests to media nodes with a Load Factor of 100%.

Blocking and Unblocking Media Node Traffic

To block or unblock traffic to a media node:

1. Go to the **Media Engine** tab.
2. Click **Edit** in the upper right corner of the screen.
3. In the **Cluster Nodes** section, select the row with the media node you wish to block or unblock traffic to.
4. Click **Unblock Traffic** or **Block Traffic**.

   The traffic to the node opens or closes accordingly. The command button you selected continues to stay selected until you change the setting.
5. Click **Save**.

Removing Media Engine Nodes

Block the traffic for a media node before removing the node. To remove a media node:

1. Go to the **Media Engine** tab.
2. Click **Edit** in the upper right corner of the screen.
3. In the **Cluster Nodes** section, select the row with the media node you wish to block or unblock traffic to.
4. Click **Block Traffic**.
5. Click **Remove**.

   The **Remove Media Node** window appears.
6. Click **OK**.
7. Click **Save**.

Refreshing Media Node Information

To refresh media node information:

1. Click **Refresh** in the **Media Engine** window.

For information about configuring the nodes of a Media Engine server, see "Managing Media Engine Nodes Configuration and Status".

Managing WebRTC Session Controller Notification Service

WebRTC Session Controller Notification Service implements the protocol specific to the selected cloud messaging provider, such as Google Cloud Messaging system

(GCM), Apple Push Notification System (APNS). This notification service is available on all engine nodes.

You can configure multiple client applications for various client platforms on the Notification Service and activate them at the same time. If your installation supports multi-tenancy, each tenant can configure multiple applications on the various client platforms using same or different credentials on the cloud messaging provider.

Manage the notification service by defining and updating the properties of your client applications in the **Notification Service** tab. This tab displays a table of the client applications currently configured for the notification service.

Configuring WebRTC Session Controller Notification Service

To configure the notification service for WebRTC Session Controller:

1. If you are not already viewing the **Home** tab, access the **Home** tab. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
2. Click **Notification Service**.
3. Click **Edit** in the upper right corner of the screen.
4. In the **Notification Service** tab, manage the notification service by completing the following tasks.
   a. Creating Applications for the Notification Service
   b. Updating an Application in the Notification Service
   c. Removing Applications from the Notification Service
5. Click **Save**.

Creating Applications for the Notification Service

Manage an application that uses the Notification Service by adding your applications and entering their configuration parameters. See Table 2–6 for a description of the parameters. You should have the JSON message to enter as template entry for an application as described in About Templates for Message Notifications.

Do the following:

Add the Application to the Client Applications Table

1. Go to the **Notification Service** tab.
2. Click **Edit** in the upper right corner of the screen.
3. In the **Client Applications** window, click **Create**.
4. In the **Add an Application** window:
   a. Enter an application ID and a unique name for the application. See Table 2–6 for details.
   b. Click **OK**.
5. To save your updates thus far, click **Save**.

Configure the JSON Message as a Template

1. If you are not in the Edit mode, click **Edit** in the upper right corner of the screen.
2. From the list of application entries, select the row for the application.
3. In the **Template** field:

a. Click the **Add** link.

   The **Enter JSON Template** window is displayed.

b. Copy your JSON message and paste it in this window.

c. Click **OK**.

4. To save your updates thus far, click **Save**.

Enable Traffic to Access the Application

1. If you are not in the Edit mode, click **Edit** in the upper right corner of the screen.
2. From the list of application entries, select the row for the application.
3. In the **Enable** field for the selected application entry, select the check box, to allow traffic.
4. To save your updates thus far, click **Save**.

Configure the Cloud Providers

1. If you are not in the Edit mode, click **Edit** in the upper right corner of the screen.
2. From the list of application entries, select the row for the application.
3. In the **Cloud Providers** field, click the **Add** link.

   A list of supported cloud providers is displayed.
4. From the list of cloud providers, select the cloud provider.
5. Click **OK**.
6. To save your updates thus far, click **Save**.

Provide the API Key for Non-iOS Applications

1. If you are not in the Edit mode, click **Edit** in the upper right corner of the screen.
2. From the list of application entries, select the row for the application.
3. Place your cursor over the **API Key** entry field and double-click.

   The cursor is active in the field.
4. Enter the API key you obtained from the Google Developers console.
5. To save your updates thus far, click **Save**.

Add an SSL Certificate

1. If you are not in the Edit mode, click **Edit** in the upper right corner of the screen.
2. From the list of application entries, select the row for the application.
3. In the **Certificate** field for the application entry, click the **Add** sign.
4. In the **Certificate** window, click **Create**.
5. In the **Add certificate** window, click **Browse**.
6. In the **File Upload** window, locate the certificate and click **Open**.
7. In the **Certificate Name** field, enter the name for the certificate.
8. In the **Certificate Password** field, enter the certificate password.
9. Click **OK**.
10. Click **Save** in the upper right corner of the screen.

When the application data is saved successfully, the **Client Applications window** becomes disabled and the **Edit** button is enabled.

Client Application Configuration settings

Table 2–6 describes the properties for the client applications you define in the WebRTC Session Controller console.

Table 2–6 Configuring Support for Notification Service

Property	Description
Application ID	The unique identifier for the Android or iOS application registering to receive messages. Enter the identifier in the format appropriate for the application. For example, the naming convention for an entry could be: ■ *com.android.hello* for an Android application ■ *com.ios.hello* for an iOS application
Name	Name of the application. Enter a unique name.
Template	JSON message to be used as a template. The WebRTC Session Controller server combines this message with the message payload received from the application to create the push notification. See "About Templates for Message Notifications".
Enable	The API service is disabled, by default. To enable this API, select the check box.
Cloud Providers	The cloud provider for the API service. Click the down arrow and select the required cloud provider.
API Key	The API Key you obtained from the **Credentials** page of the Google Developers console. **Note**: This entry is not required for iOS applications.
Certificate	Add an SSL certificate for Apple Push Notification Server. See "About Secure Connections". **Note**: This entry is optional for Android applications.

Updating an Application in the Notification Service

To update an application listed in the Notification service:

1. Go to the **Notification Service** tab.
2. Click **Edit** in the upper right corner of the screen.
3. In the **Client Applications** table, select the row with the application entry you wish to update.
4. Click **Edit** in the upper right corner of the screen.
5. Edit the contents of that row. For information about how to delete an SSL certificate, see "Deleting an SSL Certificate".

> **Note:** You cannot update the entry in the Application ID field.

6. Do one of the following:
   - To save your edits, click **Save** in the upper right corner of the screen.

- To discard your edits, click **Cancel** in the upper right corner of the screen.

Removing Applications from the Notification Service

To remove an application from the displayed table:

1. Go to the **Notification Service** tab.
2. Click **Edit** in the upper right corner of the screen.
3. In the **Client Applications** table, select the row with the application entry you wish to delete.
4. Click **Edit** in the upper right corner of the screen.
5. Click **Delete**.
6. In the **Delete an application** window, click **OK**.
7. Click **Save** in the upper right corner of the screen.

Deleting an SSL Certificate

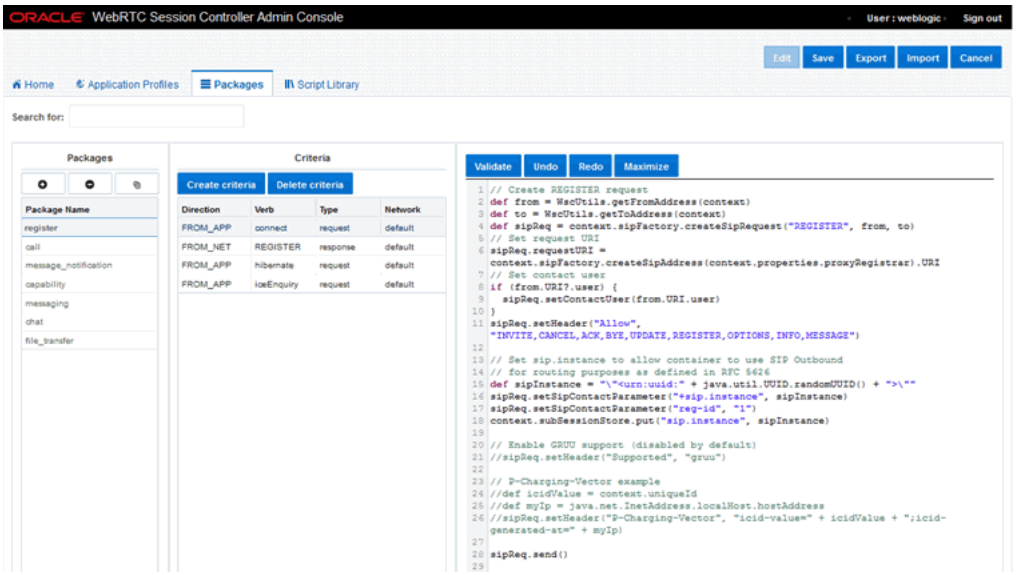
To delete a certificate for an application:

1. If you are not already viewing the Home tab, access the **Home** tab. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
2. Click **Notification Service**.
3. In the **Client Applications** table, select the row with the application entry from which to delete the SSL certificate.
4. Click **Edit** in the upper right corner of the screen.
5. In the **Certificate** field for the application entry, click the **Add** sign.
6. In the **Certificate** window, select the certificate to delete.
7. Click **Delete**.
8. To save your changes, click **Save**.

Configuring Messaging Packages

A package is a collection of all the criteria (Groovy scripts) necessary to translate the telecom messages in a session between JSON to SIP protocols. The **Packages** tab at the global level enables you to configure the messaging packages that WebRTC Session Controller installation supports. Figure 2–3 shows an example.

Figure 2–3 Global Configuration for Supported Messaging Packages



In this figure, the **Search** field contains the entry "sipRequest". The display shows the result of the filter "sipRequest". The table under **Package Name** lists the packages that contain this string. The **Criteria** table lists the criteria configured for the selected package, **call**. The **Groovy Script** section displays the content of the criteria row entry selected in the **Criteria** table for the selected package.

About the Global Packages Tab

The **Packages** Tab at the global level is made up of three panes:

- **Packages**

  The **Packages** pane in the Packages tab displays three icons and a table below them. The table displays the names of the default packages that are currently in the database and created by the current user account. The three icons are used to create, delete, and clone a package.

  The packages that you create at this level display as selections when you create an application profile (accessed through the **Application Profiles** tab).

- **Criteria**

  The **Criteria** pane displays the possible message criteria defined for the selected package.

  Each signaling engine criterion forms a single Groovy script that performs the translation and processing tasks for a single type of JSON or SIP message. Create separate criteria for all possible JSON or SIP message that your signaling engine implementation processes. In synchronous request/response communication, create a separate criteria for each request and response message.

  Criteria are applied to messages based on this information included in each criteria. Table 2–7 describes the properties for the request and response messages you define in the WebRTC Session Controller administration console.

- **Groovy Script**

  When you select a row in the **Criteria** table, WebRTC Session Controller displays the Groovy version of this message criteria. Depending on your browser

allowance, the **Groovy Script** pane displays to the right of or below the **Criteria** table.

To validate the displayed Groovy script, undo, or redo your last action, use the **Validate**, **Undo**, **Redo** command buttons respectively. To maximize or minimize the viewing of this pane, use the **Maximize** or **Minimize** command buttons.

Creating Packages

When you create a new package WebRTC Session Controller just creates a shell that you fill with criteria. This procedure assumes that you have already created the criteria required.

To create a package:

1. If you are at the **Packages** tab of the WebRTC Session Controller administration console:
   a. Access the administration console configuration tabs. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
   b. Click the **Packages** tab.
2. In the **Packages** pane of the **Packages** tab, click the **Add** icon.
3. Click **Edit** in the upper right corner of the screen.
4. In the **Create a package** dialog box:
   a. Enter a name for the new package.

      Package names must be unique in the list. A package name must begin with an alphabet (a - z) or a number (0 - 9).

   b. Click **OK**.

   Your new package now appears in the package table which is arranged alphabetically.
5. To make your changes take effect, click **Save**.

You are now ready to configure and manage the package.

Managing Package Criteria

To manage the package criteria for an existing package:

1. Go to the **Packages** tab.
2. From the list of packages under **Package Name**, select the package.
3. Click **Edit** in the upper right corner of the screen.
4. In the **Packages** tab, manage the package by doing the following:
   a. Configuring Package Criteria.
   b. Updating Package Criteria.
   c. Deleting a Criteria.
5. Click **Save**.

Configuring Package Criteria

To create a signaling engine criteria for a selected package:

1. Click **Edit** in the upper right corner of the screen.
2. Click **Create Criteria**.

   A row appears at the top of the criteria table. A Groovy script pane appears next to or below the criteria table, depending on the browser size.
3. Provide the values for the **Direction**, **Verb**, **Type**, and **Network** fields for this criteria. See Table 2–7 for a description of the fields.
4. In the **Groovy Script** window, enter a Groovy script as appropriate for this criteria.

   The Groovy script defines all actions for this criteria. See "About the Groovy Scripts" in *WebRTC Session Controller Extension Developer's Guide*.
5. To ensure that your Groovy script meets the signaling engine validation requirements, click **Validate**. Fix all errors.

   See "About the WebRTC Session Controller Console Validation Tests" for a list of the validation tests and error messages.
6. To save your new criteria, click **Save**. For details, see "About the Groovy Scripts" in *WebRTC Session Controller Extension Developer's Guide*.

Table 2–7 Configuring Support for Notification Service

Property	Description
Direction	This entry indicates the message origin. Select from one of the following: ▪ **FROM_APP** for messages that originate from a WebRTC-enabled browser. ▪ **FROM_NET** for messages originating from your IMS core (SIP server or proxy). ▪ **SYSTEM** for messages that originated in a WebRTC Session Controller server
Verb	A verb matching the type of JSON or SIP request or response. For example, an UPDATE verb matches SIP UPDATE requests and response messages, and a complete verb matches a JSON complete request or response message. Enter the verb that identifies the message action verb (SIP method or JSON action) that the criteria matches. For example, REGISTER, PUT, GET, ACK, BYE, CANCEL, INVITE, complete, shutdown,
Type	The type of message; can be one of: ▪ **request** ▪ **response** ▪ **message** ▪ **error** ▪ **acknowledgment**
Network	Identifies the application that the message is being used for.

Updating Package Criteria

To update the contents of a package criteria:

1. Go to the **Packages** tab in the administration console.
2. From the list of packages under **Package Name**, select the package.

3. Click **Edit** in the upper right corner of the screen.
4. From the **Criteria** table, select the row entry for the criteria.
5. Do one or both of the following, as required:
   - Update the **Direction**, **Verb**, **Type**, and **Network** fields for this criteria. See Table 2–7 for a description of the fields.
   - Update the Groovy Script.
6. To verify the signaling engine validation requirements, click **Validate**. Fix all errors.
7. To save the updated criteria, click **Save**.

Deleting a Criteria

To delete a package criteria:

1. Go to the **Packages** tab in the administration console.
2. From the list of packages under **Package Name**, select the package.
3. From the **Criteria** table, select the row entry for the criteria.
4. Click **Edit** in the upper right corner of the screen.
5. Click **Delete Criteria**.
6. To save your updates, click **Save**.

About the WebRTC Session Controller Global Script Library

The **Script Library** tab displays the contents of the mandatory WebRTC Session Controller functions that are required to handle message processing.

> **Important:** The signature of these methods must not be changed, but the implementation can be modified according to your requirements.

In the **Edit** mode, use the **Validate**, **Undo**, **Redo** command buttons to validate the displayed Groovy script, undo, or redo your last action in this pane.

Managing WebRTC Session Controller Application Profiles

An application profile that you create in WebRTC Session Controller is a collection of packages. Each package contains criteria that translate (and probably change) WebRTC application to SIP network communication for a single program.

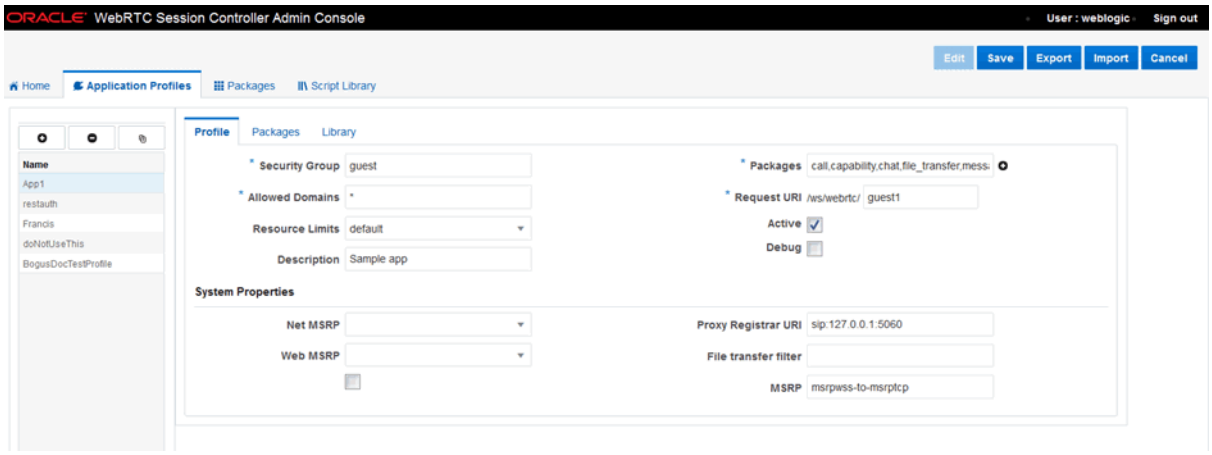
In the WebRTC Session Controller administration console, set up the application profile by selecting the packages from the set of globally defined packages. Provide information to identify the application profile and by defining other parameters such as a security group, domains, and resource limits.

Each application profile contains its set of Groovy scripts. After you change the configuration and save the changes, the modified Groovy is saved in the configuration file. At runtime, WebRTC Session Controller generates a copy of the complete configuration containing each profile data combined with the common Groovy script (packages and script library) data.

About the Application Profiles Tab

The **Application Profiles** tab enables you to configure the application profiles that your WebRTC Session Controller installation supports. Figure 2–4 displays the contents of this tab.

Figure 2–4 Application Profiles Tab



When you select the Application Profiles tab, it displays two panes. To the left, a navigation pane lists the names of the currently created application profiles. To the right is a larger pane with three tabs, **Profile**, **Packages**, and **Library**. These tabs display the general profile information, the Groovy scripts for the packages selected for this application profile and the general script library, respectively.

Managing Application Profiles

The application profile navigation pane consists of a table of application profile names currently configured in WebRTC Session Controller. Three icons above the **Name** table allow you to add, delete, or clone an application.

Creating Your Application Profile

To create an application profile:

1. If you are not logged in to WebRTC Session Controller, access the administration console configuration tabs. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
2. Click **Application Profiles**.
3. Click **Edit** in the upper right corner of the screen.
4. In the navigation panel to the left, click the **Add** link.
5. In the **Create an application profile** dialog window, enter a name for the application profile.

   Application profile names must unique, begin with an alphabet character, and contain alphanumeric characters only (a-z, A-Z, and 0-9).

   Click **OK**.
6. Complete the configuration for the application profile by completing the following tasks:

- Providing the Profile Information for the Application
- Managing the Groovy Script for the Application Profile

7. Click **Save**.

Providing the Profile Information for the Application

Some fields in the **Profile** tab display the default values defined earlier in the **Home** tab. Fields marked with asterisks require a valid entry. You can modify the default values also.

Enter or update the following information associated with the selected application:

1. General runtime parameters to associate with your application profile, described in Table 2–8.

Table 2–8 Configurable Signaling Engine Runtime Parameters

Parameter	Description
Security Group	A required field. Enter the name of the security group for this application. WebLogic Server contains some default security groups that you can use. For details about using security groups, see the discussion on users, groups, and security roles in *Oracle Fusion Middleware Securing Resources and Policies for Oracle WebLogic Server*.
Allowed Domains	A required field containing a list of allowed domains that serve as a white list of domains the application is allowed to contact. Enter all domains to allow cross-origin resources sharing (CORS) with this application. Enter the names of the allowed domains. For a tenant of a SaaS application, a specific web domain name. To support cross domain access, configure the domain name in the tenant profile. WebRTC Session Controller would apply this setting on a per tenant profile basis.
Resource Limits	Select the resource limits that allow you to protect system performance by limiting the impact on **Signaling Engine**. These resource limits can also serve as application white- and black-lists for individual applications. To view the configured parameters for each resource limit, select the **Signaling Engine** tab under the **Home** tab.
Description	Enter a short description of the application profile.
Packages	The messaging packages this application supports. See "Managing Packages in Your Application Profile".
Request URI	Enter the URI endpoint that you want WebRTC applications to use to access WebRTC Session Controller. This entry is the value configured for the **Guest URl Match Pattern** field located on the **Provider Specific tab** for the associated authentication provider in WebLogic Server Administration Console.
Active	The application setting. Active, by default. To deactivate the application, clear the check box.
Debug	To run the application in a debug mode, select this check box.

2. System properties to associate with your application profile, described in Table 2–9.

Note: The values you assign to a system property on the **Application Profile** tab overrides the value assigned to that property on the **Home** tab.

Table 2–9 Configurable Signaling Engine System Properties

Parameter	Description
Proxy Registrar URI	The default SIP proxy server/Registrar URI. The value you enter in this field becomes the default for any new application you create. Access this parameter in Groovy as **context.properties.proxyRegistrar** using **SipContext**, **AuthenticationContext**, **TemplateContext**, or **WebContext**.
Dynamic Media Anchoring Type	Select a media anchoring option supported by WebRTC Session Controller. The possible selections are: ■ Disabled The application cannot connect to the Media Engine. ■ web-to-web-anchor-conditional web to web conditional anchoring is used in a session when WebRTC-enabled browsers are allowed to communicate directly. If for some reason the browsers cannot communicate directly, they can communicate through WebRTC Session Controller. ■ web-to-web-anchored web to web forced anchoring is used in a session when all media flows through Media Engine. The supported Media Engine session type, is assigned to the Groovy constant ME_CONFIG_NAME_DMA, in the Groovy library
File Transfer Filter	Pattern for resolving file_transfer packages by SDP. Access this parameter in Groovy as **context.properties.fileTransferPattern** using **SipContext**, **AuthenticationContext**, **TemplateContext**, or **WebContext**.
Net MSRP	A required field. The WebLogic Server contains some default security groups that you can use. For details about using security groups, see the discussion on users, groups, and security roles in *Oracle Fusion Middleware Securing Resources and Policies for Oracle WebLogic Server*.
Web MSRP	A required field. A list of allowed domains that serve as a white list of domains the application is allowed to contact.
MSRP	Select the pattern for resolving MSRP packages by the Session Description Protocol (SDP). MSRP signaling is carried in SIP INVITE requests. When WebRTC Session Controller receives a SIP INVITE, it determines whether the request should be processed as a call, msrp chat or msrp file transfer. To do so, it looks at these regex expressions. Access this parameter in Groovy as **context.properties.msrpPattern** using **SipContext**, **AuthenticationContext**, **TemplateContext**, or **WebContext**.

3. Click **Save**.

Managing Packages in Your Application Profile

To map messaging packages to your application profile:

Mapping Packages

1. If you are not in the **Profile** tab under for the application entry:
   a. Access the administration console configuration tabs. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
   b. Click **Application Profiles**.
   c. Select the application name.
2. Click **Edit** in the upper right corner of the screen.
3. For the **Packages** field, click the **Add** sign next to the field.

   The **Package mapping** dialog displays the names of the available packages.
4. In the **Package mapping** dialog, do the following for each package your application is to support:
   a. To include a package, select the check box next to its name.
   b. If using an alias, then, in the **Alias** column, enter an alias.

      For example, an alias entry *myFam* for a call package *call* displays as *call:myFam*.

      Note: The alias name can be mapped to a different package without a need for any change to the internal code of the client. For example:

      You have a client names *Aimee.com* that uses the call package named *call*. You create a special package for *Aimee.com*, named as *callAmee*. In this step, you use this alias entry to create a map between *call* and *callAmee*.

      In the run environment, for all the call packages of *Aimee.com*, the signaling engine will use the package *callAmee*.

      ---

   c. Click **OK**.
5. Click **Save**.

Removing a Package from the Mapped List

1. Go to the **Profile** tab for the application entry,
2. Click **Edit** in the upper right corner of the screen.
3. Click the **Add** sign next to the **Packages** field for your application profile.

   The **Package mapping** dialog displays the names of the available packages.
4. To delete the package, clear the check box.
5. Click **OK**.
6. Click **Save**.

Managing the Groovy Script for the Application Profile

Manage the Groovy script for the application profiles using the **Package** tab associated with each application profile entry.

Validating the Scripts for the Selected Packages

1. If you are not in the **Profile** tab under for the application entry:
   a. Access the administration console configuration tabs. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
   b. Click **Application Profiles**.
   c. Select the application profile name entry.
2. Click **Edit** in the upper right corner of the screen.
3. Click **Packages**.
4. To view the scripts, click **Populate**.

   WebRTC Session Controller displays all the default functionality in the Groovy scripts associated with the selected packages. The selected criteria are displayed in the form of methods. You can uncomment an entry and enter the specific overriding functionality and save the changes.

   At runtime, the signaling engine calls the updates package script.
5. Update the script as appropriate.
6. Click **Validate**. Fix all errors using the **Undo** and **Redo** buttons in this pane.
7. Click **Save**.

Validating the Global Library Information for the Application Profile

1. If you are not in the **Profile** tab under for the application entry:
   a. Access the administration console configuration tabs. See "Accessing the WebRTC Session Controller Console Configuration Tabs".
   b. Click **Application Profiles**.
   c. Select the application profile name entry.
2. Click **Edit** in the upper right corner of the screen.
3. Click **Library**.
4. To view the global scripts, click **Populate**.

   The **Library** tab displays the default functionality of the WebRTC Session Controller library. These library methods are required to handle message processing for the packages associated with this application profile. They are displayed as commented methods. You can uncomment an entry and enter the specific overriding functionality and save the changes.

   At runtime, the signaling engine calls the updates package script.
5. Update the script as appropriate.
6. Click **Validate**. Fix all errors using the **Undo** and **Redo** buttons in this pane.
7. Click **Save**.

Exporting and Importing a Configuration

You can export your current configuration settings to a file or import a set of configuration settings from a file to which a configuration instance was previously saved.

To export a configuration:

1. Click **Edit** in the upper right corner of the screen.
2. Click **Export**.

   The configuration is exported to the **wsc-config.xml** file in your system directory for downloads.

To import a saved configuration:

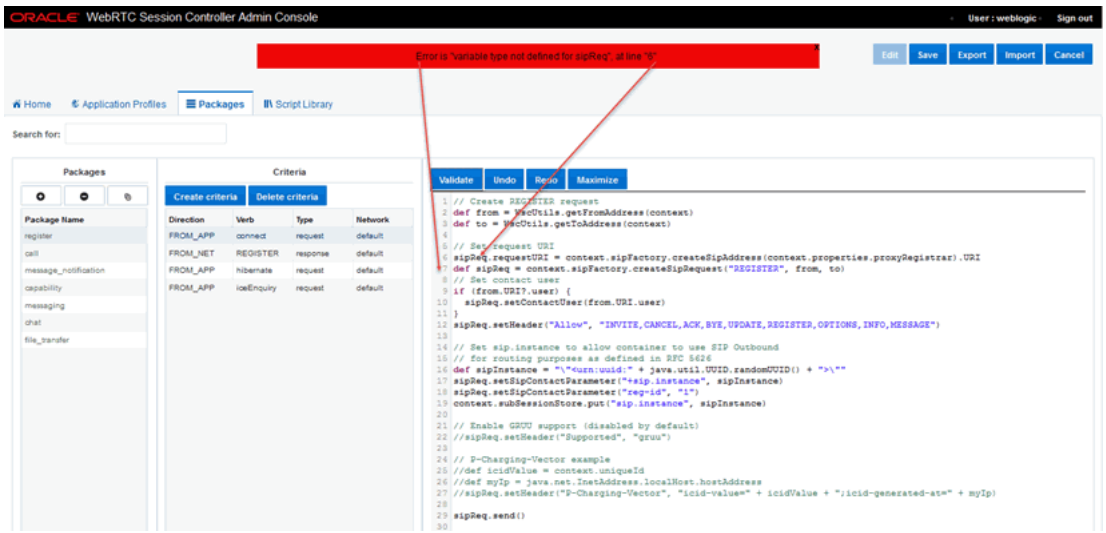
1. Click **Edit** in the upper right corner of the screen.
2. Click **Import**.
3. In the **Import** dialog, choose the XML file that contains the configuration that you wish to import.
4. Click **OK**.

Debugging Groovy Script Run Time Errors

You can diagnose Groovy script problems using the stack trace in the *domain_home*/**wsc.log** file, which contains Signaling Engine stack trace messages. You identify the individual Groovy script by searching for the individual criteria method name that contains the criteria information. See "About the Groovy Scripts" for details about the signature that each script uses, and the method it calls.

Figure 2–5 shows an illustration of a formatting problem in the **register** package, in the **FROM_APP/connect/request/default** criteria shown highlighted. The red arrows show the problem, the **sipReq** variable is used *before* it is declared.

Figure 2–5 Groovy Script With a Syntax Error



This is a violation of Java syntax, and as you would expect, the operation failed and these debugging messages were written to the **wsc.log** file:

```
Caused by: groovy.lang.MissingPropertyException: No such property: sipReq for
class: Script2
     at
org.codehaus.groovy.runtime.ScriptBytecodeAdapter.unwrap(ScriptBytecodeAdapter.jav
a:50)
     at
org.codehaus.groovy.runtime.callsite.PogoGetPropertySite.getProperty(PogoGetProper
```

```
tySite.java:49)
     at
org.codehaus.groovy.runtime.callsite.AbstractCallSite.callGroovyObjectGetProperty(
AbstractCallSite.java:231)
     at Script2.pkg_register_dir_FROM_APP_typ_request_verb_connect_netsvc_
default(Script2.groovy:705)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
     at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
43)
     at java.lang.reflect.Method.invoke(Method.java:606)
     at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:90)
     at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:233)
     at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:1085)
     at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:952)
     at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:909)
     at groovy.lang.Closure.call(Closure.java:411)
     at
org.codehaus.groovy.jsr223.GroovyScriptEngineImpl.callGlobal(GroovyScriptEngineImp
l.java:411)
     at
org.codehaus.groovy.jsr223.GroovyScriptEngineImpl.callGlobal(GroovyScriptEngineImp
l.java:405)
     at
org.codehaus.groovy.jsr223.GroovyScriptEngineImpl.invokeImpl(GroovyScriptEngineImp
l.java:394)
     ...
```

The package and criteria values of the offending Groovy script are identified in this line from **wsc.log:**

```
Script2.pkg_register_dir_FROM_APP_typ_request_verb_connect_netsvc_
default(Script2.groovy:705)
```

This message shows the method name that Signaling Engine creates from each criteria's package name and criteria values (highlighted).

In this example, the criteria with the syntax error is in the **register** package. Within that package the criteria with the problem has a **FROM_APP** direction; a **request** type; a **connect** verb; and a **default** network service. This matches the syntax error of the **FROM_APP/connect/request/default** criteria shown in Figure 2–5.

In addition, as you are developing your Groovy scripts, you can validate the scripts by clicking the **Validate** button in the Groovy Script editor pane. Syntax errors and other issues are reported in the console. Figure 2–5 shows the validation error message.

About the WebRTC Session Controller Console Validation Tests

The WebRTC Session Controller console runs validation tests to confirm that your Groovy scripts, Groovy library, packages, and applications are all valid. It runs the validation tests each time you commit changes to an application, package, or criteria, or click the **Validate** button.

Table 2–10 lists the validation error types and their error messages.

Table 2–10 WebRTC Session Controller Groovy Script Validity Tests

Error Type	Error Message
APPLICATION_NAME_NOT_UNIQUE	Application name *application_name* is not unique.
DUPLICATE_CRITERIA	Duplicate criteria found for criteria *direction*, *verb*, *type*, *network_service* in package *package_name*.
DUPLICATE_LWNG_APPLICATION_ID	Notification Service Application Id *app_id* is not unique.
DUPLICATE_LWNG_APPLICATION_NAME	Notification Service Application name *app_name* is not unique.
DUPLICATE_LWNG_CERTIFICATE_NAME	Certificate name *cert_name* is not unique.
DUPLICATE_PACKAGE_NAME_IN_APP	Package name *package_name* is not unique in application profile *application_name*.
DUPLICATE_TENANT_NAME_IN_APP	Tenant name *tenant_name* is not unique in application profile *app_name*.
EMPTY_LWNG_API_KEY	API KEY should be defined for Notification Service Application with id *app_id*.
EMPTY_LWNG_APPLICATION_ID	Notification Service Application Id should be defined for Notification Service Application with name *app_name*.
EMPTY_LWNG_APPLICATION_NAME	Notification Service Application name should be defined for Notification Service Application with Id *app_id*.
EMPTY_LWNG_CERTIFICATE	Name should be defined for Notification Service Certificate.
EMPTY_LWNG_CERTIFICATE_CONTENT	Certificate Content should be defined for Notification Service certificate with name *certificate_name*.
EMPTY_LWNG_CERTIFICATE_NAME	Certificate name should be defined for Notification Service Application with Id *app_id*.
EMPTY_LWNG_CERTIFICATE_PASSPHRASE	Passphrase should be defined for Notification Service certificate with name *certificate_name*.
EMPTY_LWNG_PROVIDER	Provider name should be defined for Notification Service Application with Id *app_id*.
EMPTY_LWNG_TEMPLATE	Template should be defined for Notification Service Application with is *app_id*.
GROOVY_APPLICATION_LIBRARY_COMPILATION_ERRORS	Application Library validation error, *reason* for the application profile *application_name* at line *line_number*.
GROOVY_APPLICATION_PROFILE_PKG_COMPILATION_ERRORS	Application profile validation error, *reason* for the application profile *application_name*, method *method_name*, error *reason* at line *line_number*.
GROOVY_APPLICATION_PROFILE_RESTRICTIONS	Error occurred in application profile *application_name*.
GROOVY_COMPILATION_ERRORS	Error is *reason*, at line number *line_number*.
GROOVY_SCRIPT_LIBRARY_COMPILATION_ERRORS	Script Library validation error, *reason* at line *line_number*.
GROOVY_SCRIPT_LIBRARY_RESTRICTIONS	Error occurred in script library *scriptlibrary_name*.
GROOVY_SCRIPT_RESTRICTIONS	Error message is *reason*.
GROOVYALL_COMPILATION_ERRORS	Error occurred for the criteria *direction*, *verb*, *type*, *network_service* in the package*package_name* and the error message is *reason* at line *line_number*.
GROOVYALL_SCRIPT_RESTRICTIONS	Error occurred for the criteria *direction*, *verb*, *type*, *network_service* in the package*package_name* and the error message is *reason*.

Table 2–10 (Cont.) WebRTC Session Controller Groovy Script Validity Tests

Error Type	Error Message
INVALID_ALLOWED_DOMAINS_ EXPRESSION	Allowed domain contains invalid characters *domains_expression*.
INVALID_APP_NAME	Application profile name contains invalid characters, *application_ profile_name*.
INVALID_CRITERIA_NETWORK	The package *package_name* network contains invalid characters.
INVALID_CRITERIA_TYPE	The package *package_name* type contains invalid characters.
INVALID_CRITERIA_VERB	The package *package_name* verb contains invalid characters.
INVALID_LWNG_APPLICATION_ID	Notification Service Application Id should be in a valid Android Package or Apple Bundle format for Notification Service Application with Id *app_id*.
INVALID_LWNG_CERTIFICATE_CONTENT	Notification Service Certificate content is not a valid p12 file or the password is incorrect for certificate with name *cert_name*.
INVALID_LWNG_CERTIFICATE_NAME	Certificate name *certificate_name* is referenced from application with Id *app_id*, but it does not exist.
INVALID_LWNG_DELETE_CERT_CMD	Notification Service Certificate *cert_name* is being used by *app_ name*. Please update these apps before deleting the Certificate.
INVALID_LWNG_PROXY_IP	Notification Service Proxy IP *proxy_ip_address* should be in the format 001.001.001.
INVALID_LWNG_PROXY_PORT	Notification Service Proxy Port *port_number* should be in the range of 1 to 65535.
INVALID_LWNG_TEMPLATE	Template *template_name* is not a valid JSON formatted string for application with Id *app_id*.
INVALID_MODULE_NAME	The *module_name* module contains invalid characters.
INVALID_MODULE_REFERENCE	Nonexistent module *module_name* is referenced from application profile *application_name.*
INVALID_NETWORK	The *network_name* network contains invalid characters.
INVALID_PACKAGE_REFERENCE	Package name *package_name* is referenced from application profile *application_name* but it does not exist.
INVALID_PKG_NAME	The *package_name* package contains invalid characters.
INVALID_PROXY_URI_PATTERN	Invalid proxy registrar URI *uri*.
INVALID_REQUEST_URI	The request URI contains invalid characters *request_uri* for the application profile, *application_name*.
INVALID_RESOURCE_LIMITS_NAME	Invalid resource name *resource_name* in *application_name application* application profile.
INVALID_RESOURCE_NAME	Resource contains invalid characters, *resource_name*.
INVALID_SCRIPT_TYPE	The script *script_name* contains an error of type *error_type*.
INVALID_SECURITY_EXPRESSION	Security group contains invalid characters *security_expression*.
INVALID_TENANT_PROFILE_NAME	Tenant profile name contains invalid characters *tenant_name*.
INVALID_TENANT_PROFILE_REFERENCE	Tenant name *tenant_name* is referenced from application profile *app_name* but it does not exist.
INVALID_TYPE	The package *package_name* type contains invalid characters *type*.
INVALID_VERB	The package *package_name* verb contains invalid characters *verb*.
ME_ADDRESS_IS_NOT_UNIQUE	Media Engine address *address* is not unique.

Table 2–10 (Cont.) WebRTC Session Controller Groovy Script Validity Tests

Error Type	Error Message
ME_INVALID_LOGIN	Invalid credentials found, for the Media Engine.
ME_LIFE_CYCLE_ERROR	Media Engine life cycle is not initiated for the server *server_name* and port *port_number*.
ME_NODE_NOT_REACHABLE	Server not reachable with address *address* and port *port_number*.
ME_PORT_IS_INVALID	Invalid media engine port *port_number*.
MODULE_NAME_NOT_UNIQUE	Module name *module_name* is not unique.
PACKAGE_NAME_NOT_UNIQUE	Package name *package_name* is not unique.
PKG_REQUIRED	At least one package is required for the application profile *application_name*.
REQUEST_URI_UNIQUE	Request URI *request_uri* is not unique in application profile *application_profile_name*.
RESOURCE_LIMITS_PROFILE_NAME_IS_CANNOT_DELETE	Cannot delete resource limits *resource_limit_profile_name*, as it is used by application profile(s) *application_name*.
RESOURCE_LIMITS_PROFILE_NAME_IS_NOT_UNIQUE	Resource limits profile name *resource_limit_profile_name* is not unique.
RESTRICTED_GROOVY	Error occurred in restricted groovy script at line *line_number*.
SCRIPT_NAME_NOT_UNIQUE	Script name *script_name* is not unique for the *package_name* package.
TENANT_PROFILE_NAME_NOT_UNIQUE	Tenant profile name *tenant_name* is not unique.

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	Sub Tab	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, report 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	Sub Tab	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	Call State Storage	View call state Coherence cache service configuration settings supported by the Sip Server. You can specify the number of worker threads and the number of partitions used in the call-state Coherence cache service by the Sip Server.
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 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 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	Call State Storage	View run-time state and statistics information about call-state service, the call-state cache, and the call-state metadata cache used by the SIP server.

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 the 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.

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

Method	Description
`save()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to a temporary configuration file.
`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 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.
`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 Name	Description
`commit`	Commits a configuration update transaction.
`exportConfiguration`	Exports the whole **wsc** configuration, to a specified path.
`getCurrentLock`	Gets current lock if one exist. The request fails if the lock is owned by another user.
`importConfiguration`	Imports the whole **wsc** configuration from the specified path.
`isLocked`	Checks if the configuration is locked (by any user).
`lockAndEdit`	Begins the configuration update transaction.
`revert`	Reverts a configuration update transaction.
`validate`	Validates the transaction.
`validateAllScripts`	Validates all the scripts.
`validateMediaEngines`	Validates all the Media Engines and checks to see if they are reachable from the Signaling Engine.
`validateScript`	Validates a particular script.
`validateScriptLibrary`	Validates the script libraries.

Configuration MBeans for WebRTC Session Controller

Table 3–5 lists the configuration MBeans for WebRTC Session Controller.

Table 3–5 WebRTC Session Controller Configuration MBeans

MBean	Description
`ApplicationMBean`	WebRTC Session Controller application configuration MBean. This MBean allows you to do the following for an application profile: add a library, add or clone a package, add package methods; get and set parameters in the application profile, such as the name, description, request URI.
`AscMBean`	Media Engine configuration MBean. This MBean provides access to the address, port, and state of the Media Engine.
`AscMBeans`	Media Engine configuration MBean. This MBean provides access to the media agent notifications, user name, password.
`ClientApplicationsMBean`	Client applications MBean. This MBean is used to create an application, delete an application given its appId, and retrieve an array of applications.
`ClientApplicationMBean`	Client application settings configuration MBean This MBean is used to access to the parameters displayed for a specific application.
`ConfigAdminMBean`	WebRTC Session Controller configuration administration MBean. This MBean is used to access to the general administration configuration parameters.
`NotificationPropertiesMBean`	Notification Service Configuration MBean This MBean is used to get and set the proxy IP address and proxy port of the notification service entry.
`PackageMBean`	WebRTC Session Controller package configuration MBean. This MBean is used to create and remove a script configuration, retrieve package name, and all scripts.
`ProviderCertificatesMBean`	Cloud Messaging provider Certificate Configuration MBean This MBean is used to create a provider certificate, delete a provider certificate given its name, and retrieve an array of provider certificates.
`ProviderCertificateMBean`	Cloud Messaging provider Certificate MBean This MBean is used to set or retrieve the content, name and pass phrase of a provider certificate.
`ResourceLimitsProfileMBean`	WebRTC Session Controller resource limits configuration MBean. This MBean is used to set or retrieve the resource limits for the resource parameters of a resource limit entry.
`ScriptLibraryMBean`	WebRTC Session Controller script library configuration MBean. This MBean is used to set or retrieve the script library macros and retrieve the type of a script library.
`ScriptMBean`	WebRTC Session Controller script configuration MBean. This MBean is used to get the name, content, criteria, type, and method name of a script; and set the script content and criteria.
`StatisticsRuntimeMBean`	WebRTC Session Controller application configuration MBean. This MBean is used to retrieve the runtime session statistics and watermarks.
`SystemConfigurationsMBean`	WebRTC Session Controller application configuration MBean. This MBean is used get and set the system integration parameters and media engine configurations

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

MBean	Description
`TenantProfileMBean`	WebRTC Session Controller application configuration MBean. This MBean is used get and set the parameters of a tenant profile.
`WebSocketMBean`	WebRTC Session Controller WebSocket configuration MBean. This MBean is used get and set the maximum allowed connections and the idle time out (in seconds).
`WscConfigMBean`	WebRTC Session Controller main configuration MBean. This MBean is used create and delete the main configuration settings in WebRTC Session Controller.

See the **oracle.wsc.core.configuration.admin.mbean** package in *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.

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.

Managing Application and Tenant Profiles Using WebLogic Scripting Tool

Manage the tenant profile using WebLogic Scripting Tool:

- **WscConfigMbean**

  Use the following methods:

  – To create a tenant profile, use the **createTenantProfile** method. This method returns an **ObjectName**. A JMX client can access the **TenantProfileMBean** with this name entry.

  - To remove the tenant profile, use the **removeTenantProfile** method

- **ApplicationMbean**:

  Use the following methods:

  - To assign one or more tenant profiles to an application profile, call the **setTenantProfile** method.
  - To get a list of the tenant profiles assigned to the application profile, use the **getTenantProfile** method.

  To assign one or more tenant profiles to an application profile, call the **setTenantProfile** method. The **getTenantProfile** method of the **ApplicationMbean** MBean returns the tenant profile.

- To administer a tenant profile, access the **TenantProfileMBean**. You can get and set its elements such as the name, key, resource limits and the groovy properties associated with the tenant profile.
- To generate the statistics for monitoring at system, application profile, and tenant profile levels, use the methods in the **StatisticsRuntimeMBean**.

For more information on accessing these MBeans, see *WebRTC Session Controller Configuration API Reference.*

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 and EDRs" 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 call-state storage 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 Coherence servers in each partition.

   The engine server cannot function until SIP Coherence servers 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 Coherence server by using either the **startManagedWebLogic.cmd** script installed with the Configuration Wizard, or a custom startup script. The **startManagedWebLogic.cmd** script requires that you specify the name of the server to start up and the URL of the Administration Server for the domain. For example:

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

3. Start engine servers.

   Start Signaling Engine servers and begin processing client requests. Signaling engine 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 Coherence configuration. This sequence also avoids engine error messages that are generated when SIP call-state storage is 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.
WebRTC Session Controller	-DallowSessionTransfer	See "Supporting Session Rehydration for Device Handover Scenarios" for more information.
WebRTC Session Controller	-Doracle.wsc.stats=false	See "Monitoring Statistics and Resource Limits" for more information.
WebRTC Session Controller	-Doracle.wsc.se-me-http=true	See "Disabling the HTTPS Setting in WebLogic Server" 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.

Supporting Session Rehydration for Device Handover Scenarios

Your customer who is logged in to your application on one device (a cellphone) may move to another device (a laptop softphone) to continue the same activity in the same application. The shift of the application session from one device to another occurs due to the customer's actions in the personal, local, or wide area network.

The application Session (along with its subsessions) currently active in your application on one device belonging to your customer becomes active on your application on another device belonging to the same customer. In order to allow the same user or tenant to connect with the WSC server using the same WebSocket session Id, set the startup option **allowSessionTransfer** to **true**:

```
./startWeblogic.sh -DallowSessionTransfer=true
```

By default, this option is set to **false** and prevents the WSC server from rehydrating your application on the target device with the active session transferred from the original device.

For information about how the application session state information is transferred to another device, see "Handling Session Rehydration When the User Moves to Another Device" in *WebRTC Session Controller Application Developer's Guide*.

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 or token-based authentication. A return code of 200 indicates that authentication was successful.

- OAuth 2.0 authentication

  This 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 or phone number, with the consent of that user.

The following sections describe the configuration steps for each of these 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 administration server if it is not already running. See *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://hostname:port/console`

   where *hostname* 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. Select **myrealm** in the Realms table.
5. Select 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. Select the newly created authentication provider in the list of Authentication Providers, and select 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, select **DefaultAuthenticator**.
10. Select the **Common** tab and choose a value for the **Control Flag**.

    For information on **Control Flag** settings, see "Setting the JAAS Control Flag Option" in *Administering Security for Oracle WebLogic Server*.
11. Click **Save**.
12. Log out of the WebLogic administration interface and restart WebRTC Session Controller.

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

Configuring the WebRTC Session Controller Guest Access Application

In WebRTC Session Controller Administration Console, configure and manage the tenant application profile for each tenant through the **Application Profiles** tab.

See "Managing WebRTC Session Controller Application Profiles".

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 endpoints defined to handle authentication and identity assertion requests.

Configuring the WebLogic Server HTTP Authentication Provider

To configure the WebLogic Server HTTP access provider:

Create an entry for the Security provider

1. Start your Signaling Engine servers if they are not already running. See *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://hostname:port/console`

   where *hostname* 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. Select **myrealm** in the Realms table.
5. Select the **Providers** tab and then click **New**.
6. Enter a name in the Name text box, in the Type drop down list, select **WscRestAuthenticator**.
7. Click **OK**.

Choose a value for the Control Flag

1. From the list of Authentication Providers, select the newly created authentication provider.
2. Select the **Common** tab.
3. Choose a value for the **Control Flag**.

   For information on **Control Flag** settings, see "Setting the JAAS Control Flag Option" in *Administering Security for Oracle WebLogic Server*.

Enter the provider-specific configuration values

1. Select the **Provider Specific** tab.
2. Enter a **Group Name** to associate a group with authentication requests rather than individual user names. Make a note of this group name.
3. Enter a **Token Name**. The token name must match the name of the authentication token that is sent in the HTTP request parameter.
4. Enter an **Authentication End Point Url**. A REST endpoint URL that handles authentication.
5. To enable authentication over http, check **Allow Http**.
6. In the **Forward Header Prefix** field, enter the prefix to prepend on header names when forwarded to the REST service.
7. In the **Forward Parameter Names** field, enter the HTTP parameters to extract from HTTP requests and send as query parameters with authenticate and identity assertion REST requests.

   For multitenancy scenarios, enter the tenant key for this tenant. The default configuration entry is **tenant_profile_key**. It is included by default in HTTP requests.
8. In the **Identity Asserter End Point Url** field, enter a REST endpoint URL that handles matching the authentication token to a user.

If a user is found, a JSON string is returned by the REST endpoint with the user's credentials. Otherwise an HTTP 401 Forbidden error is returned.

9. In the **Forward Header Names** field, enter the HTTP headers to extract from HTTP request and send as headers with authenticate and identity assertion REST requests.

   The extracted header name is prefixed with the value from the **Forward Header Prefix**.

Save your configuration and restart WebRTC Session Controller

1. Click **Save**.
2. Log out of the WebLogic administration interface.
3. Restart WebRTC Session Controller.

Note: If authentication is successful, and if the response body returned by the remote HTTP endpoint is valid JSON formatted data, WebRTC Session Controller normalizes the JSON data as a Java Map and embeds this normalized data as credential information in the authenticated subject. That credential information is accessible in the groovy layer, enabling you to use it to build a credential map for the SIP Register request.

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

Configuring the WebRTC Session Controller HTTP Access Application

In WebRTC Session Controller Administration Console, configure and manage the tenant application profile for each tenant through the **Application Profiles** tab.

See "Managing WebRTC Session Controller Application Profiles".

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 URL
- An OAuth client ID supplied to you by the OAuth service provider
- The service provider's OAuth server URL
- Your OAuth client secret, defined when you create your account with your OAuth service provider

Following the general OAuth configuration steps, two specific OAuth configuration examples are provided:

- Example: Configuring Facebook OAuth Authentication
- Example: Configuring Google OAuth Authentication

Configuring the WebLogic Server OAuth Access Provider

To configure the WebLogic Server OAuth access provider:

Create an entry for the OAuth Security provider

1. Start your Signaling Engine servers if they are not already running. See *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://hostname:port/console`

   where *hostname* 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. Select **myrealm** in the **Realms** table.
5. Select 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 requires no further configuration.

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 **OK**.

Enter the Access Token for the Provider

1. From the list of Authentication Providers, select the newly created authentication provider.
2. From the set of **Available** tokens under **Active Types**, select an authentication token to assign as access token to the provider in Active Types and click **Save**.
   a. From the set of **Available** tokens under **Active Types**, select the authentication token.
   b. To move the selected token to the **Chosen** field, click the double arrow pointing to that field.

   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.

3. 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 authentication source from the **Active Types** list.)

Enter the provider-specific configuration values

1. Select the **Provider Specific** tab.
2. Enter the information specific to this provider, 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.
Fields As User Name	Required. Determines which OAuth provider resources are used as principal names. Multiple entries are separated by commas. If the first entry returns nothing, then the second entry is used, continuing down the list.
OAuth Redirect Url	Optional. The URI to which the browser is re-directed after successful authentication by the OAuth provider.
Proxy Server	Optional. The proxy URI used to connect to the OAuth server.
OAuth Client Secret	Required. The OAuth client secret provided to you by your OAuth provider.
OAuth Client ID	Required. The OAuth client ID provided to you by your OAuth service provider.
Proxy Port	Optional. The proxy port used to connect to the OAuth server.
OAuth Server Url	Required. The URI of your OAuth service provider's OAuth server which issues access tokens.
OAuth User Info Url	Required. The OAuth providers URI that provides user information.

Save your configuration and restart WebRTC Session Controller

1. Click **Save**.
2. Log out of the WebLogic administration interface.
3. Restart WebRTC Session Controller.

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

Configuring the WebRTC Session Controller OAuth Access Application

In WebRTC Session Controller Administration Console, configure and manage the tenant application profile for each tenant through the **Application Profiles** tab.

See "Managing WebRTC Session Controller Application Profiles".

How Authentication Schemes Work in Multitenancy Scenarios

Before you proceed, please review the description about multitenancy in WebRTC Session Controller in the section, "About Multitenancy".

In general, your SaaS applications authenticate the users of a tenant according to their proprietary mechanisms. After a user is successfully authenticated, the SaaS application generates a token for the authenticated user. The user presents the token (using the client SDKs) to access WebRTC Session Controller resources. To validate the token, WebRTC Session Controller accesses a REST service.

WebRTC Session Controller does the following:

- Basic Authentication HTTP request

  When a Basic Authentication HTTP request arrives, WebRTC Session Controller extracts the user name, password, any optional query parameters and headers enabled for forwarding in **Forward Header Names** and **Forward Parameter Names** fields. (The tenant key is the value entered in the **Forward Parameter Names** field).

  WscRestAuthenticator makes a REST request to the URL specified in the **Authentication End Point Url** field by using:

  – The enabled forward parameters as query parameters

  – The enabled forward headers as HTTP headers with the header name prefixed with the value from forward header prefix field

  – The user name and password as basic authentication credentials

- HTTP request with a query parameter

  When an HTTP request with a query parameter arrives with the name specified in the **Token Name** field, this parameter is inserted by a servlet filter in the provider as HTTP header under the name **RestAccessAuthToken** to have identity assertion invoked.

- When an HTTP request arrives with an HTTP header named **RestAccessAuthToken**, WebRTC Session Controller extracts the user name, password, any optional query parameters and headers enabled for forwarding in **Forward Header Names** and **Forward Parameter Names** fields. (The tenant key is the value entered in the **Forward Parameter Names** field).

  WscRestAuthenticatorl makes a REST request to the URL specified in the **Identity Asserter End Point Url** field by using:

  – The enabled forward headers as HTTP headers with the header name prefixed with the value from the **Forward Header Prefix** field.

  – The enabled forward parameters as query parameters together with a "token" parameter with the value from the **RestAccessAuthToken** header

- When either authentication request or identity assertion REST request completes successfully, principals with the value from the **Group Name** field are added to the subject together with a public credential map object containing the credentials returned by the REST server as a JSON formatted response. For authentication requests, the user name is also added as a subject principal.

About the Default REST Request Format

Example 4–1 shows a Basic authenticated HTTP request for user "wsc1" and with and without tenant profile key parameter included:

Example 4–1 Basic Authenticated HTTP Request

For a user "wsc1" and no tenant profile key parameter included:

```
GET /authenticate/basic HTTP/1.1\r\n
Content-Type: application/json\r\n
Accept: application/json\r\n
Authorization: Basic wsc1:d2VsY29tZTE=\r\n
Host: localhost:8190\r\n
...
```

For a user "wsc1token" with "tenant1" as the tenant profile key parameter:

```
GET /authenticate/basic?tenant_profile_key=tenant1 HTTP/1.1\r\n
Content-Type: application/json\r\n
Accept: application/json\r\n
Authorization: Basic wsc1:d2VsY29tZTE=\r\n
Host: localhost:8190\r\n
...
```

Example 4–2 shows an Identity asserted HTTP request for user "wsc1" and no tenant profile key parameter included:

Example 4–2 Identity Asserted HTTP Request

For user "wsc1" and no tenant profile key parameter included:

```
GET /authenticate/basic?token=wsc1token HTTP/1.1\r\n
Content-Type: application/json\r\n
Accept: application/json\r\n
Host: localhost:8190\r\n
...
```

For user "wsc1token" with "tenant1" as the tenant profile key parameter:

```
GET /authenticate/token?token=wsc1token&tenant_profile_key=tenant1 HTTP/1.1\r\n
Content-Type: application/json\r\n
Accept: application/json\r\n
Host: localhost:8190\r\n
```

Working with Custom and WebLogic LDAP Security Providers

If you employ a custom security provider, ensure that you provide the tenant key and the necessary logic to process the requests. You can also make use of **ServletAuthenticationFilter** to perform pre- and post-processing for authentication functions, including identity assertion.

For more information on **ServletAuthenticationFilter**, see the description about "Servlet Authentication Filters" in *Fusion Middleware Developing Security Providers for Oracle WebLogic Server*.

If you employ the WebLogic LDAP security provider, ensure that your implementation isolates the stores for each tenant. To do so, use the following workarounds:

- Configure multiple authentication providers. Additionally configure the control flag as **sufficient** in the WebLogic Admin Server.
- Create a separate domain for each tenant.

Example: Configuring Facebook OAuth Authentication

This example outlines the steps to follow to configure OAuth authentication using Facebook as an OAuth authentication provider.

> **Note:** You must have a Facebook and be registered a Facebook or application developer before you can configure OAuth authentication as described in this example.

Configure a Facebook Authentication App

To configure a Facebook authentication app:

1. Login to http://developers.facebook.com.
2. Click the **Apps** menu and then click **Add a New App**.
3. Choose a platform.
4. Enter a name for your app.
5. Click **Create New Facebook App ID**.
6. Choose a category from the Category drop down list and click **Create App ID**.
7. Click **Skip Quick Start**.
8. Click the Settings in the left panel and copy the **App ID** and the **App Secret** to a scratch file for future reference.

   > **Note:** You must authenticate your Facebook account to display the App Secret.

9. Add a Contact Email and click **Save Changes**.
10. Click **Add Platform** then click **Website**, enter the URL and port for your site, and click **Save Changes**.

    > **Note:** Facebook does not accept IP addresses. You must use a domain name.

11. Click **Status & Review** in the left panel, and in the Status tab set the switch adjacent the label *Do you want to make this app and all its live features available to the general public?* to **YES**. Click the **Confirm** button when prompted.

Configure the Facebook WebRTC Session Controller OAuth Authentication Provider

To configure a Facebook WebRTC Session Controller OAuth authentication provider:

1. Start your Signaling Engine servers if they are not already running. See *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://hostname:port/console`

where *hostname* 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. Select **myrealm** in the **Realms** table.
5. Select 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 requires no further configuration.

7. Click **New**.
8. Enter a name for the provider in the Name text box, in the Type drop down list, select **WscOAuthIdentityAsserter**, and click **OK**.
9. Select 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**.

> **WARNING: The user interface will let you select multiple OAuth tokens for a single provider. Only select a single token for the Facebook OAuth provider. Note that each OAuth provider you provision must have a separate and distinct OAuth token.**

11. Select the Provider Specific tab and enter the following information as described in Table 4–2.

Table 4–2 OAuth Provider Specific Attributes

Attribute Name	Attribute Description
Group Name	This should be set to the value you entered in step 8.
OAuth User Info Url	Leave this set to the default, https://graph.facebook.com/me?.
Proxy Port	Set the Proxy Port to 80.
OAuth Client ID	Enter the App ID from your Facebook App configuration.
OAuth Server Url	Leave this set to the default, https://graph.facebook.com/oauth/access_token.
OAuth Redirect Url	Set this to the redirect URL you provided when creating your Facebook App.
Fields As User Name	Leave this set to the default.

Table 4–2 (Cont.) OAuth Provider Specific Attributes

Attribute Name	Attribute Description
OAuth Client Secret	Enter the App Secret from your Facebook App configuration.
Proxy Server	Enter the URI of your proxy server.

12. Click **Save**.
13. Log out of the WebLogic administration interface and restart WebRTC Session Controller.

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

Example: Configuring Google OAuth Authentication

This example outlines the steps to follow to configure OAuth authentication using Google as an OAuth authentication provider.

> **Note:** You must have a Google Gmail account and be registered as a Google application developer before you can configure OAuth authentication as described in this example.

Configure a Google Authentication Project

To configure a Google authentication project:

1. Login to `https://console.developers.google.com/project`.
2. Click the **Create Project**.
3. Enter a Project Name and a Project ID, and click **Create**.
4. Click **APIs & auth** in the left panel and then click **Credentials**.
5. In the right panel, click **Create new Client ID** and then click **Configure consent screen**.
6. Choose an email address and enter a Product Name. Fill in any other information you require and click **Save**.
7. In the Create Client ID dialog, choose **Web application** for the Application Type.
8. In Authorized JavaScript Origins, enter the URI and port for your WebRTC Session Controller application, and in Authorized Redirect URI enter the URI to use for authentication redirects.

> **Note:** Google does not accept IP addresses. You must use a domain name.

9. Click **Create Client ID**.
10. Once the Client ID is created, copy the **Client ID** and the **Client Secret** to a scratch file for future reference

Configure the Google WebRTC Session Controller OAuth Authentication Provider

To configure a Google WebRTC Session Controller OAuth authentication provider:

1. Start your Signaling Engine servers if they are not already running. See *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://hostname:port/console`

   where *hostname* 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. Select **myrealm** in the **Realms** table.
5. Select 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 requires no further configuration.

7. Click **New**.
8. Enter a name for the provider in the Name text box, in the Type drop down list, select **WscOAuthIdentityAsserter**, and click **OK**.
9. Select 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**.

    WARNING: The user interface will let you select multiple OAuth tokens for a single provider. Only select a single token for the Facebook OAuth provider. Note that each OAuth provider you provision must have a separate and distinct OAuth token.

11. Select the Provider Specific tab and enter the following information as described in Table 4–3.

Table 4–3 OAuth Provider Specific Attributes

Attribute Name	Attribute Description
Group Name	This should be set to the value you entered in step 8.
OAuth User Info Url	Set this to https://www.googleapis.com/oauth2/v1/userinfo.
Proxy Port	Set this to 80.
OAuth Client ID	Enter the Client ID from your Google project configuration.

Table 4–3 (Cont.) OAuth Provider Specific Attributes

Attribute Name	Attribute Description
OAuth Server Url	Set this to https://accounts.google.com/o/oauth2/token.
OAuth Redirect Url	Set this to the redirect URL you provided when creating your Google project.
Fields As User Name	Leave this set to the default.
OAuth Client Secret	Enter the Client Secret from your Google project configuration.
Proxy Server	Enter the URI of your proxy server.

12. Click **Save**.
13. Log out of the WebLogic administration interface and restart WebRTC Session Controller.

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

About Post-Authentication Redirection

In certain cases, you may want to implement a two stage authentication workflow for your WebRTC Session Controller application. In a two stage authentication workflow, once a user has been authenticated by a standard authentication method (HTTP, OAuth or WebLogic in the case of WebRTC Session Controller), an additional separate authentication method is invoked. That separate authentication method usually takes the form of a one-time password which is delivered to the user either by email or short message (SMS). Once the one-time password is dispatched to the user, the user is redirected to a separate authentication web page where the one-time password is validated. After validation, the second stage authentication is usually skipped on subsequent logins.

> **Note:** WebRTC Session Controller does not provide facilities for one-time password generation and authentication. Such a system must be implemented by you according to your requirements.

In order to support two stage authentication, WebRTC Session Controller provides a Groovy script library function, **validateAuthenticatedUser**.

About the validateAuthenticatedUser Function

The **validateAuthenticatedUser** function lets you evaluate a user's HTTP request details such as request parameters and cookies, and provide redirection to a web page if required based upon those details.

See "Editing validateAuthenticatedUser" for details on accessing and updating the validateAuthenticatedUser function.

> **Note:** While the **validateAuthenticatedUser** function is defined in the WebRTC Session Controller Script Library it must be implemented as per your system requirements. The default function logic is only for purposes of illustration.

Syntax

The **validateAuthenticatedUser** function has the following syntax:

```
void validateAuthenticatedUser(final HttpFilterContext httpFilterContext)
```

The **HttpFilterContext** class includes methods that return: parts of the request URL, the client's IP address, the authenticated subject, a Map of HTTP request parameters, and a Map of request cookies. It also has methods that redirect the client to a specified URL, log out of the current session, and so on. For a complete description of the **HttpFilterContext** class, see the *Oracle Communications WebRTC Session Controller Configuration API Reference*.

Example

Example 4–3 illustrates a simple **validateAuthenticatedUser** implementation as well as the use of some **HttpFilterContext** methods.

Example 4–3 validateAuthenticatedUser

```
void validateAuthenticatedUser(final HttpFilterContext httpFilterContext) {

  def loginCookie = httpFilterContext.cookies.WSC_LOGIN_COOKIE
  def tempCookieValue = "temp_session_cookie";
  if (tempCookieValue != loginCookie) {
    httpFilterContext.redirect('/test/sample');
    httpFilterContext.logOut('/test/newpage');
    httpFilterContext.addCookie(loginCookie, tempCookieValue);
  }
```

The function executes in the following manner:

1. The login cookie is retrieved from the `httpFilterContext.cookies.WSC_LOGIN_COOKIE` and stored in `loginCookie`.
2. An additional cookie is defined for the current session.
3. The cookie created for the current session is compared to the login cookie.
4. If the cookies match, no redirection occurs.
5. If the cookies do not match, the function sets the redirect and logout URLs and copies the session's cookie value to the login cookie.
6. Upon the user's second access attempt, the session cookie and the login cookie will match, and no redirection will occur.

That example uses custom cookies to track if this is the first login for a specific user. The cookie is reset when the browser restarts, which means that the user's next login will trigger a new redirect. To prevent continual redirects, you will need to flag a user as successfully authenticated in a persistent manner. For example, you can base the decision to redirect on an additional metadata comparison against the AUTHENTICATED_SUBJECT.

Note: Such support requires additional custom integration with the security provider.

Editing validateAuthenticatedUser

To edit the **validAuthenticatedUser** function:

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

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

   where *hostname* 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 **Script Library** tab.
3. Click **Edit**.
4. Edit the function **validateAuthenticatedUser** as required for your needs.
5. Click **Validate Library** to make sure you have not introduced any errors.
6. Click **Save** to save your changes to the Script Library.

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 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 server, you must create a Diameter 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 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 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 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 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 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 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 *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 *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 Container Properties

This chapter describes how to configure SIP container features in the engine 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** sub tab 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 servers.

Adding Servers to the WebRTC Session Controller Cluster

WebRTC Session Controller instances configured as replicated domains include the default **BEA_ENGINE_TIER_CLUST** cluster for the signaling engine servers. 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 servers add new call state data to the SIP call-state store, they maintain data structures to track the SIP protocol timers and application timers associated with each call. Engine servers periodically poll the SIP Coherence call-state store to determine which timers have expired, given the current time. By default, multiple engine server poll to the call-state store are staggered to avoid contention on the timer tables. Engine servers then process all expired timers using threads allocated in the **wlss.timer** work manager.

Configuring Timer Affinity (Optional)

With the default timer processing mechanism, a given engine 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, each engine server periodically polls the SIP call-state store for processed timers. When polling the SIP call-state store, 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 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 call-state store.

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 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 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 among engine servers. For example, one engine server might process all expired timers, whereas other engine 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 server with a system clock 250 milliseconds faster than other servers will process more expired timers than other engine servers, will cause retransmits to begin in half the allotted time, and may force messages to time out prematurely.

7

Using the Lightweight Proxy Registrar

This chapter describes the Oracle Communications WebRTC Session Controller Lightweight Proxy Registrar and how to configure it.

You need to perform the tasks in this chapter only if you intend to use the Lightweight Proxy Registrar. If you will not use the Lightweight Proxy Registrar, you can skip this chapter.

About the Lightweight Proxy Registrar

The Lightweight Proxy Registrar introduces a layer between the WebRTC Session Controller Signaling Engine and the Proxy Registrar. The Lightweight Proxy Registrar reduces resources consumed in the Proxy Registrar, which reduces overall cost.

WebRTC-based clients come and go as people open and close their browsers. Each WebSocket connection to WebRTC Session Controller triggers a SIP registration. A single user often has multiple devices, which equates to multiple endpoints and requires more register and unregister requests. These factors make the number of registrations at any time difficult to predict, making it harder to plan needed resources. Any cost associated with registration might not be well known and could be problematic.

The Lightweight Proxy Register addresses these problems in one of two ways:

- Multiplexing registration requests from many WebRTC endpoints into a single SIP registration per user
- Managing all registrations, leaving no registrations for the Proxy Registrar. In this case, an external system must route inbound calls to WebRTC Session Controller, for example, by using static routes based on the domain.

Customers who do not want or need a SIP or IMS integration do not need to use them This case is suitable for an enterprise that only wants to connect WebRTC endpoints.

The Lightweight Proxy Registrar forks inbound and outbound calls, or SIP INVITE messages, to multiple connections.

About SIP Registration Modes

WebRTC Session Controller operates in three modes:

- **Normal**: The Lightweight Proxy Registrar is not part of the call flow, which means that every WebRTC endpoint will trigger a unique SIP registration towards the proxy registrar. This is the default behavior.
- **Single**: The WebRTC Session Controller Signaling Engine sends REGISTER requests to the Lightweight Proxy Registrar, which tracks and forwards only the

first registration per user to the Proxy Registrar. Likewise, the Signaling Engine sends a de-registration message to the Proxy Registrar when the last registration for a user is removed.

- **Static**: The same as Single mode except that no registrations are sent to the Proxy Registrar. You can use this mode when IP Multimedia Subsystem (IMS) integration is not required, although you can use this mode with an IMS integration. Two possible cases for using this mode are:
  - You do not want or need SIP or IMS integration, such that only WebRTC endpoints can communicate.
  - You want to completely off load all registrations, but still be able to route calls to and from WSC using, for example, static configured routes.

About Proxy Forking Modes

For requests that originate from WebRTC endpoints, you can configure the WebRTC Session Controller to operate in one of the following modes:

- **Always**: WebRTC Session Controller forwards outbound requests to a remote proxy and does not use the Lightweight Proxy Registrar
- **Conditional**: if one or more WebRTC endpoints exist in the registrar repository, the Lightweight Proxy forks the request locally. If no endpoint exists, the request is routed to a remote proxy.
- **Never**: The Lightweight Proxy handles all requests internally and WebRTC Session Controller never routes requests to a remote proxy. If no endpoints exist in the registrar repository, the Lightweight Proxy Registrar responds with the error: 404 Not Found.

About Lightweight Proxy Registrar Components

The Lightweight Proxy Registrar consists of the following components:

- Lightweight Registrar

  The Lightweight Registrar maintains the list of active bindings in the Location Service, propagates requests from WebRTC endpoints, propagates REGISTER requests or responses to and from the external registrar, and determines the registration mode based on the registration mode configuration.

- Lightweight Proxy

  The Lightweight Proxy processes both inbound and outbound call setup attempts by forwarding and, optionally, forking SIP INVITE requests.

- Location Service

  The Location Service maintains information about the location of the called party.

- Custom Application Router

  The Custom Application Router is called by the container to select which SIP servlet application will service each initial request.

About the Lightweight Registrar

The Lightweight Registrar is triggered by SIP REGISTER requests and responses. It takes the following actions:

- Maintains the list of active bindings per Addresses-of-Record in the Location Service, based on the Contact headers and expiration intervals in the requests or responses from the external registrar.
- Only propagates requests received from WebRTC endpoints based on the registration modes. It does not generate any re-register requests or maintain any timers.
- Propagates REGISTER requests or responses, without modifying them, to or from the external registrar. Transparently forwards any authentication or authorization headers between the external Proxy Registrar and the WebRTC Session Controller Signaling Engine. The last REGISTER request in Single mode, however, is changed by the Lightweight Proxy Registrar to include exactly the same Contact header(s) as the first REGISTER request for the sake of removing the bindings correctly in the external Proxy Registrar.
- Determines the registration mode based on the registration mode configuration.

About the Lightweight Proxy

The Lightweight Proxy is triggered by both inbound and outbound SIP INVITE requests.The Lightweight Proxy sets up both inbound and outbound calls by forwarding and, optionally, forking the SIP INVITE requests.

The Lightweight Proxy takes the following actions:

- Forks INVITE requests to multiple WebRTC endpoints based on the bindings information in the Location Service. The called party address-of-record for the subscriber lookup is based on the To header of the INVITE request.
- Forks the call in parallel if it finds one or more bindings. Otherwise, it forks the call to the outbound Proxy or responds with an error, depending on the forking mode. If no WebRTC endpoint for the called party corresponds to the address-of-record, the Lightweight Proxy responds with the error: 404 Not Found.
- Determines the forking mode based on the Forking Mode Configuration. In Always mode, the Custom Application Router filters out requests.

About the Location Service

The Lightweight Proxy uses the Location Service to obtain information about the possible location of the called party. The Lightweight Registrar maintains the information as a result of processing the REGISTER message.

The Location Service contains a lookup table of the bindings between the address-of-record keys and registration records. Each record represents one or more contact header addresses, which have been received from prior registration requests from WebRTC endpoints. Data is kept in memory, replicated across the WebRTC Session Controller servers, but is not written to disk. This means that a full cluster restart clears all data because a cluster restart means all WebRTC sessions are gone.

Table 7–1 represents an example of a lookup table that contains three records.

Table 7–1 Example Lookup Table

Address-of-Record	Registration Record
alice@example.com	<instance-id1, reg-id1> = <alice-contact1, expires=t1> <instance-id2, reg-id2> = <alice-contact2, expires=t2>

Table 7–1 (Cont.) Example Lookup Table

Address-of-Record	Registration Record
bob@example.com	<instance-id3, reg-id3> = <alice-contact3, expires=t3> <instance-id4, reg-id4> = <alice-contact4, expires=t4> <instance-id5, reg-id5> = <alice-contact5, expires=t5>
alice@otherdomain.com	<instance-id1, reg-id1> = <alice-contact6, expires=t6>

Handling Multitenancy

In a service as a software (SaaS) environment, a single user identity may be associated with multiple tenant profiles and each of them may reach different SIP proxy registrars.

To handle this scenario, the Signaling Engine inserts a **tenantToken** to the SipURI of the contact header in requests sent to the Lightweight Proxy Registrar. In turn, the Lightweight Proxy Registrar employs this token in its decision -making process and routes the request to the corresponding proxies.

A record in the lookup table uses three entries:

Table 7–2 Example Lookup Table for Multitenancy

TenantToken	Address-of-Record	Address-of-Record
tenant1	alice@example.com	<instance-id1, reg-id1> = <alice-contact1, expires=t1> <instance-id2, reg-id2> = <alice-contact2, expires=t2>
tenant2	alice@example.com	<instance-id1, reg-id1> = <alice-contact6, expires=t6>

About the Custom Application Router

The custom SIP application router is used to route traffic between various WebRTC Session Controller components (SIP servlets). The container calls the application router to select the servlet application that will service each initial request.

About Multiple Identity Support

The Lightweight Proxy Registrar supports the association of multiple subscriber identities with a registered address-of-record. For example, subscriber Maria could have identities of maria-home, maria-work, and maria-mobile. If she connects as both maria-home and maria-mobile, each with different SIP contact addresses, the Lightweight Proxy Registrar stores these relationships and associates them with subscriber Maria. If an inbound call generates an INVITE message for maria-home, the Lightweight Proxy Registrar retrieves all of Maria's registered identities and forks an INVITE to maria-mobile as well.

Configuring the Lightweight Proxy Registrar

To configure the Lightweight Proxy Registrar, you set the SIP registration mode and the proxy forking mode through existing Groovy scripts. Beyond that, the Lightweight Proxy Registrar acts appropriately based on the SIP messages it receives.

Configuring Registration Mode

The default registration mode is Normal, which indicates the Lightweight Registrar is not used and all registration requests are sent to an external registrar. You set the

registration mode only if you want to change the default mode or change the registration mode you previously set.

To configure the registration mode:

1. Log in to the WebRTC Session Controller console using your user name and password.
2. Select the **Packages** tab.
3. Under **Package Name**, select **register**.
4. Select the row with these values:
   - **Direction**: FROM_APP
   - **Verb**: connect
   - **Type**: request
5. Click the **Edit** button.
6. In the Groovy Script section, locate the line that begins with the following text:

   ```
   sipReq.requestURI = context.sipFactory.createSipAddress(Constants.PROXY_SIP_
   URI).URI
   ```

7. Modify the value of **sipReq.requestURI** as follows, based on the mode that you want to use:

   Static mode:

   Replace the Request-URI with a URI corresponding to one of the local SIP listening interfaces. For example:

   ```
   sipReq.requestURI =
   context.sipFactory.createSipAddress("sip:127.0.0.1:5060").URI
   ```

   Single mode:

   Specify the outbound proxy in the Request-URI and push an additional Route header by adding the **def route** entry to specify one of the local SIP listening interfaces. This header indicates that the Lightweight Proxy Registrar should be visited prior to the outbound proxy, depending on the forwarding decision by the Lightweight Proxy Registrar:

   ```
   sipReq.requestURI = context.sipFactory.createSipAddress(Constant.PROXY_SIP_
   URI).URI
   def route = context.sipFactory.createSipAddress("sip:127.0.0.1:5060;lr")
   sipReq.pushRoute(route)
   ```

   Normal mode:

   No changes are required for Normal mode.

Configuring Forking Mode

The default forking mode is Always, which indicates the Lightweight Proxy is not used and all requests are sent to an external proxy. Set the forking mode only if you want to change the default mode or change the forking mode you previously set.

To configure the forking mode:

1. Log in to the WebRTC Session Controller console using your user name and password.

2. Select the **Packages** tab.
3. Under **Package Name**, select **call**.
4. Select the row with these values:
   - **Direction**: FROM_APP
   - **Verb**: start
   - **Type**: request
5. In the Groovy Script section, locate the following lines:

```
def route = context.sipFactory.createSipAddress(Constants.PROXY_SIP_URI +
";lr")
sipRequest.pushRoute(route)
```

6. Modify the lines as follows, based on the mode that you want to use:

   Conditional mode:

   Specify the outbound proxy route and push an additional Route header that contains one of the local SIP listening interfaces. The header indicates that the Lightweight Proxy Registrar should be visited prior to the outbound Proxy, based on the forwarding decision made by the Lightweight Proxy Registrar.

```
def route = context.sipFactory.createSipAddress(Constants.PROXY_SIP_URI +
";lr")
sipRequest.pushRoute(route)
def localroute = context.sipFactory.createSipAddress("sip:127.0.0.1:5060;lr")
sipRequest.pushRoute(localRoute)
```

   Never mode:

   Replace the route header contents with an address corresponding to one of the local SIP listening interfaces.

```
def localroute = context.sipFactory.createSipAddress("sip:127.0.0.1:5060;lr")
sipRequest.pushRoute(localRoute)
```

   Always mode:

   No changes are required for Always mode.

8

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 server in your system. Engine servers access the SIP call-state store using the Coherence cluster configured in the domain.

Note: If you configure the Coherence cluster to use Unicast addressing, you must configure the engines to use either explicit listen addresses or explicit well-known addresses to allow all cluster domain servers to locate each other.

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 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 server for each one. When a particular network interface on the engine 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 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 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 nodes to communicate within the cluster on IPv4 (or IPv6) while supporting the other protocol version for external SIP traffic. To configure engine nodes on an IPv6 network, simply specify IPv6 listen addresses for each server instance and, if desired, for the Coherence cluster communication.

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 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.

  Oracle recommends the TLS V.1.0 protocol for the best security. TLS1 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.

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 8–1.

Example 8–1 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 8–2 for an example configuration.

Example 8–2 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 Listen Addresses for Servers

Each server in the domain is a member in the Coherence cluster, and the default Coherence configuration uses a generated well-known address list based on server listen addresses. You must use explicit listen addresses with the domain servers for Coherence to correctly form a cluster.

You can set up explicit listen addresses using the domain creation wizard or, after creating a domain, by using the Administration console and following these instructions:

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.

Configuring Coherence Cluster Addressing

If you do not want to use explicit listen addresses with domain servers or want to isolate Coherence cluster communication to its own network, you can configure Coherence cluster addressing to use it's own addressing scheme, using one of the following cluster modes.

- Multicast with multicast address, port and time to live. Multicast communication can make more efficient use of the network in some circumstances, but also might not work in all environments.
- Unicast addressing, specifying explicit well-known addresses (WKAs) and explicit Unicast listen ports for servers.

The default setting is Unicast addressing together with a well-known address list generated from the domain server listen addresses

For more details, see "Configuring and Managing Coherence Clusters" in *Administering Clusters for Oracle WebLogic Server*.

9

Configuring Server Failure Detection

This chapter describes how to configure Oracle Communications WebRTC Session Controller to improve failover performance when a server becomes physically disconnected from the network.

Overview of Failover Detection

To achieve a highly-available production system, the WebRTC Session Controller uses the Oracle Coherence distributed cache service to retrieve and write call-state data. The cache service consists of a number of partitions that are spread across the servers that are running in the cluster. Each partition has a primary copy of call-state storage assigned to one server in the cluster, and a backup copy assigned to another server in the cluster. This means that a call state that is required to process a request may reside on a remote server and possibly even a remote machine.

The WebRTC Session Controller architecture depends on the Coherence cache service to detect when a server has failed or becomes disconnected. When an engine cannot access or write call-state data because a server is unavailable, the Coherence cache service detects this and reassigns the lost server's partitions to another server in the cluster and ensures a new backup copy is made available on a different server, if one is running.

Coherence Cluster Overview

The Coherence cache service uses its own cluster communication protocol, known as Tangosol Cluster Management Protocol (TCMP), to invoke remote servers, detect server failure and achieve high availability. This protocol uses an optimized algorithm to quickly detect that a server has become physically disconnected from the network. This algorithm, and the configuration options that are available to modify its behavior, are described in detail in the Oracle Coherence documentation. See the following documentation for more information on Coherence and its distributed cache service.

- "Introduction to Coherence Clusters" in *Developing Applications with Oracle Coherence*
- "Understanding Distributed Caches" in *Developing Applications with Oracle Coherence*

See "Configuring Coherence" and "SIP Coherence Configuration Reference (coherence.xml)" for additional information on configuring Coherence for the WebRTC Session Controller.

Split-Brain Handling

The WebRTC Session Controller relies to a large extent on Oracle Coherence to detect and handle a split-brain condition. A split-brain condition can occur, for example, when connectivity is restored between two or more parts of a cluster that had been isolated from each other. When the WebRTC Session Controller detects such a condition, it attempts to recover by shutting down part of the cluster and expecting the affected servers to restart and join the surviving cluster as new members.

When Coherence detects a split-brain condition, its behavior is controlled primarily through the options related to death detection in the cluster-related configuration.

Coherence Configuration

You can use the following three mechanisms to modify Coherence configuration options:

- The default Coherence cluster configuration file
- The system properties
- The **tangosol-coherence-override.xml** file

> **WARNING: No servers in the domain can be running when you make changes to the Coherence configuration. Also, the configuration must be the same for all servers in the domain or unexpected behavior can result.**

Cluster Configuration File

The default Coherence cluster configuration file, **Custom-Default.xml**, resides in the following location:

`$DOMAIN_HOME/config/coherence/Coherence-Default/`

where *$DOMAIN_HOME* is the root directory for the domain.

Table 9–1 describes the default configuration options that you can specify.

Table 9–1 Coherence Cluster Configuration File Options

Option	Element Name	System Property Name	Default Value
TCP-ring IP-timeout	<tcp-ring-listener><pingtimeout>	tangosol.coherence.ipmonitor.pingtimeout	5
TCP-ring IP-attempts	<tcp-ring-listener><pingattempts	tangosol.coherence.ipmonitor.pingtattempts	2
Service Guardian Timeout	<service-guardian><timeout-milliseconds>	tangosol.coherence.guard.timeout	305000
Packet Delivery Timeout	<packet-delivery><timeout-milliseconds>	tangosol.coherence.packet.timeout	300000

You can override these default configuration options either by modifying the corresponding system properties or creating an override configuration file, called **tangosol-coherence-override.xml**, which you add to the system CLASSPATH variable on all servers.

See the following Coherence documentation for information on which configuration options you can override and for information on how to use the override configuration option:

- "Configuring a Coherence Cluster" in *Administering Clusters for Oracle WebLogic Server*
- "Death Detection Recommendations" in *Administering Oracle Coherence*
- "Configuring Death Detection" in *Developing Applications with Oracle Coherence*
- "Understanding the XML Overrride Feature" in *Developing Applications with Oracle Coherence*
- "Coherence Operational Configuration Reference" in *Developing Applications with Oracle Coherence*

10

Using the Engine Cache

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

Overview of Engine Caching

A WebRTC Session Controller Signaling Engine cluster manages call-state data in several partitions in the memory of each engine server. Each call-state entry resides in one such partition on a specific engine server in the cluster. In many cases the engine server requesting the call-state entry is not the same engine server where it is stored. Engine servers fetch and write data in the SIP call-state store as necessary. Each call state data partition can have one or more backup copies in another server to provide automatic failover in the event that a SIP call-state store server fails or shuts down for some reason.

WebRTC Session Controller also provides the option for engine servers to cache a portion of the call-state data locally. When a local cache is used, an engine server first checks its local cache. If the cache contains the required data, and the local copy of the data is up-to-date (compared to the SIP call-state store copy), the engine locks the call state in the SIP call-state store 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 call-state store.

The engine cache stores only the call state data that has been most recently used by engine 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 cache is not configurable.

Using a local cache is most beneficial when a SIP-aware load balancer manages requests to the engine cluster. With a SIP-aware load balancer, all of the requests for an established call are directed to the same engine 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 severs (having different cache contents).

Configuring Engine Caching

By default, engine caching is enabled. To disable partial caching of call state data in the engine, 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 cache is not configurable.

Monitoring and Tuning Cache Performance

The **SipPerformanceRuntime** MBean monitors the behavior of the engine cache. Table 10–1 describes the MBean attributes.

Table 10–1 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 server's local cache. It increments this attribute even if the cached data is out-of-date and requires updating with data from the SIP call-state store.
`cacheValidHits`	The server increments this attribute 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 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.

11

Configuring Coherence

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

WebRTC Session Controller uses Coherence for the following purposes:

- Cluster-wide engine communication and state management
- Application call-state storage and management for concurrent SIP calls

About Coherence Engine Communication and State Management

The Domain Creation Wizard automatically creates a default Coherence cluster for managing WebRTC Session Controller information when it sets up new domains. The default cluster includes the engine servers and the administrative server in your environment.

Configuring Coherence for Engine Communication and State Management

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 **Coherence-Default**.
5. Configure the parameters for the Coherence cluster as needed.
6. Click **Save**.

Each engine 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 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 the engine server or the Administration Server for which you want to configure Coherence settings.
5. In the **Configuration** tab, select **Coherence**.
6. Configure the Coherence parameters for the server.
7. Click **Save**.

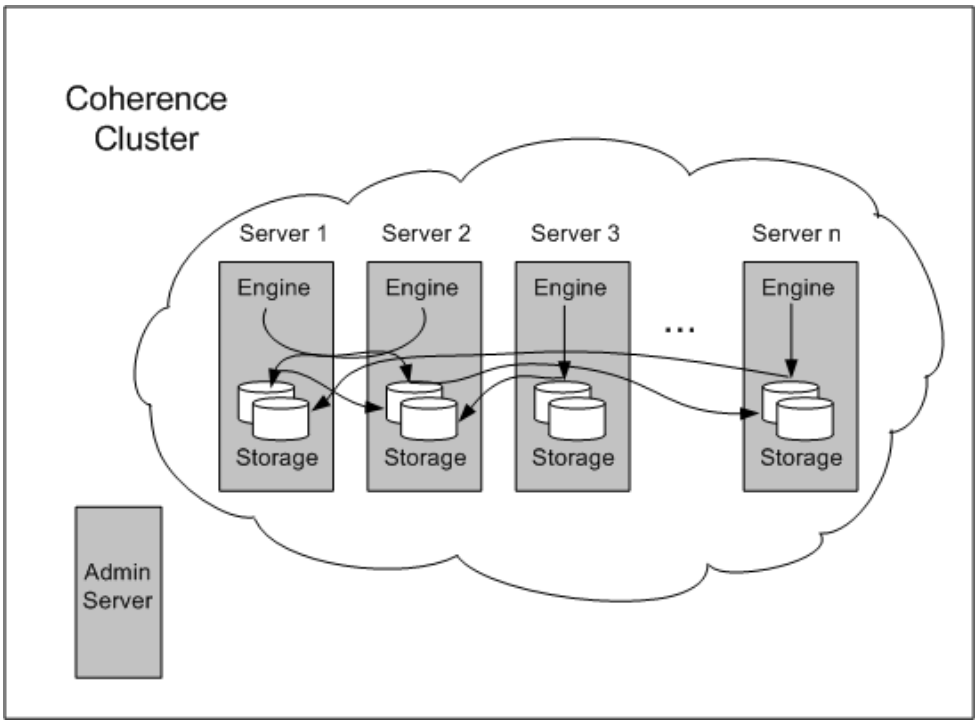
About Call-State Storage and Management for SIP Calls

The Coherence call-state storage facility for WebRTC Session Controller is built on the distributed cache service of WebLogic Server 12.1.3. In each managed server in the domain cluster, Coherence combines logic and processing with state-storage data. Coherence writes data to the primary partition cache-storage server and it, in turn, writes a backup copy to the configured number of backup copies.

See "Understanding Distributed Caches" in *Developing Applications with Oracle Coherence* for an explanation of Coherence distributed caches.

Figure 11–1 illustrates an administration server with a Coherence cluster for call-state storage:

Figure 11–1 Coherence Cluster for Call-State Storage



The Coherence call-state storage facility includes the following features:

- Built-in support for dynamically adding or removing nodes

- Partitions that migrate dynamically, eliminating the need to configure replica servers and their partitions
- Enhanced data serialization with Portable Object Format (PoF)
- Proven node death detection for fail-over and split brain handling
- Flexible configuration
- Advanced network protocol that leverages UDP and supports multi-cast to optimize network usage
- Graceful migration of partitions from one node to another during startup and shutdown, limiting the impact on ongoing traffic and reducing the risk of overload

Configuring Coherence Call-State Storage

The **coherence.xml** custom resource file specifies a subset of the configuration options that control call-state storage. The **config.xml** file specifies the custom resource file as *$domain_home*/**config/custom/coherence.xml**. The entry in the **config.xml** file looks like this:

```
<custom-resource>
    <name>coherence</name>
    <target>BEA_ENGINE_TIER_CLUST</target>
    <descriptor-file-name>custom/coherence.xml</descriptor-file-name>
    <resource-class>com.bea.wcp.sip.management.descriptor.
        resource.CoherenceStorageResource</resource-class>
    <resource-class>com.bea.wcp.sip.management.descriptor.resource.
        CoherenceStorageResource</resource-class>
    <descriptor-bean-class>oracle.occas.management.descriptor.beans.
        storage.CoherenceStorageBean</descriptor-bean-class>
</custom-resource>
```

The following parameters describe the **coherence.xml** file. They define a default call-state storage domain.

```
<?xml version='1.0' encoding='UTF-8'?>
<coherence-storage>
  <cache-config>
    <thread-count>20</thread-count>
    <partition-count>257</partition-count>
  </cache-config>
</coherence-storage>
```

Modifying the Call-State Storage Configuration

> **Note:** You cannot modify the configuration when servers in the domain are running.

To view and modify SIP call-state storage parameters:

1. Log in to the Administration Console for the WebRTC Session Controller administration server.
2. In the Domain Structure tree, click the **SipServer** node.
3. Click the **Configuration** tab.
4. Click the **Call State Storage** tab.

5. Enter values for **Thread Count** or **Partition Count** or both.
6. Click **Save**.

Table 11–1 describes the rules that apply to the Thread Count and Partition Count parameters:

Table 11–1 Call State Storage Configuration Parameters

Parameter	Type	Validation Rule	Restart Server?	Notes
Thread Count	integer	-1 to 32767	Yes	-1 = caller thread; 0 = service thread; otherwise, thread pool
Partition Count	integer	1 to 32767	Yes (all at the same time)	Must be prime number

The values are saved in the *domain_home*/**config/custom/coherence.xml** file, where *domain_home* is the root directory of the WebRTC Session Controller domain.

You can also set call-state storage parameters using WLST. See "Using WLST (JMX) to Configure WebRTC Session Controller" for more information.

Monitoring Coherence Call-State Storage

To monitor SIP call-state storage:

1. Log in to the Administration Console for the WebRTC Session Controller administration server.
2. In the Domain Structure tree, click **SipServer**.
3. Click the **Monitoring** tab.
4. Click the **Call State Storage** tab.
5. Click one of the following tabs, depending on the parameters you want to monitor:
   - Call State Service
   - Call State Cache
   - Call State Metadata Cache
   - Call State Index Cache

   Tables 11-2 through 11-5 describe the parameters that you can monitor on these tabs.

Table 11–2 describes the parameters that you can monitor on the **Service** tab for each server:

Table 11–2 Call State Service Monitoring Parameters

Column Name	MBean Attribute	Description
Local Messages	MessagesLocal	The total number of self-addressed messages since the last time the statistics were reset. These messages service process-local requests and do not have an associated network cost.
Received Messages	MessagesReceived	The total number of messages received by this service since the last time statistics were reset. This value accounts for messages received by any local, dedicated, or shared transport.

Table 11–2 (Cont.) Call State Service Monitoring Parameters

Column Name	MBean Attribute	Description
Sent Messages	MessagesSent	The number of messages sent by this service since the last time statistics were reset. This value accounts for any messages sent by local, dedicated, or shared transport.
Owned Backup Partitions	OwnedPartitionsBackup	The number of partitions that this member backs up (responsible for the backup storage).
Owned Primary Partitions	OwnedPartitionsPrimary	The number of partitions that this member owns (responsible for the primary storage).
Endangered Partitions	PartitionsEndangered	The total number of partitions that are not backed up.
Unbalanced Partitions	PartitionsUnbalanced	The total number of primary and backup partitions that remain to be transferred until the distribution across storage-enabled service members is fully balanced.
Vulnerable Partitions	PartitionsVulnerable	The total number of partitions that are backed up on the same machine where the primary partition owner resides.
Average Request Duration	RequestAverageDuration	The average duration in milliseconds of an individual synchronous request issued by the service.
Max Request Duration	RequestMaxDuration	The maximum duration in milliseconds of a synchronous request issued by the service.
Pending Request Count	RequestPendingCount	The number of pending synchronous requests issued by the service.
Average Task Duration	TaskAverageDuration	The average duration in milliseconds of an individual task execution.
Task Backlog	TaskBacklog	The size of the backlog queue that holds tasks scheduled to be executed by one of the service threads
Max Task Backlog	TaskMaxBacklog	The maximum size of the backlog queue.
Idle Thread Count	ThreadIdleCount	The number of currently idle threads in the service thread pool.

Table 11–3 describes the parameters that you can monitor on the **Call State Cache** tab for each server. The cache name is CallState.

Table 11–3 Call State Cache Monitoring Parameters

Column Name	MBean Attribute Name	Description
Entry Count	Size	The number of call-state objects currently stored.
Data Size	Units	The total number of bytes of call-state objects used for call-state objects currently stored.

Table 11–4 describes the parameters that you can monitor on the **Call State Metadata Cache** tab for each server. The cache name is CallState.meta. These are call-state lock and timer entries.

Table 11–4 Call State Cache Metadata Monitoring Parameters

Column Name	MBean Attribute Name	Description
Entry Count	Size	The number of call-state meta data objects.
Data Size	Units	The total number of bytes used for call-state meta data objects.

Table 11–5 describes the parameters that you can monitor on the **Call State Index Cache** tab for each server. The cache name is CallState.idx. These are call-state secondary index entries.

Table 11–5 Call State Index Cache Monitoring Parameters

Column Name	MBean Attribute Name	Description
Entry Count	Size	The number of call-state secondary index entries currently stored.
Data Size	Units	The total number of bytes of call-state secondary index entries currently stored.

You can monitor all parameters by connecting directly to the servers using JConsole.

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:

12

Logging SIP Requests and Responses and EDRs

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.

In addition, for each engine you can enable logging of event detail records to the **oracle.wsc.core.edr** file. See "Accessing Event Detail Records" for more information.

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 12–1, " Pattern-matching Variables and Sample Values" lists the parameters and parameter values that you can specify as **init-param** entries.

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 12–1.

Example 12–1 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 12–1, require a variable (**var** element) that identifies the portion of the SIP message to evaluate. Table 12–1 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; WebRTC Session Controller enables mapping of both request and response variables to logging servlets.

Table 12–1 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.xm** use the same Document Type Definition (DTD). See "trace-pattern.dtd Reference" for more information.

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 12–2 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 12–2 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 12–3 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 12–4.

Example 12–4 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.

Accessing Event Detail Records

The Signalling Engine collects data in an event detail record (EDR) for each event that occurs in a subsession for the Call, Chat, and File Transfer packages. Each engine creates event detail records and writes them to a file, **oracle.wsc.core.edr**, for the sessions that it owns.

An event detail record consists of a number of attributes whose values are written to the file, separated by commas. The following example shows the format of a call event detail record:

```
"call", "Event Data Record for call", "guest",
"wlss-ffc87c6f-03a1b947a75bee7641d1e6caa71af17e@127.0.0.1", "alice@example.com",
"bob@example.com", "guest4817396667549633 47", "0", "Mon Aug 04 22:25:44 IST 2014",
"603"
```

Table 12–2 shows the supported list of attributes in the order in which they appear in an EDR:

Table 12–2 EDR Attributes

Attribute	Value Description	Data Type	Example
EVENT_NAME	call/chat/file_transfer	String	"call"
DESCRIPTION	Description	String	"Event Data Record for call"
APPLICATION	Application for which EDR is generated	String	"guest"
SESSION_ID	Unique subsession ID (call-id)	String	"wlss-ffc87c6f-03a1b947a75bee7641d1e6caa71af17e@127.0.0.1"
INITIATOR	Initiator of subsession	String	"alice@example.com"
TARGET	Target of the subsession	String	"bob@example.com"
USER_ID	Web user ID	String	"alice@web.net"
SESSION_DURATION	Duration of the subsession in seconds	long	61
START_TIME	Start time for the subsession	Date	Mon Aug 04 22:25:44 IST 2014
FAILURE_REASON	Reason for the failure.	Int	603

Note: The actual start time of the call might be perceived differently by the user. The Signalling Engine cannot identify the time that the media packets are sent between two users because RTP packets are not routed through it. With WebRTC, packets can go directly between the browsers. Consequently, the session duration time is based on the Signaling Engine's perception of the session, not the exact duration of the media packet flow.

You enable EDR logging in the administration console.

You can specify additional configuration options in the **edr-log4j2-conf.xml** file. In this file is missing, EDRs are logged to the **WSC.log** file by default.

Table 12–5 lists the contents of a sample **edr-log4j2-conf.xml** file:

Example 12–5 EDR Logging Configuration Options in the edr-log4j2-conf.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
       <Appenders>
               <RollingFile  name="file"
fileName="servers/${sys:weblogic.Name}/logs/wsc.log"
filePattern="servers/${sys:weblogic.Name}/logs/wsc-%d{yyyyMM}-%i.log">
                 <PatternLayout pattern="%5p %d [%-15.15t] (%-25.25c:%4L) - %m%n"
/>
                 <Policies>
                       <TimeBasedTriggeringPolicy />
                       <SizeBasedTriggeringPolicy size="100 MB"/>
                </Policies>
               </RollingFile>

               <RollingFile  name="edr.file"
fileName="servers/${sys:weblogic.Name}/logs/edr.log"
filePattern="servers/${sys:weblogic.Name}/logs/edr-%d{yyyyMM}-%i.log">
```

```
                <PatternLayout pattern="%m%n" />
                <Policies>
                      <TimeBasedTriggeringPolicy />
                      <SizeBasedTriggeringPolicy size="100 MB"/>
                </Policies>
              </RollingFile>
       </Appenders>

       <Loggers>
              <Logger name="oracle.wsc.core.edr.EventDataRecorder" level="debug"
additivity="false">
                      <AppenderRef ref="edr.file"/>
              </Logger>
              <Root level="info">
                <AppenderRef ref="file" />
              </Root>
       </Loggers>

</Configuration>
```

You provide the name of the **log4j** file as part of configuration. For information on **log4j** configuration, see

http://logging.apache.org/log4j/2.x/faq.html#config_location.

For information about the **log4j** file, see

http://logging.apache.org/log4j/2.x/manual/configuration.html.

Managing EDRs in a Multitenancy Scenario

WebRTC Session Controller stores the tenant information in log files named with the tenant name. For example, the EDR of tenant A whose tenant name is *tenantA* is recorded in file named **edr-***tenantA***.log**.

WebRTC Session Controller RoutingAppender to route EDRs of tenants to the appropriate log file associated with each tenant. Example 12–6 shows an example of an appender configuration.

Example 12–6 An example Appender Configuration

```
<Routing name="edr.file">
      <Routes pattern="$${ctx:tenantName}">
        <!-- This route is chosen if ThreadContext has no value for key
tenantName. -->
        <Route key="$${ctx:tenantName}">
          <RollingFile name="edr-default"
fileName="servers/${sys:weblogic.Name}/logs/edr.log"

filePattern="servers/${sys:weblogic.Name}/logs/edr-%d{yyyyMM}-%i.log">
            <PatternLayout pattern="%m%n" />
            <Policies>
              <TimeBasedTriggeringPolicy />
              <SizeBasedTriggeringPolicy size="100 MB"/>
            </Policies>
          </RollingFile>
        </Route>

        <!-- This route is chosen if ThreadContext has a value for tenantName
             The value dynamically determines the name of the log file. -->
        <Route>
```

```
            <RollingFile name="Rolling-${ctx:tenantName}"
          fileName="servers/${sys:weblogic.Name}/logs/edr-${ctx:tenantName}.log"

          filePattern="servers/${sys:weblogic.Name}/logs/edr-${ctx:tenantName}-%d{yyyyMM}-%i
          .log">
                      <PatternLayout pattern="%m%n" />
                      <Policies>
                        <TimeBasedTriggeringPolicy />
                        <SizeBasedTriggeringPolicy size="100 MB"/>
                      </Policies>
                    </RollingFile>
                  </Route>
                </Routes>
              </Routing>
```

Table 12–7 lists the contents of a sample **edr-log4j2-conf.xml** file configured to support multitenancy:

Example 12–7 Sample edr-log4j2-conf.xml File (Multitenancy)

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">

    <Appenders>
     <RollingFile
          name="file"
          fileName="servers/${sys:weblogic.Name}/logs/wsc.log"
         filePattern="servers/${sys:weblogic.Name}/logs/wsc.log-%d{yyyy-MM-dd}-%i"
          immediateFlush="true"
           append="true">

        <PatternLayout pattern="%5p %d [%-15.15t] (%-25.25c:%4L) - %m%n" />
        <Policies>
          <OnStartupTriggeringPolicy />
          <TimeBasedTriggeringPolicy interval="24" modulate="true" />
          <SizeBasedTriggeringPolicy size="100 MB"/>
        </Policies>
        <DefaultRolloverStrategy max="10"/>
      </RollingFile>
      <RollingFile
              name="threat"
              fileName="servers/${sys:weblogic.Name}/logs/wsc-threat.log"
              filePattern="servers/${sys:weblogic.Name}/logs/wsc-threat.log-%d{yyyy-MM-dd}-%i"
              immediateFlush="true"
             append="true">

        <PatternLayout pattern="%d %m%n" />
        <Policies>
          <OnStartupTriggeringPolicy />
          <TimeBasedTriggeringPolicy interval="24" modulate="true" />
          <SizeBasedTriggeringPolicy size="100 MB"/>
        </Policies>
        <DefaultRolloverStrategy max="10"/>
      </RollingFile>

      <RollingFile
              name="debug"
              fileName="servers/${sys:weblogic.Name}/logs/wsc-debug-server.log"

filePattern="servers/${sys:weblogic.Name}/logs/wsc-debug-server.log-%d{yyyy-MM-dd}-%i"
```

```
               immediateFlush="true"
               append="true">

         <PatternLayout pattern="%5p %d [%-15.15t] (%-25.25c:%4L) - %m%n" />
         <Policies>
           <OnStartupTriggeringPolicy />
           <TimeBasedTriggeringPolicy interval="24" modulate="true" />
           <SizeBasedTriggeringPolicy size="100 MB"/>
         </Policies>
         <DefaultRolloverStrategy max="10"/>
       </RollingFile>

       <RollingFile
               name="clientDebug"
               fileName="servers/${sys:weblogic.Name}/logs/wsc-debug-client.log"

filePattern="servers/${sys:weblogic.Name}/logs/wsc-debug-client.log-%d{yyyy-MM-dd}-%i"
               immediateFlush="true"
               append="true">

         <PatternLayout pattern="%5p %d [%-15.15t] (%-25.25c:%4L) - %m%n" />
         <Policies>
           <OnStartupTriggeringPolicy />
           <TimeBasedTriggeringPolicy interval="24" modulate="true" />
           <SizeBasedTriggeringPolicy size="100 MB"/>
         </Policies>
         <DefaultRolloverStrategy max="10"/>
       </RollingFile>

       <Routing name="edr.file">
         <Routes pattern="$${ctx:tenantName}">
           <!-- This route is chosen if ThreadContext has no value for key tenatName. -->
           <Route key="$${ctx:tenantName}">
             <RollingFile name="edr-default" fileame="servers/${sys:weblogic.Name}/logs/edr.log"
filePattern="servers/${sys:weblogic.Name}/logs/edr-%d{yyyyMM}-%i.log">
               <PatternLayout pattern="%m%n" />
               <Policies>
                 <TimeBasedTriggeringPolicy />
                 <SizeBasedTriggeringPolicy size="100 MB"/>
               </Policies>
             </RollingFile>
           </Route>

           <!-- This route is chosen if ThreadContext has a value for tenantName
                The value dynamically determines the name of the log file. -->
           <Route>
             <RollingFile name="Rolling-${ctx:tenantName}"
fileName="servers/${sys:weblogic.Name}/logs/edr-${ctx:tenantName}.log"

filePattern="servers/${sys:weblogic.Name}/logs/edr-${ctx:tenantName}-%d{yyyyMM}-%i.log">
               <PatternLayout pattern="%m%n" />
               <Policies>
                 <TimeBasedTriggeringPolicy />
                 <SizeBasedTriggeringPolicy size="100 MB"/>
              </Policies>
             </RollingFile>
           </Route>
         </Routes>
       </Routing>
     </Appenders>
```

```
  <Loggers>
    <Logger name="oracle.wsc.core.edr.EventDataRecorder" level="debug" additivity="false">
       <AppenderRef ref="edr.file"/>
     </Logger>
     <Logger name="oracle.wsc.core.threat.log" level="info" additivity="false">
       <AppenderRef ref="threat"/>
     </Logger>
     <Logger name="oracle.wsc.core.debug" level="debug" additivity="false">
      <AppenderRef ref="debug"/>
    </Logger>
     <Logger name="oracle.wsc.core.clientdebug" level="debug" additivity="false">
       <AppenderRef ref="clientDebug"/>
     </Logger>
     <Root level="warn">
       <AppenderRef ref="file" />
     </Root>
   </Loggers>

</Configuration>
```

13

Monitoring Statistics and Resource Limits

This chapter describes the implementation and management of statistics and resource limits in Oracle WebRTC Session Controller.

About WebRTC Session Controller Statistics

You configure your WebRTC Session Controller installation with license requirements, resource limits for the system and the application or tenant accessing the system and its services.

At runtime, when your customers access and use your WebRTC Session Controller system and the services you provide, you can access and monitor the following in WebRTC Session Controller:

- Licenses. See "About the Monitoring of Licenses".
- Resource limits. See "About Resource Limits".
- Counters. See "About Statistics Counters".

About the Monitoring of Licenses

As a system administrator, you can ensure that WebRTC Session Controller is not exceeding the licensing limit for concurrent sessions and/or named users by monitoring the total number of concurrent sessions and/or named users in the system at any time.

In WebRTC Session Controller, concurrent sessions are the aggregate number of established virtual connections between two endpoints represented by subscriber devices or network switching equipment and traversing the licensed software at any one time.

A named user is an individual authorized by you to use the programs which are installed on a single server or multiple servers. This definition of a named user is valid regardless of whether the individual is actively using the programs at any given time. Additionally, WebRTC Session Controller counts a non human operated device that can access the programs as a named user in addition to all individuals authorized to use the programs.

WebRTC Session Controller supports the following licensing metrics at the system level:

- Total number of active sessions
- Total number of sub-sessions
- Total number of uniquely-named users

- High watermark for active sessions, sub sessions and named users

To monitor these metrics, see "Monitoring the Metrics".

About Resource Limits

WebRTC Session Controller Administration Console supports the configuration of resource limit entries, where each entry is given a unique name. Each resource limit entry consists of the following resource parameters to which limits can be set:

- Number of active sessions allowed.
- Number of sessions allowed per user.
- Number of sub sessions allowed per session.
- Number of sub sessions allowed per user.

Each tenant of an WebRTC Session Controller installation can configure and store a set of resource limit entries in the Signaling Engine. At the time that a tenant creates an entry for an application and configures the application profile for the application, the tenant selects a resource limit entry to associate with each application.

WebRTC Session Controller provides a default resource limit selection entry called, **default**. Use this resource limit entry to provide resource limits as a default selection for the applications a tenant configures in the application profile.

About the default Resource Limit Entry

When WebRTC Session Controller is installed, the four resource parameters for the **default** entry have their limits set to **-1**. A value of **-1** for a resource entry parameter indicates that the resource parameter is allowed unlimited resource. WebRTC Session Controller does not track any resource parameter with its maximum limit set to **-1**.

You can edit the limits set for the **default** resource limit entry. When you do so, WebRTC Session Controller enforces the updated resource limits.

Note: If you do not edit the installation configuration for **default** (with unlimited resources for all the parameters) as one of the resource limit entries and an application uses this default, then, none of the resource parameters use is tracked for the application.

About Statistics Counters

Table 13–1 lists the available WebRTC Session Controller statistics counters, including information on each statistic's type as well as in which statistics set each is available.

Table 13–1 WebRTC Session Controller Statistics Counters

Name	Levels Where Applied	Description
SESSION_COUNT	System, Application, Tenant, Application User, Tenant user	Number of active WebRTC Session Controller sessions
SESSION_PEAK	System	Maximum number of active WebRTC Session Controller sessions
SUB_SESSION_COUNT	System	Number of active sub sessions, for example SIP sessions.

Table 13–1 (Cont.) WebRTC Session Controller Statistics Counters

Name	Levels Where Applied	Description
SUB_SESSION_PEAK	System	Max number of active sub sessions, for example, SIP sessions.
SUB_SESSION_ATTEMPT_COUNT	System	The number of SIP session creation attempts where SipSession == INITIAL.
SUB_SESSION_SUCCESS_COUNT	System	The number of SIP sessions successfully created where SipSession == CONFIRMED.
SUB_SESSION_TERMINATE_ COUNT	System	The number of SIP sessions terminated where SipSession == TERMINATED.
REGISTRATION_SUCCESS_COUNT	System	Number of successful registrations.
REGISTRATION_UNSUCCESS_ COUNT	System	Number of unsuccessful registrations.
MEDIA_SESSION_COUNT	System	The number of active media sessions.
ACTIVE_USERS_COUNT	System	Number of active users
ACTIVE_USERS_PEAK	System	Maximum number of active users
SIP_MESSAGE_SEND_COUNT	System	The number of SIP messages sent by WebRTC Session Controller.
SIP_MESSAGE_RECV_COUNT	System	The number of SIP messages received by WebRTC Session Controller.
SIP_(method)_REQUEST_SEND_ COUNT	System	The number of SIP requests for each method sent by WebRTC Session Controller.
SIP_(method)_REQUEST_RECV_ COUNT	System	The number of SIP requests for each method received by WebRTC Session Controller.
SIP_(status)_RESPONSE_SEND_ COUNT	System	The number of SIP responses of each status (1xx, 2xx, 3xx, 4xx, 5xx and 6xx) sent by WebRTC Session Controller.
SIP_(status)_RESPONSE_RECV_ COUNT	System	The number of SIP responses of each status 1xx, 2xx, 3xx, 4xx, 5xx and 6xx) received by WebRTC Session Controller.

For a description about the MBeans employed to monitor these counters, see "About StatisticsRuntimeMBean" and "About the SipRuntimeMBean".

Configuring Resource Limits

You can configure resource limits for applications and for tenants in the Signaling Engine.

Configuring Resource Limits in the Signaling Engine

In the **Signaling Engine** tab of WebRTC Session Controller Administration Console, you create a set of resource limit entries, with each entry given a unique name. For each resource limit entry, you provide limits to the resource parameters.

For information on how to provide resource limit entries for the Signaling engine, see "Global Resource Limit Parameters of the Signaling Engine".

Configuring Resource Limits for Applications

The **Resource Limits** field in the **Application Profiles** tab of WebRTC Session Controller Administration Console, displays the set of saved resource limit entries that you created in the **Signaling Engine** tab. When you create the application profile for an application, you select the resource limit entry that best suits your application.

For information on how to assign a resource limit entry to your application, see "Providing the Profile Information for the Application".

Monitoring the Metrics

You can monitor these metrics:

- System usage. See "Monitoring the System at RunTime".
- Sip Server usage. See "Monitoring SIP Counters at Runtime".
- High watermark log messages. See "Monitoring High Watermark Log Messages".

Monitoring the System at RunTime

At runtime, you can get the statistics for:

- Usage:
  - SESSION_COUNT
  - SESSION_PEAK
  - SUB_SESSION_COUNT
  - SUB_SESSION_PEAK
  - SUB_SESSION_ATTEMPT_COUNT
  - SUB_SESSION_SUCCESS_COUNT
  - SUB_SESSION_TERMINATE_COUNT
  - REGISTRATION_SUCCESS_COUNT
  - REGISTRATION_UNSUCCESS_COUNT
  - MEDIA_SESSION_COUNT
  - ACTIVE_USERS_COUNT
  - ACTIVE_USERS_PEAK

  See Table 13–1 for a description of the counters.
- High Watermark counts for:
  - Active named users
  - Active sessions
  - Active sub sessions

To retrieve the above statistics, you use the **StatisticsRuntimeMBean**.

About StatisticsRuntimeMBean

As its name suggests, **StatisticsRuntimeMBean** is a runtime MBean that provides statistics. The following MBeans are available at runtime:

- System level

A single MBean instance that reports counters from the SYSTEM statistics set with the object name:

```
oracle.wsc:Type=StatisticsRuntimeMBean,Location=[ServerName]]
```

- Application level

  One mbean per application that reports counters from the Application statistics set with the Object name

  ```
  oracle.wsc:Type=StatisticsRuntimeMBean,Location=[ServerName],scope=application,
  application=[ApplicationName]]
  ```

- Tenant level

  One mbean per tenant that reports counters from the TENANT statistics set with the Object name:

  ```
  oracle.wsc:Type=StatisticsRuntimeMBean,Location=[ServerName],scope=tenant,tenan
  tProfile=[TenantProfileName]]]
  ```

You can access WebRTC Session Controller statistics counters and operations using JMX or the JConsole utility through this MBean.

For information on the methods of the **StatisticsRunTimeMBean**, see *WebRTC Session Controller JavaScript API Reference*.

Monitoring SIP Counters at Runtime

At runtime, **SipRunTimeMBean** enables you can retrieve the number for following:

- SIP_MESSAGE_SEND_COUNT
- SIP_MESSAGE_RECV_COUNT
- SIP_(method)_REQUEST_SEND_COUNT
- SIP_(method)_REQUEST_RECV_COUNT
- SIP_(status)_RESPONSE_SEND_COUNT
- SIP_(status)_RESPONSE_RECV_COUNT

About the SipRuntimeMBean

SipRuntimeMBean is a single mbean instance reflects SIP counters from the SYSTEM statistics set with the Object name

```
oracle.wsc:Type=SipRuntimeMBean,Location=[ServerName]]]]
```

For information on the methods of the **SipRuntimeMBean**, see *WebRTC Session Controller JavaScript API Reference*.

Monitoring High Watermark Log Messages

The High Watermark is the indicator which represents the highest value seen until now for a monitored entry. Suppose that the total count stored for a monitored entry goes from 1 to 10 to 5. The high watermark value for this entry is 10.

WebRTC Session Controller logs a high watermark message for the following:

- Active sessions
- Active sub sessions

- Active named users

Example 13–1 lists some warning messages in a sample **wsc.log** file:

Example 13–1 Example High Watermark messages in the wsc.log file

```
WARN 2014-03-28 13:00:31,452 [pool-4-thread-1] (ats.StatsServiceImpl:  96) - New high watermark for
active named user count: 42
WARN 2014-03-25 16:18:55,816 [pool-4-thread-1] (ats.StatsServiceImpl: 125) - New high watermark for
active session count: 99
```

When the system reaches a new all time high value for any of the licence metrics, WebRTC Session Controller logs high watermark messages in the **wsc.log** file on the engine servers. The complete path to this file is *domain_name*/**servers/***server_name*/**logs/wsc.log** file, where *domain_name* is the name of the WebRTC Session Controller domain and *server_name* is the name of the server.

You can audit the log files at a later time to see the maximum values that have been reached.

Note: In a cluster configuration, WebRTC Session Controller resets the watermark values on a restart of the servers.

Disabling the Monitoring of System Statistics

WebRTC Session Controller collects statistics associated with licensing, by default. To disable the monitoring of system statistics, use the following system property when you start WebLogic server:

-Doracle.wsc.stats=false

For information about startup command options, see Table 3–6.

14

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.

High Availability

High availability refers to a system design that eliminates or minimizes the amount of time that a system is inaccessible due to some type of system failure.

WebRTC Session Controller achieves high availability primarily due to the features of the underlying Weblogic Server platform. These features include:

- WebLogic Server clusters that distribute the work load among the multiple instances of WebLogic Server running on the nodes in the cluster. In the event of failure, the session state of the failed WebLogic Server is available to other node that can continue the work. If the cluster is configured correctly, services can also migrate to another node in the event of failure. See "Understanding Weblogic Server Clustering" in *Administering Clusters for Oracle WebLogic Server* for more information.
- Coherence clusters that distribute data across members to ensure that data is always available. See "Configuring and Managing Coherence Clusters" in *Administering Clusters for Oracle WebLogic Server* for more information.
- Overload protection that enables WebLogic Server to detect and recover from overload conditions. See "Avoiding and Managing Overload" in *Administering Server Environments* for more information.

- Network channels that segregate traffic by type to use resources effectively. See "Configuring Network Resources" in *Administering Server Environments* for more information
- Work Managers that optimize and prioritize work based on rules and performance statistics. See "Using Work Managers to Optimize Scheduled Work" in *Administering Server Environments* for more information.

You can also use virtual machines (VMs) to mitigate system failure. An individual server has multiple points of potential failure, including CPU, RAM, network ports, and disk drives. A virtual machine, on the other hand, can satisfy its resource requirements from a pool of hardware resources so that a physical disk failure does not result in a failure of the virtual disk. The virtual machine simply employs another available disk drive to compensate for the one that failed. A balanced deployment of VMs running separate Signalling Engines and Media Engines on different hosts can take full advantage of cross-host high availability for both Signalling Engine and Media Engine clusters.

For information on installing a Media Engine cluster to support redundancy and failover, high-availability, and load balancing, see the sections on installing media engine clusters in the *WebRTC Session Controller Installation Guide*.

Overload Protection

There are two sets of tuning parameters related to overload protection, one set for the SIP side and another set for the HTTP or WebSocket side. For WebRTC Session Controller, the greater threats are from the HTTP (Internet) side.

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 "Engine Server 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 servers and partitions in a dedicated cluster.

Server partitions store redundant copies of call state information, and automatically failover to one another should a partition or server fail.

See *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, you can 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.

There is no specific configuration in WebRTC Session Controller to support redundant sites. They are two independent sites that are not aware of each other, which means that you need to configure and provision each site manually.

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 call-state storage 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 creating a full snapshot of the domain at regular intervals. For example, you might want to create a snapshot when the following events occur:

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

> **Note:** The domain directory is present on the Administration Server and each Managed Server but the Administration Server has the master copy, which you must back up. You do not need to back up any files on a Managed Server.

The WebLogic `pack` command creates a template archive file (.jar) based on an existing WebLogic domain. For example, the following command creates a template file called **C:\oracle\user_templates\mydomain.jar**.

```
pack -domain=C:\oracle\user_projects\domains\mydomain -template=C:\oracle\user_
templates\mydomain.jar -template_name="My WebLogic Domain"
```

The name of the template is My WebLogic Domain.

See *Creating Templates and Domains Using the Pack and Unpack Commands* for information on using the `pack` and `unpack` commands.

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

Backing Up Logging Servlet Applications

If you use WebRTC Session Controller logging Servlets (see "Logging SIP Requests and Responses and EDRs") 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 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 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).apply any patches that had been applied to the failed server.
2. Apply any patches that had been applied to the failed server.
3. Use the `unpack` command to create a WebLogic domain from the template that you created when you backed up the domain. See "Storing the Domain Configuration Offline" for more information. See *Creating Templates and Domains Using the Pack and Unpack Commands* for more information on the `pack` and `unpack` commands.

   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.
4. 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".
5. 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 *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".

15

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 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.

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 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.

16

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 Server Configuration Reference (sipserver.xml)
- SIP Coherence Configuration Reference (coherence.xml)
- Diameter Configuration Reference (diameter.xml)

17

Engine Server Configuration Reference (sipserver.xml)

This chapter describes the Oracle Communications WebRTC Session Controller engine server 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 a server installation. The **sipserver.xml** file 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 6, "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 6, "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 servers process expired timers. By default (when enable-timer-affinity is omitted from **sipserver.xml**, or is set to **false**), an engine server that polls the SIP call-state store for expired timers might process all available expired timers. When enable-timer-affinity is set to **true**, engine servers polling the SIP call-state store 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 17–1.

Table 17–1 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 17–1 (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 17–1 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 17–1 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 12, "Logging SIP Requests and Responses and EDRs".

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 17–2, " 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 17–2 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 17–2 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 17–2 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'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 engine servers, 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 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 cluster configured for an upgrade. WebRTC Session Controller uses this mapping to ensure that engine 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 17–3.

Table 17–3 Nested cluster-loadbalancer-map Elements

Element	Description
cluster-name	The configured name of an engine cluster.
sip-uri	The internal SIP URI that maps to the engine 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 17–3 shows a sample **cluster-loadbalancer-map** entry used during an upgrade.

Example 17–3 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 the section on upgrading production WebRTC Session Controller software in the *WSC Installation Guide* 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 17–4 shows a sample **sip-security** configuration.

Example 17–4 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 servers to cache a portion of the call-state data locally, to improve performance with SIP-aware load balancers. When a local cache is used, an engine 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 call-state store), the engine locks the call state in the SIP call-state store but reads directly from its cache.

By default the engine 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 10, "Using the Engine 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 17–4.

Table 17–4 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 call-state store. 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 cluster of the remote installation. You can also specify a comma-delimited list of addresses corresponding to each engine server. The URLs must specify the t3 protocol.

Example 17–5 shows a sample configuration that uses RDBMS storage for long-lived call state and geographically-redundant replication. Call states are replicated to two engine servers in a remote location.

Example 17–5 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 17–5.

Table 17–5 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 17–5 (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 17–6 shows a sample connection-reuse-pool configuration having two pools.

Example 17–6 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.

18

SIP Coherence Configuration Reference (coherence.xml)

This chapter describes the Coherence configuration file, coherence.xml, for Oracle Communications WebRTC Session Controller.

Overview of coherence.xml

The **coherence.xml** configuration file identifies servers that manage the concurrent call state for SIP applications, and specifies distributed cache settings. See "Configuring Coherence" for information on configuring Coherence.

The **coherence.xml** file resides in the *domain_home*/**config/custom** subdirectory where *domain_home* is the root directory of WebRTC Session Controller domain.

Editing coherence.xml

You can edit **coherence.xml** using either the Administration Console or a text editor. Changes to the configuration cannot be applied to servers dynamically; you must restart servers to change the SIP server configuration.

XML Schema

The 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 coherence.xml File

Example 18–1 shows the default **coherence.xml** file.

Example 18–1 Default coherence.xml File

```
<?xml version='1.0' encoding='UTF-8'?>
<coherence-storage>
  <cache-config>
    <thread-count>20</thread-count>
    <partition-count>257</partition-count>
  </cache-config>
</coherence-storage>
```

XML Element Description

Table 18–1 describes the elements in the coherence.xml file that govern the Coherence distributed cache service.

Table 18–1 coherence.xml File Elements

Element	Description
thread-count	Specifies the number of threads used in the call-state Coherence cache service used by the SIP server. Oracle recommends that this value be a positive integer but you can specify 0 or -1 to obtain specific behaviors. See the `thread-count` element description in "Cache Configuration Elements" in *Developing Applications with Oracle Coherence* for more information.
partition-count	Specifies the number of partitions used in the call-state Coherence cache service used by the SIP server. You must specify a positive integer and should specify a prime number. See the `partition-count` element description in "Cache Configuration Elements" in *Developing Applications with Oracle Coherence* for more information.

19

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 19–1 shows the element hierarchy of the **diameter.xml** file.

Figure 19–1 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 6, "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 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".

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".

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 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.

Part IV

WebRTC Session Controller Media Engine Administration

This part provides administration information for Oracle Communications WebRTC Session Controller Media Engine.

This part contains the following chapters:

20

Managing and Administering ME Systems

The chapter describes the administrator tasks that you can perform when managing a new WebRTC Session Controller Media Engine (ME) system. Before using the information in this guide, be sure that you have properly installed the ME, as covered in the *WebRTC Session Controller Installation Guide*.

References

For detailed descriptions of the commands that you can use for administrative tasks, as well as instructions for using the management interfaces, refer to the *Oracle Communications WebRTC Session Controller Media Engine Object Reference*.

For information on configuring policies, refer to the *Oracle Communications OS-E Session Services Configuration Guide*.

Administrator and User Roles

The *administrator* is any person who configures and manages the ME system in the network.

The *user* is a SIP client, usually a VoIP call sender or receiver, of SIP messages that are transmitted to, and over the ME system to a destination. A SIP user may have one or more SIP URIs in SIP sessions that traverse the platform between the user's originating SIP application or device and the SIP server endpoint (such as Microsoft LCS, IBM Sametime, Avaya, etc.). SIP clients who establish SIP sessions are subject to SIP policies that are configured by the ME administrator.

Enabling Management Access

When you create one or more administrative users, the ME prompts for a username and password when anyone attempts to log in. Administrative users have read/write management access to the ME configuration file. Editing and saving the configuration file updates the ME configuration file named *cxc.cfg*. If desired, administrators can commit the configuration changes to the running ME configuration.

CLI Session

The following CLI session creates a user and password (with permissions) for management access across the entire ME system.

```
NNOS-E> config access
config access> config users
Creating 'users'
```

```
config users> config user "jane doe"
Creating 'user "jane doe"'
config user "jane doe"> set password abcXYZ
confirm:******************
config user "jane doe"> set permissions access permissions grant
Creating 'access\permissions grant'
config user "jane doe"> return
config users> return
config access> config permissions grant
Creating 'permissions grant'
config permissions grant> set ftp enabled
config permissions grant> set cms enabled-web-only
config permissions grant> set cli normal
config permissions grant> set config enabled
config permissions grant> set call-logs enabled
config permissions grant> set actions enabled
config permissions grant> set status enabled
config permissions grant> set user-portal enabled
config permissions grant> set web-services enabled
```

If you are using the CMS to configure administrative users and permissions, use the CMS Access tab.

For more information on the **access** configuration object and the other properties that you can configure, refer to the *WebRTC Session Controller Media Engine Objects and Properties Reference Guide*.

Configuring Management Options

This section shows you how to set up the management options that allow you to configure the ME system.

Local Console

If you are using a directly-attached local console or terminal to configure the ME for the first time, use a terminal emulation program such as HyperTerminal to set the console parameters.

The following CLI session configures the console settings for communicating with the ME system. The example session shows the console default settings.

CLI Session

```
config> config box
config box> config console
config box> set rate 115200
config box> set data-bits 8
config box> set parity none
config box> set stop-bits 1
config box> set flow-control none
```

Telnet

Telnet is a standard TCP/IP-based terminal emulation protocol defined in RFC 854, *Telnet Protocol Specification.* Telnet allows a remote user to establish a terminal connection to the ME system over an IP network. By default, the Telnet protocol is enabled at installation time. To allow connections over Telnet, you must configure those users who are allowed access to the ME over Telnet.

The following CLI session configures the Telnet protocol on the local ME system, including the maximum number of concurrent Telnet sessions, the idle timeout period (in seconds) that ends a Telnet session due to inactivity, and the known TCP port for inbound and outbound Telnet messages.

CLI Session

```
config box> config interface eth0
config interface eth0> config ip local
config ip local> config telnet
config telnet> set admin enabled
config telnet> set max-sessions 10
config telnet> set idle-timeout 600
config telnet> set port 23
```

Secure Shell (SSH)

Secure Shell (SSH) Server Version 2 on the ME system provides secure client/server communications, remote logins, and file transfers using encryption and public-key authentication. To establish a secure connection and communications session, SSH uses a key pair that you generate or receive from a valid certificate authority (CA). By default, SSH is enabled at installation time.

An SSH session allows you to transfer files with Secure Shell File Transfer Protocol (SFTP), providing more secure transfers than FTP and an easy-to-use interface. SSH uses counters that record SFTP activity over the SSH connection.

When running SSH on the ME system, the SSH session is transparent and the CLI appears just as it would if you were connecting from a console or over Telnet. The ME implementation of SSH does not support all the user-configurable parameters typically supported by SSH workstations. If you try to change a parameter that the ME does not support, you will receive a notification that the parameter setting failed.

CLI Session

The following CLI session configures the SSH protocol on the local ME system, including the maximum number of concurrent SSH sessions, the idle timeout period (in seconds) that ends an SSH session due to inactivity, and the known TCP port for inbound and outbound SSH messages.

```
config box> config interface eth0
config interface eth0> config ip local
config ip 1ocal> config ssh
config ssh> set admin enabled
config ssh> set max-sessions 10
config ssh> set idle-timeout 600
config ssh> set port 22
```

Web/HTTP

The ME Management System allows you to configure and manage the ME system remotely using your web browser.

The ME interface supports all management capabilities provided by the CLI. Instead of entering information on a command line, you navigate menus and supply information in menu fields.

To manage the ME system over the Web, enter the IP address of the management IP interface in the Internet Explorer **File/Open** command window and log in. For example:

http://192.168.124.1/

CLI Session

The following CLI session enables Web access to the local ME and specifies the TCP port over which HTTPS traffic is sent and received on the IP interface.

```
config box> config interface eth0
config interface eth0> config ip local
config ip local> config web
config web> set admin enabled
config web> set protocol https 443
```

For detailed on using the CMS, refer to the *SIP Security and Management Solutions – System Management Reference*.

SNMP

The Simple Network Management Protocol (SNMP) allows you to communicate with the SNMP agent on the ME system from a remote management station. SNMP allows you to retrieve information about managed objects on the platform as well as initiate actions using the standard and enterprise Management Information Base (MIB) files that Oracle makes available with the product software.

The ME supports the SNMP versions SNMP v1 and SNMP v2c.

CLI Session

The following CLI session enables SNMP access to the local ME system, specifies the TCP port over which SNMP traffic is sent and received on the management interface, sets the SNMP community string, the SNMP version, and the target system IP address to which SNMP trap messages are forwarded.

```
config box> config interface eth0
config interface eth0> config ip local
config ip local> config snmp
config snmp> set admin enabled
config snmp> set port 161
config snmp> set version 2c
config snmp> set community private
config snmp> set trap-target 192.168.124.10
```

HTTP\SOAP\WSDL Interface

The ME software includes a software development kit (SDK) to provide Web Services Description Language (WSDL) accessibility to the ME.

WSDL is an XML-based language for describing Web services, and how to access them, in a platform-independent manner. Simple Object Access Protocol (SOAP) is a communication protocol for communication between applications, based on XML.

A WSDL document is a set of definitions that describe how to access a web service and what operations it will perform. The ME uses it in combination with SOAP and an XML Schema to allow a client program connecting to a web service to determine available server functions. The actions and data types required are embedded in the WSDL file, which then may be enclosed in a SOAP envelope. The SOAP protocol supports the exchange of XML-based messages, with the ME using HTTPS.

The ME performs the role of a web service server in the WSDL exchange, where an external client can make web service requests on the ME system.

The WSDL document (and its imported schema files, such as cxc.xsd) define every possible request and response provided for the service, including error responses. Depending on how you choose to integrate with the ME system, you can use the ME SDK (using Java) or you can simply take the WSDL document and generate tools in your desired language. Because web services are language independent, you can use virtually any modern language to generate the requests and the WSDL document defines what those requests need to look like for the receiving component.

For complete information on the WSDL interface, refer to the *Net-Net OS-E – Management Tools*.

Working with the ME Configuration File

All ME systems use the startup configuration file named *cxc.cfg*. This file defines all aspects of the ME system and its configuration in the network.

- Ethernet interfaces (and their IP addresses) connecting the platform to the Ethernet switches and the Internet
- Configured protocols, services, accounting and logging
- Policies that define the rules and conditions to match with SIP enterprise the carrier traffic requests.

Building the Configuration File Using the CLI

The ME configuration file (*cxc.cfg*) is made up of configuration objects and property settings that control how the system processes and manages SIP traffic. As you open these objects and set properties using the CLI (or the CMS), the ME builds a configuration hierarchy of objects that are applied to SIP sessions. You can display this configuration hierarchy using the **show** and **show -v** commands.

For new users, as well as for users who are adding functionality to their configuration, you will need to open configuration objects using the **config** command to enable the default settings for those objects, even if you choose not to edit any of their associated properties. For example, if you need to enable the **ICMP** protocol and its default settings, you simply open the object and execute **return**, as shown in the session below. Notice that the ICMP object has been added to the configuration hierarchy at the end of the session on the eth4 interface.

CLI Session

```
config> config box interface eth4
config interface eth4> config ip 172.26.2.14
config ip 172.26.2.14> config icmp
config ip 172.26.2.14> return
config interface eth4> return
config box> return
config> show -v
interface eth4
  admin enabled
  mtu 1500
  arp enabled
  speed 1Gb
  duplex full
  autoneg enabled
  ip 172.26.2.14
   admin enabled
   ip-address dhcp
```

```
geolocation 0
metric 1
classification-tag
security-domain
address-scope
filter-intf disabled
icmp
 admin enabled
 limit 10 5
```

Removing Objects From the Configuration File Using the CLI

To remove an object from the configuration hierarchy, use the CLI or CMS **delete** command. For example, the CLI session below deletes the IP interface 172.26.1.14 from the configuration hierarchy:

CLI Session

```
config> config box interface eth4
config interface eth4> delete ip 172.26.1.14
```

Editing and Saving the Configuration File Using the CLI

There are three levels of configuration: the working config which keeps a record of configuration edits, the running configuration which is used by the system, and the startup configuration file from which the system boots.

1. The startup, or default, config is saved to the /cxc/cxc.cfg file. When the ME starts, it loads the startup config into the running config. Use the `save` command, either at the config prompt (`config>`) or at the top–level prompt `Net-Net>)` by default), to save the running config to the startup config.
2. The running config is the current operational configuration. You can display the running config using the following command:

   `Net-Net>` **config show -v**

   Edit the running config using the CLI `config` command, or ME Management application. You can save the running config to a file (either the startup config file or a different file) using the `config save` command.
3. When you edit a configuration object, you get a working copy of that object. The working config maintains a record of all configuration changes you have made since the last save to the running config. However, your changes are not applied to the running config until you explicitly commit them. While you're editing an object, the show command displays your working copy. Use the `commit` command, or **exit** from config mode and answer **yes** to the prompt, to save changes from the working configuration to the running configuration.

For detailed information on using the CLI and other management services that allow you to edit the config file, refer to the *WebRTC Session Controller Media Engine Object Reference*.

Creating SIP Users and Passwords

The **user** configuration object allows you to define the users who can pass SIP traffic on this virtual system partition (VSP). (Refer to the ME Virtual System Partitions for more information about ME VSPs). The users object only applies if your SIP configuration requires local authentication in the **default-session-configuration** object

under VSP, or in the **session-configuration** object under the policy configuration object.

When you enable the local authentication file, you configure the ME to prompt those users that are passing SIP traffic to log in. The user name and password tag they enter must match the entries in this file. However, you can also create policy that, for example, does not attempt to authenticate users listed in the Active Directory.

CLI Session

The following CLI session creates a locally authenticated SIP user.

```
NNOS-E> config vsp
config vsp> config user bob-pc@companySierra.com
Creating 'user bob-pc@companySierra.com'
config user bob-pc@companySierra.com> set admin enabled
config user bob-pc@companySierra.com> set password-tag abcXYZ
```

Unlike ME administrative users, SIP users who log in with a valid user name and password do not have read/write access to the ME configuration file.

Customizing the CLI

The ME software allows you to customize the CLI to accommodate the type of display you are using, as well as change the default ME that is pre-configured with the platform.

CLI Session

The following CLI session sets the number of rows that the CLI displays in a single page to 24 lines, and resets the default top-level prompt from Net-Net> to *boston1*>, and sets an optional text banner to appear when you start the CLI.

```
config box> config cli
config cli> set display paged 24
config cli> set prompt boston1>
config cli> set banner text
```

To temporarily change the CLI display mode with changing the default configuration, use the **display** command at the top level of the CLI.

```
NNOS-E> display paged 24
```

Whenever you use paged output, the --More-- prompt accepts the following keystrokes:

- [Enter]: Displays the next line of text
- [Tab]: Displays the remainder of the text
- [Esc], Q, or q: No more text
- Any keystroke: Displays the next page of text

To change from paged output to continuous scrolled output, enter the following command:

```
config cli> set display scrolled
```

Setting ME Global Properties

You can configure global text properties associated with each ME system in the network. These global text properties include:

- hostname
- name
- description
- contact
- location
- timezone

CLI Session

The following CLI session enables the ME administrative state, and sets the optional text descriptions associated with this ME system.

```
NNOS-E> config box
config box> set admin enabled
config box> set hostname company.boston1.companySierra.com
config box> set name boston1
config box> set description Net-NetMasterBoston
config box> set contact adminFred
config box> set location corpDataCenter
config box> set timezone Pacific
```

ME Virtual System Partitions

ME's virtual system partition (VSP) is the part of the system that holds the comprehensive customer-defined configuration that controls how the system processes, stores, directs, and routes SIP traffic. The VSP is where you can create session configurations, registration and dial plans, and policies that handle SIP REGISTER and SIP INVITE traffic (and other SIP methods) that ME will receive and forward to a SIP call destination, authentication and accounting database, VoIP service provider or carrier, enterprise server, and so on.

The VSP configuration uses objects and properties that control the majority of the ME functionality.

IPMI Support

Intelligent Platform Management Interface (IPMI) is supported on the NN2600 series hardware only. Oracle cannot guarantee it will function properly on any other third-party hardware.

For more information about configuring IPMI on the ME, see the *Oracle Communications OS-E System Operations and Troubleshooting Guide*.

Specifying Management Preferences

The **cms-preferences** object allows you to configure enumeration text strings to network, database, and SIP objects that support extensions, as well as preferences for reverse DNS, trap polling intervals, phone path mapping, and the cluster and box summary information to include on the Status summary page.

CLI Session

The following CLI session configures the `securityDomain` and the `sipHeaderNameEnum` strings, how frequently (in seconds) to check for SNMP traps

```
NNOS-E> config preferences
config preferences> config cms-preferences
config cms-preferences> set enum-strings securityDomain untrusted
config cms-preferences> set enum-strings sipHeaderNameEnum accept-encoding
config cms preferences> set trap-poll-interval 60
```

For more information on configuring the optional enumeration strings, refer to *Net-Net OS-E – Objects and Properties Reference*.

Specifying DOS Query Preferences

Denial of service (DOS) attacks are designed to disable networks by flooding them with useless traffic. The ME provides transport-layer and SIP-layer query and policy capabilities to manage DOS attacks. Queries allow you to sort and view incoming and outgoing traffic in an effort to better define policies. You can use policies to determine if a packet is attacking the box, and configure the responding action. These tools quickly identify and shutout dubious traffic, thereby limiting the damage caused by DOS attacks.

CLI Session

The following CLI session opens the **dos-queries** object and a named sip-query (companySierra), followed by the sip-query options that control how the query displays and sorts DOS traffic:

```
NNOS-E> config preferences
config preferences> config dos-queries
config dos-queries> config sip-query companySierra
Creating 'sip-query companySierra'
config sip-query companySierra> set description "SIP-layer queries"
config sip-query companySierra> set admin enabled
config sip-query companySierra> set select content-type
config sip-query companySierra> set group session-id
config sip-query companySierra> set sort timestamp
config sip-query companySierra> set order ascending
```

For more information on configuring the DOS query preferences, refer to the *Net-Net OS-E – Session Services Configuration Guide* and the *Net-Net OS-E – Objects and Properties Reference*.

Restarting and Shutting Down the System

At times, you may need to shut down or restart the system.

- To shut down the system completely, press the On/Off button on the chassis to OFF.
- To perform a warm or cold restart or a system halt, use the **restart** command. A **restart warm** resets the ME application software; a **restart cold** reboots the platform, **restart halt** suspends ME operation without rebooting or restarting.
- To simultaneously warm restart all systems in the network cluster, use the **restart cluster** command.

Caution: Always save your configuration before you shut down or restart the system. When you restart the ME system, the system uses the latest saved configuration file. If you do not save a configuration prior to a reboot or shutdown, you lose any changes you made since you last saved the configuration file.

CLI Session

The following session performs an ME warm restart:

```
NNOS-E> restart warm
```

Monitoring the ME

This section describes SNMP OIDs to poll and trap, CLI commands, and other features Oracle recommends for monitoring the ME.

SNMP MIB OIDs

SNMP MIB browsers and network management applications can be used to monitor the ME. The SNMP agent allows users to access management information from the MIBs and perform SNMP queries (GETs and GET NEXTs) for information contained in the MIBs.

The SNMP agent supports SNMP V1 and V2c.

Process Restarts

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on system processes.

.iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.processTable.processEntry (.1.3.6.1.4.1.21798.1.1.214.1)

The following table shows the relevant system process OIDs.

Table 20–1 System Process OIDs

OID (text)	OID	Description
processStarts.1(monitor)	.1.3.6.1.4.1.21798.1.1.214.1.6.1	The number of times the monitor process has (re)started.
processStarts.2(manager)	.1.3.6.1.4.1.21798.1.1.214.1.6.2	The number of times the manager process has (re)started.
processStarts.3(sip)	.1.3.6.1.4.1.21798.1.1.214.1.6.3	The number of times the sip process has (re)started
processStarts.4(media)	.1.3.6.1.4.1.21798.1.1.214.1.6.4	The number of times the media process has (re)started
processStarts.5(auth)	.1.3.6.1.4.1.21798.1.1.214.1.6.5	.1.3.6.1.4.1.21798.1.1.214.1.6.5 The number of times the auth process has (re)started
processStarts.6(reg)	.1.3.6.1.4.1.21798.1.1.214.1.6.6	The number of times the reg process has (re)started
processStarts.7(h323)	.1.3.6.1.4.1.21798.1.1.214.1.6.7	The number of times the h323 process has (re)started
processStarts.8(dir)	.1.3.6.1.4.1.21798.1.1.214.1.6.8	The number of times the dir process has (re)started
processStarts.9(web)	.1.3.6.1.4.1.21798.1.1.214.1.6.9	The number of times the web process has (re)started
processStarts.10(ws)	.1.3.6.1.4.1.21798.1.1.214.1.6.10	The number of times the web services process has (re)started
processStarts.11(acct)	.1.3.6.1.4.1.21798.1.1.214.1.6.11	The number of times the acct services process has (re)started

Table 20–1 (Cont.) System Process OIDs

OID (text)	OID	Description
processStarts.12(dos)	.1.3.6.1.4.1.21798.1.1.214.1.6.12	The number of times the dos services process has (re)started
processStarts.17(ssh)	.1.3.6.1.4.1.21798.1.1.214.1.6.17	The number of times the ssh services process has (re)started
processsStarts.20(lcr)	.1.3.6.1.4.1.21798.1.1.214.1.6.20	The number of times the lcr services process has (re)started
processStarts.21(sampling)	.1.3.6.1.4.1.21798.1.1.214.1.6.22	The number of times the sampling services process has (re)started
processStarts.22(presence)	.1.3.6.1.4.1.21798.1.1.214.1.6.22	The number of times the presence services process has (re)started

Active Calls

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on system active calls.

.iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.sipStackTable.sipStackEntry (.1.3.6.1.4.1.21798.1.1.294.1)

The following table shows relevant active call OIDs.

Table 20–2 Active Call OIDs

OID (text)	OID	Description
sipStackActiveCalls	.1.3.6.1.4.1.21798.1.1.294.1.5.10 0.101.102.97.117.108.116	The number of active calls

CPU Usage

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on system CPU usage at various intervals.

.iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.cpuUsage (.1.3.6.1.4.1.21798.1.1.55)

The following table shows relevant CPU usage OIDs.

Table 20–3 CPU Usage OIDs

OID (text)	(OID)	Description
cpuUsageOneSecond.0	.1.3.6.1.4.1.21798.1.1.55.1.0	1 second sample of CPU usage %
cpuUsageTenSecond.0	.1.3.6.1.4.1.21798.1.1.55.2.0	10 second sample of CPU usage %
cpuUsageOneMinute.0	.1.3.6.1.4.1.21798.1.1.55.3.0	1 minute sample of CPU usage %
cpuUsageTenMinute.0	.1.3.6.1.4.1.21798.1.1.55.4.0	10 minute sample of CPU usage %
cpuUsageOneHour.0	.1.3.6.1.4.1.21798.1.1.55.5.0	1 hour sample of CPU usage %

Database Maintenance Status

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on database maintenance.

.iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.databaseMaintenanceStatus (.1.3.6.1.4.1.21798.1.1.58)

The following table shows relevant database maintenance OIDs.

Table 20–4 Database Maintenance OIDs

OID (text)	OID	Description
databaseMaintenanceStatusStatus .0	.1.3.6.1.4.1.21798.1.1.58.1.0	Current database maintenance status
databasemaintenanceStatusResult .0	.1.3.6.1.4.1.21798.1.1.58.5.0	The result of the last database maintenance job

Fault Groups

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on fault groups.

.iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.groupsTable.groupsEntry (.1.3.6.1.4.1.21798.1.1.112.1)

The following table shows relevant fault group OIDs.

Table 20–5 Fault Group OIDs

OID (text)	OID	Description
groupsActive.<#>	.1.3.6.1.4.1.21798.1.1.112.1.5.<#>	Status of group number.

Location Cache

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on location cache.

.iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.locationSummary (.1.3.6.1.4.1.21798.1.1.158) OID (text) OID

The following table shows relevant location cache OIDs.

Table 20–6 Location Cache OIDs

OID (text)	OID	Description
locationSummaryTotalAORs.0	.1.3.6.1.4.1.21798.1.1.158.1.0	The number of Cache entries on the system

Memory Failures

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on memory failures.

.iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.memory (.1.3.6.1.4.1.21798.1.1.182)

The following table shows relevant memory failure OIDs.

Table 20–7 Memory Failure OIDs

OID (text)	OID	Description
memorySystemHeapAllocFailures.0	.1.3.6.1.4.1.21798.1.1.182.19.0	The number of System Heap Allocation Failures
memoryMallocHeapAllocFailures.0	.1.3.6.1.4.1.21798.1.1.182.20.0	The number of Malloc Heap Allocation Failures
memoryOpenSSLAllocFailures.0	.1.3.6.1.4.1.21798.1.1.182.21.0	The number of OpenSSL Heap Allocation Failures
memoryRVHeapAllocFailures.0	.1.3.6.1.4.1.21798.1.1.182.22.0	The number of RV (SIP Stack Library) Heap Allocation Failures
memoryOtherHeapAllocFailures.0	memoryOtherHeapAllocFailures.0.1.3.6.1.4.1.21798.1.1.182.23.0.	The number of other Heap Allocation Failures
memoryPoolAllocFailures.0	.1.3.6.1.4.1.21798.1.1.182.24.0	The number of Pool Allocation Failures

Hardware Faults

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on hardware faults.

iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.sensorInfo (.1.3.6.1.4.1.21798.1.1.263)

The following table shows relevant hardware fault OIDs.

Table 20–8 Hardware Fault OIDs

OID (text)	OID	Description
sensorInfoFaults.0	.1.3.6.1.4.1.21798.1.1.263.3.0	The number of Faults reported by the onboard hardware monitoring module.

SIP Status

Oracle recommends the following list of SNMP OIDs to GET every five minutes from the CXC MIB (cxc.mib) to obtain information on the SIP stack.

.iso.org.dod.internet.private.enterprises.covergence.cxc.cxcStatus.sipStackTable.sipStackEntry (.1.3.6.1.4.1.21798.1.1.294.1)

The following table shows relevant SIP status OIDs.

Table 20–9 SIP Status OIDs

OID (text)	OID	Description
sipStackStatus	.1.3.6.1.4.1.21798.1.1.294.1.3.100.101.102.97.117.108.116	State of the SIP stack
sipStackActiveCalls	.1.3.6.1.4.1.21798.1.1.294.1.5.100.101.102.97.117.108.116	.1.3.6.1.4.1.21798.1.1.294.1.5.100.101.102.97.117.108.116 Active SIP calls
sipStackFailedCalls	.1.3.6.1.4.1.21798.1.1.294.1.12.100.101.102.97.117.108.116	Failed SIP calls

SNMP Traps

The ME can be configured to send out SNMP traps to a configured SNMP trap receiver. This is timely data to alert the user to issues with the system.

Table 20–10 lists the SNMP traps Oracle recommends to investigate further.

Table 20–10 SNMP Traps

OID (text)	OID	Description
cAMissing	.1.3.6.1.4.1.21798.1.4.7	Indicates that a CA file specified in a TLS certificate configuration entry cannot be found
certDecryptError	.1.3.6.1.4.1.21798.1.4.8	Indicates that a certificate file specified in a certificate configuration could not be decrypted, probably due to an incorrect or missing passphrase
certExpired	.1.3.6.1.4.1.21798.1.4.9	Indicates that a certificate file specified in a certificate configuration is no longer valid, as specified in the certificate's 'notAfter' extension
certExpiring	.1.3.6.1.4.1.21798.1.4.10	Indicates that a certificate file specified in a certificate configuration will expire shortly (within the next 7 days), as specified in the certificate's 'notAfter' extension
certFormat	.1.3.6.1.4.1.21798.1.4.11	Indicates that a certificate file specified in a TLS certificate configuration entry is not of a supported format (PEM or PKCS#12)
certMissing	.1.3.6.1.4.1.21798.1.4.12	Indicates that a certificate file specified in a TLS certificate configuration entry cannot be found or opened
certNoPrivateKey	.1.3.6.1.4.1.21798.1.4.13	Indicates that a certificate file specified in a TLS certificate configuration entry does not have a valid private key; this could be due to an incorrect passphrase.
certNotYetValid	.1.3.6.1.4.1.21798.1.4.14	Indicates that a certificate file specified in a certificate configuration is not yet valid, as specified in the certificate's 'notBefore' extension
cRLMissing	.1.3.6.1.4.1.21798.1.4.15	Indicates that a CRL file specified in a TLS certificate configuration entry cannot be found
dosSIPPolicyTrap	.1.3.6.1.4.1.21798.1.4.17	Indicates that a dynamic policy rule is instituted in response to a SIP Policy threshold being crossed
dosTransportPolicyTrap	.1.3.6.1.4.1.21798.1.4.18	Indicates that a dynamic policy rule is instituted in response to a Transport Policy threshold being crossed
dosUrlPolicyTrap	.1.3.6.1.4.1.21798.1.4.19	Indicates that a dynamic policy rule is instituted in response to a URL Policy
headEndUndersubscribed	.1.3.6.1.4.1.21798.1.4.22	A head-end interface is undersubscribed, and therefore SIP messages are being dropped
lBConfiguredAsBoth	.1.3.6.1.4.1.21798.1.4.24	An interface has been configured as both a head-end and a backing, and therefore SIP load-balancing will not function

Table 20–10 (Cont.) SNMP Traps

OID (text)	OID	Description
licenseExpiring	.1.3.6.1.4.1.21798.1.4.25	Report the imminent expiration of a license
licenseExpiring	.1.3.6.1.4.1.21798.1.4.25	Report the imminent expiration of a license
mediaSessionDroppedPackets	.1.3.6.1.4.1.21798.1.4.26	Indicates the dropped media packets for a session exceeded the threshold specified
mediaVerificationFail	.1.3.6.1.4.1.21798.1.4.27	Indicates a media stream within a call exceeds the expected parameters
monitorAlert	.1.3.6.1.4.1.21798.1.4.28	Report that a monitor parameter has crossed the configured threshold
processDown	.1.3.6.1.4.1.21798.1.4.30	Report that a process has gone down
raidEventTrap	1.3.6.1.4.1.21798.1.4.31	Indicates the RAID controller has generated an event
sensorEvents	1.3.6.1.4.1.21798.1.4.33	Report that a sensor event has occurred
sipParseErrorsTrap	.1.3.6.1.4.1.21798.1.4.35	The number of parse errors in received SIP messages has exceeded the configured threshold.
sipServerEvent	.1.3.6.1.4.1.21798.1.4.36	Report on state of SIP server
storageDeviceFull	.1.3.6.1.4.1.21798.1.4.37	The CXC attempted to record media but the free space is less than the configured threshold
synCookiesTrap	.1.3.6.1.4.1.21798.1.4.38	An increase in the TcpSynCookiesSent counter indicates a possible TCP SYN flood attack
systemHalt	.1.3.6.1.4.1.21798.1.4.39	Report that a system halt has been initiated
masterServiceChange	.1.3.6.1.4.1.21798.1.4.53	Report a master service state change
masterServiceHostChange	.1.3.6.1.4.1.21798.1.4.54	Report a master service host box state change

CLI Commands

The following list of show status commands can be used to provide information on overall system performance.

- show processes
- show active-call-summary
- show cpu-usage
- show database-maintenance-status
- show groups
- show location-cache
- show memory failures
- show sensor-info
- show sensor-events
- show login-sessions
- show sip-stack

- show faults
- show interfaces
- show master-services
- show vrrp-hosts
- show media-ports-summary
- show mounts

The following list of show status commands can be used to provide information on general web services.

- show dynamic-event-services
- show web-services-callout-detail
- show web-services-callout-status
- show web-services-client-status
- show web-services-fault-status
- show web-services-ports
- show web-services-request-status
- show web-services-status

The following list of show status commands can be used to provide information if you have a virtual host application running on the ME.

- show web-services-virtual-host-application-parameters
- show web-services-virtual-host-application-servlet-parameters
- show web-services-virtual-host-application-servlets
- show web-services-virtual-host-applications
- show web-services-virtual-hosts

For more information on these show commands, see the *WebRTC Session Controller Media Engine Object Reference*.

Other Monitoring Tools

This section describes several other tools you can use to monitor the ME.

Syslog

By configuring the ME to send out system messages to a configured Syslog server, you can obtain data useful for historical logging and detailed troubleshooting.

> **Note:** Enable only filters that specify events to monitor to avoid alarming on many irrelevant events.

CMS Web

You can monitor various system data on the ME via the CMS Web graphical interface using a standard HTTP secure browser.

Web Services Description Languages (WSDL) API

The WSDL/SOAP (Simple Object Access Protocol) management interface on the ME allows you to monitor status, execute actions, and read and write the configuration. It also provides special-purpose functionality to support integration of location information, event, and policy services with external services.

Accounting CDRs

You can configure the ME to create and send Accounting Call Detail Records (CDR). CDRs can be written to .csv files, RADIUS servers, and external databases (i.e. MySQL, Postgres, Microsoft SQL, etc.). This data can be farmed for monitoring purposes as well as traditional billing uses. For example, determining call completion rates at various high and low points during the day.

21

Configuring Permissions, Users, and Authorization

This chapter describes configuring and managing permissions, users, and authorization under the ME's Access tab.

Configuring Permissions

Under the Access tab you can configure permissions. From this object you can enable or disable access to a variety of ME services. Once a permission set is created, it can be applied to configured users.

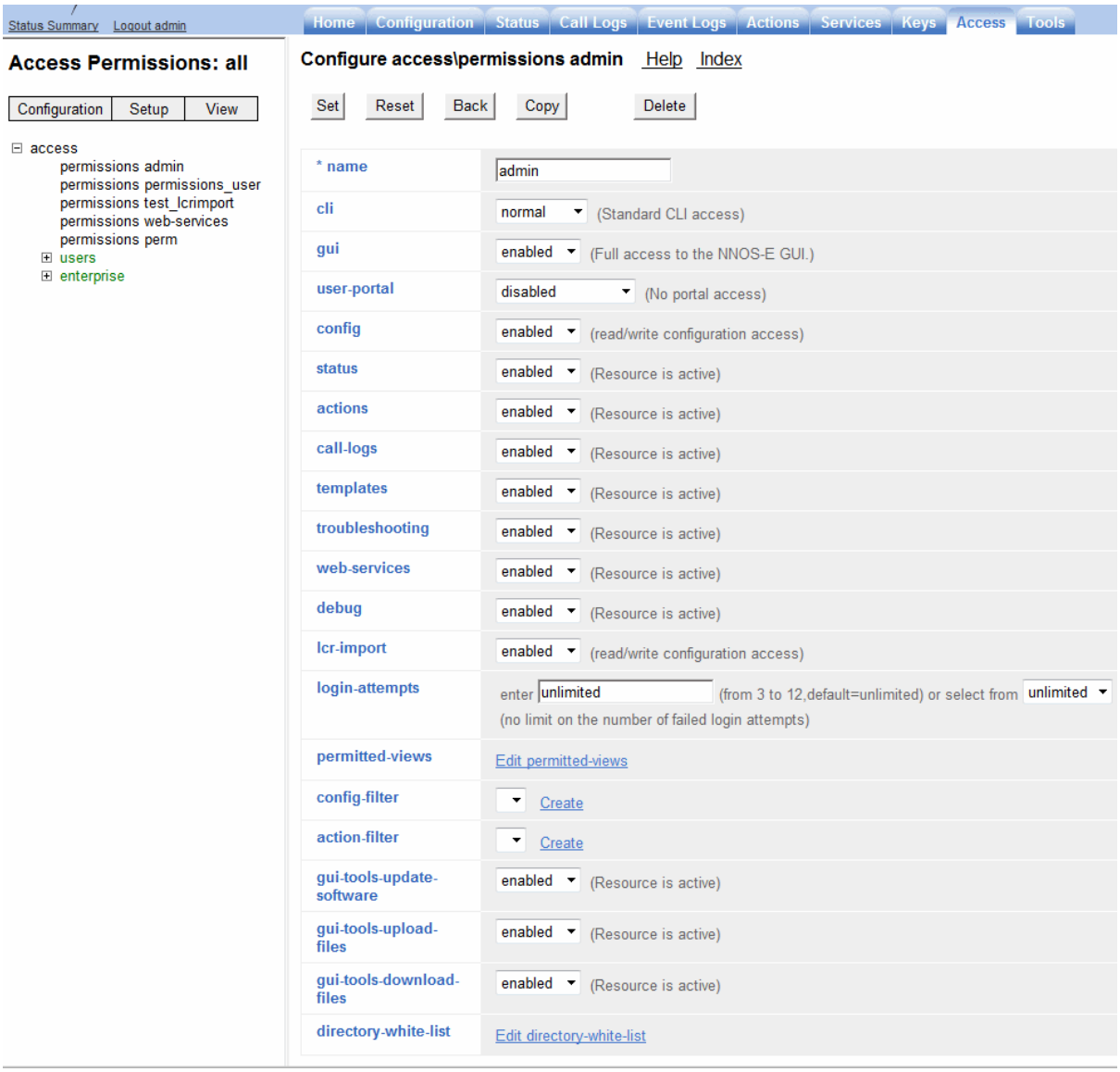
To create a permission set:

1. Select the **Access** tab and click **Access**.
2. Click **Add permissions**.
3. Enter the **name** you want to give this permission set and select **Create**.

The **permissions** object appears. For more information on the **permissions** object and properties, see the *WebRTC Session Controller Media Engine Object Reference*.

Figure 21–1 shows a permission set named **admin**.

Figure 21–1



Note: To edit an existing permission set, click **Edit** beside that permission and the **permissions** object. To delete a permission set click **Delete**.

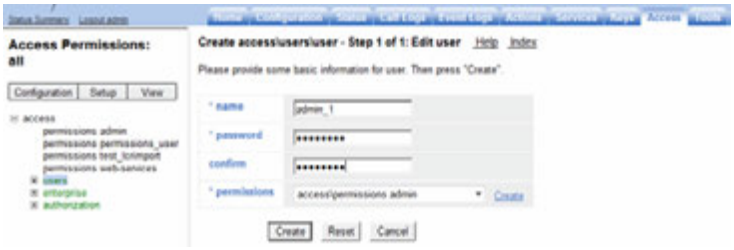
Configuring Users

Configure ME users using the Access tab's **users** object.

When creating a user, you assign them a name, a password, and apply to them a configured permissions set.

To create a user:

1. Select the **Access** tab and click **users**.
2. **admin**: Set to enabled to allow configured users access to the ME.
3. Click **Add user**. The user object appears.



4. **name**: Enter a name to give this user.
5. **password**: Enter a password for this user.

> **Note:** Via the **password-policy** object, you can specify password requirements for configured users. For more information on the **password-policy** object, see the *WebRTC Session Controller Media Engine Object Reference*.

6. **confirm**: Reenter the password.
7. **permissions**: Select a pre-configured permissions set to apply to this user from the drop-down list. If you have not configured permissions yet, click **Create**.
8. Click **Create**.
9. Click **Set**. Update and save the configuration.

Configuring Action and Config Filters

The ME supports filtering mechanisms which control which users have access to specific actions and configuration objects and properties. These filters are configured under the **access > permissions** object.

The three permission filters are:

- Config-filter
- Action-filter-blacklist
- Action-filter-whitelist

There are three steps necessary to assign action and configuration filters to configured users. You must create the filters, assign filters to permissions set, then assign each user a permission set.

Configuring Config-Filters

Via the **config-filter** property, you can select a config-filter containing a list of configuration objects and properties you want to restrict certain users from being able to access.

Config-filters have three permission levels.

- read-write: Users can modify the configuration
- read-only: Users can view the configuration but cannot modify it
- none: Users can neither view nor modify the configuration

By default, child objects and properties inherit permissions from their parent classes, however, a user may apply a lesser permission to a child object or property. The following table lists the inheritance of permissions for the configuration.

Table 21–1 Configuration Permissions Inheritance

Inherited Permission	Child Object/Property Permission	Effective Permission of Child Object/Property Permission
read-write	read-write	read-write
read-write	read-only	read-only
read-write	none	none
read-only	read-only	read-only
read-only	none	none
none	none	none

To configure a **config-filter**:

1. Select the **Access** tab and click **Access**.
2. Click **Configure** next to **permission-filters**.
3. Click **Add config-filter**.
4. **name**: Specify a name to give this config-filter.
5. Click **Create**. The **filter** object appears.
6. **admin**: Set to **enabled** to enable this config-filter.
7. Click **Add filter**.
8. **filter**: Specify a configuration object by entering the class, object, and property in free form, separating each with a back slash "\".
9. Click **Create**.
10. Repeat Steps 7 and 8 for as many configuration objects you want to apply to this filter.
11. Click **Set**. Update and save the configuration.

To specify a filter permission:

1. Click **Edit** next to the filter.
2. **permission**: Select the permission level for this filter from the drop-down list. This is set to **none** by default.
3. Repeat this for each filter.
4. Click **Set**. Update and save the configuration.

Configuring Action-Filters

Via the **action-filter-blacklist** property, you can select an action-filter containing a list of actions you want to restrict certain users from using. When a user attempts to execute a restricted action, he gets the following error message:

```
Insufficient permissions for user
```

Via the **action-filter-whitelist** property, you can select an action-filter containing a list of actions you want to allow certain users to use.

The action-filter-whitelist property supports the use of a wildcard. The wildcard is an asterisk (*) that can be located at the end of a string only. For example, to create an action-filter for all call-control actions, enter **call-control-***.

When action-filters are configured on the ME, the ME always checks the **action-filter-blacklist** settings first. If the action is found on the blacklist, the user is not allowed to use it.

If both the **action-filter-blacklist** and **action-filter-whitelist** are configured and an action does not appear on either list, the user is restricted from using the action.

If an action is not found on the **action-filter-blacklist** and **action-filter-whitelist** is not configured, the user is allowed to use it.

> **Note:** You must enter actions into the **action-filter-blacklist** and **action-filter-whitelist** properties without any arguments. When anything more than an action name is specified, the ME ignores the filter.

To configure an **action-filter**:

1. Select the **Access** tab and click **Access**.
2. Click **Configure** next to **permission-filters**.
3. Click **Add action-filter**.
4. **name**: Specify a name to give this **action-filter**.
5. Click **Create**. The **filter** object appears.
6. **admin**: Set to **enabled** to enable this action-filter.
7. Click **Add filter**.
8. **filter**: Specify an action, without any arguments, to be applied to this filter.

> **Note:** If you enter an action with arguments, the action is ignored.

9. Repeat Steps 7 and 8 for as many actions you want to apply to this filter.
10. Click **Set**. Update and save the configuration.

Applying Filters to Permissions Sets

Once you have created config-filters and action-filters, you must apply them to a permission set.

To apply config-filters and action-filters to a permissions set:

1. Select the **Access** tab and click **Access**.
2. Click **Add permissions** to create a new permissions set or click **Edit** next to an existing permissions set.
3. **config-filter**: Select a config-filter from the drop-down list whose configuration objects you want to restrict users with this permissions set from using. If you have not yet created a config-filter, click Create next to this property.

4. **action-filter-blacklist**: Select an action-filter from the drop-down list whose actions you want to restrict users with this permissions set from using. If you have not yet created an action-filter, click Create next to this property.
5. **action-filter-whitelist**: Select an action-filter from the drop-down list whose actions you want to allow users with this permissions set to use. If you have not yet created an action-filter, click Create next to this property.
6. Click **Set**. Update and save the configuration.

Once you have configured config-filters and action-filters and applied them to a permissions set, you can assign the permissions set to users. For more information on applying permissions set to users, see Configuring Users.

Configuring Authorization

Once you have configured permission sets and users, you can further define user access by configuring authorization. Authorization consists of creating specific grants, or privileges.

There are three types of grants you can create:

- default-grants: Applies to all configured ME users
- attribute-grants: Applies to configured ME users based on values extracted from their attributes.
- group-grants: Applies to configured ME users based on group membership

The grants you can create apply to just a small segment of actions, which are divided into groups called resource-types. A resource-type is the ME function on which you are setting permissions.

The following table lists the resource types along with their corresponding actions.

Table 21–2 Resource Types

Resource-Type	Associated Actions	CRUD Privileges
call	call-control-accept	N/A
N/A	call-control-annotate	N/A
N/A	call-control-attach	CU
N/A	call-control-call	C
N/A	call-control-call-to-session	CU
N/A	call-control-connect	N/A
N/A	call-control-create-session	C
N/A	call-control-destroy-session	D
N/A	call-control-detach	D
N/A	call-control-disconnect	D
N/A	call-control-fork	U
N/A	call-control-get-annotation	U
N/A	call-control-hold	U
N/A	call-control-info-request	U
N/A	call-control-intercept	U

Table 21–2 (Cont.) Resource Types

Resource-Type	Associated Actions	CRUD Privileges
N/A	call-control-join	U
N/A	call-control-message-request	U
N/A	call-control-modify	U
N/A	call-control-mute-off	U
N/A	call-control-mute-on	U
N/A	call-control-notify	U
N/A	call-control-notify-request	U
N/A	call-control-options-request	U
N/A	call-control-park	CU
N/A	call-control-park-to-session	CU
N/A	call-control-persistence	U
N/A	call-control-record-stop	C
N/A	call-control-redirect	U
N/A	call-control-reject	U
N/A	call-control-retrieve	U
N/A	call-control-ringing	U
N/A	call-control-send-message	U
N/A	call-control-subscribe-request	U
N/A	call-control-terminate	D
N/A	call-control-transfer	U
call-recording	call-control-record-start	C
N/A	call-control-record-stop	C
call-monitor	call-control-monitor-file	CU
N/A	call-control-monitor-session	CU
call-media-insertion	call-control-drop-file	CU
N/A	call-control-insert-dtmf	U
N/A	call-control-media-pause	CU
N/A	call-control-media-resume	CU
N/A	call-control-media-scanner-start	CU
N/A	call-control-media-scanner-stop	CU
N/A	call-control-media-seek	CU
N/A	call-control-media-stop	CU
N/A	call-control-memo-begin	CU
N/A	call-control-memo-end	CU
N/A	call-control-play	U
sip-request	sip-send-message	CU
N/A	sip-send-notify	CU

Table 21–2 (Cont.) Resource Types

Resource-Type	Associated Actions	CRUD Privileges
N/A	sip-send-options	CU
N/A	sip-send-other	CU
N/A	sip-send-subscribe	CU
N/A	sip-send-unsubscribe	CU
registration	register	C
N/A	unregister	D
event-channel	dynamic-event-service register	CR
N/A	dynamic-event-service keepalive	U
N/A	dynamic-event-service unregister	D

In cases where an action has required either <*handle*> or <*session ID*> arguments, the ME extracts the To and From URI identities from each call leg, matches them against the resource-identity specified in a user's privileges, and determines whether that user is authorized to perform an operation.

When configuring a grant, you must define privileges for that resource-type. Privileges specify what a user can or cannot do with that resource-type.

Privileges on the ME follow the standard CRUD model:

- create
- retrieve
- update
- delete

Configuring Default Grants

Configure grants under the Access tab's **authorization** object.

Default grants are one of three types of grants you can configure on the ME. Default grants are grants that apply to all ME users matching the specified resource identity.

To configure default grants:

1. Select the **Access** tab and click **authorization**.
2. Set **admin** to **enabled** to enable authorization.
3. Click **Add default-grant**. The **default-grant** object appears.
4. **name**: Enter a name to give this grant.
5. **resource-identity**: Select the type of matching to use to identify a resource-type. The following are valid values:
   - equals <*value*>: The value that a user provides during an authorization request must be exactly the same as the resulting resource-identity. This is the default setting.
   - matches <*expression*>: The value that a user provides during an authorization request is matched against the resource-identity using a regular expression match.

> **Note:** For more information on using Regular Expressions, see the *WebRTC Session Controller Media Engine Object Reference*.

- any: Any value a user provides during an authorization request matches.

6. **resource-type**: Select the resource-type for this grant from the drop-down list.
7. **privileges**: Check the CRUD privileges to allow for this resource-type. By default, they are all selected.
8. Click **Create**.
9. Click **Set**. Update and save the configuration.

Configuring Attribute Grants

Attribute grants are grants that apply to all ME users that have the attribute and match the specified resource-identity.

To configure attribute-grants:

1. Select the **Access** tab and click **authorization**.
2. name: Enter the name of the attribute for which you are creating this grant.

> **Note:** The name you provide must be the name of an actual attribute used within the directory.

3. Click **Create**. The **attribute-grant** object appears.
4. Click **Add grant-pattern**.
5. **name**: Enter a descriptive name to give this grant.
6. **pattern**: Enter the regular expression pattern to use to define the attribute.
7. **resource-identity**: Select the type of matching to use to identify a resource-type. The following are valid values:
   - equals <*value*>: The value that a user provides during an authorization request must be exactly the same as the resulting resource-identity. This is the default setting.
   - matches <*expression*>: The value that a user provides during an authorization request is matched against the resource-identity using a regular expression match.

> **Note:** For more information on using Regular Expressions, see the *WebRTC Session Controller Media Engine Object Reference*.

   - any: Any value a user provides during an authorization request matches.
8. **resource-type**: Select the resource-type that this extracted value represents from the drop-down list.
9. **privileges**: Check the CRUD privileges to allow for this resource-type. By default, they are all selected.
10. Click **Create**.

11. Click **Set**. Update and save the configuration.

Configuring Group Grants

Under the **group-grant** object, you can configure default and attribute grants for specific groups. Group grants apply to users belonging to these groups and matching the resource-identity.

To add a group-grant:

1. Select the **Access** tab and click **authorization**.
2. Click **Add group-grant**.
3. **name**: Enter the name of the group for which you are configuring this grant.
4. Click **Create**. The **group-grant** object appears.
5. Click **Add default-grant** to configure a default grant for this group or click **Add attribute-grant** to configure an attribute grant for this group.
6. Configure the default or attribute grant as described above.

> **Note:** For more information on configuring **default-grants** see Configuring Default Grants. For more information on configuring **attribute-grants** see Configuring Attribute grants.

7. Click **Set**. Update and save the configuration.

Viewing User Privilege Information

There are three show commands which allow you to view information on your grant configuration: **show authorized-user-privileges**, **show authorized-user-attributes**, and **show authorized-user-groups**.

The **show authorized-user-privileges** action displays information about users' authorization privileges from the user cache.

> **Note:** If a user has never logged into the ME, their name does not appear in the cache and, therefore, is not displayed in the **show authorized-user-privileges** command output.

```
NNOS-E>show authorized-user-privileges

username   resource-type privilege identity-type resource-identity
--------   ------------- --------- ------------- -----------------
admin      event-channel C+R+U+D   equals        /system/*
```

The following table lists and describes the properties associated with the **show authorized-user-privileges** show command.

Table 21–3 Show Authorized-User-Privileges Properties

Field	Description
username	The name of the configured ME user.
resource-type	The resource-type of the grant configured for this user.

Table 21–3 (Cont.) Show Authorized-User-Privileges Properties

Field	Description
privilege	The CRUD privileges of the of the resource-type configured for this user.
identity-type	The method in which the ME matches the users' resource-identity.
resource-identity	The value or regular expression the ME uses to check users' authorization privileges.

The **show authorized-user-attributes** action displays information about configured ME users and their attributes and values.

```
NNOS-E>show authorized-user-attributes

username   attribute                    value
--------   ---------                    -----
sjones     mail                         sjones@acmepacket.com
sjones     msrtcsip-primaryuseraddress  sip:sjones@acmepacket.com
sjones     cn                           Sam Jones
sjones     samaccountname               sjones
sjones     msrtcsip-line                tel:+17815557256
sjones     st                           MA
sjones     telephonenumber              +1 (781) 555-4839
```

The following table lists and describes the properties associated with the **show authorized-user-attributes** show command.

Table 21–4 Show Authorized-User-Attributes Properties

Field	Description
username	The configured ME user.
attribute	The attribute name.
value	The value of the attribute for that user.

The **show authorized-user-groups** action displays the configured users and the groups to which they belong from the user cache.

```
NNOS-E>show authorized-user-groups

username     group
--------     -----
sjones       eng
sjones       software
sjones       dev
sjones       ct
sjones       engineering
sjones       deliveries
sjones       funcspec
```

The following table lists and describes the properties associated with the **show authorized-user-group** show command.

Table 21–5 Show Authorized-User-Group Properties

Field	Description
username	The configured ME user.
group	The group to which the user belongs.

The **show authorized-user-summary** action displays an abbreviated version of users' authorization privileges from the user cache.

```
NNOS-E>show authorized-user-summary

username    resource-types
--------    --------------
admin       event-channel
test_user   event-channel
```

The following table lists and describes the properties associated with the **show authorized-user-summary** show command.

Table 21–6 Show Authorized-User-Summary Properties

Field	Description
username	The name of the configured ME user.
resource-type	The resource-type of the grant configured for this user.

22

Enabling ME Interfaces and Protocols

This chapter describes network interfaces and the protocols that you can enable on ME systems.

ME Sample Networks

Figure 22–1 illustrates a sample enterprise network with a single ME system.

Figure 22–1 A Sample Enterprise Network with a Single ME System

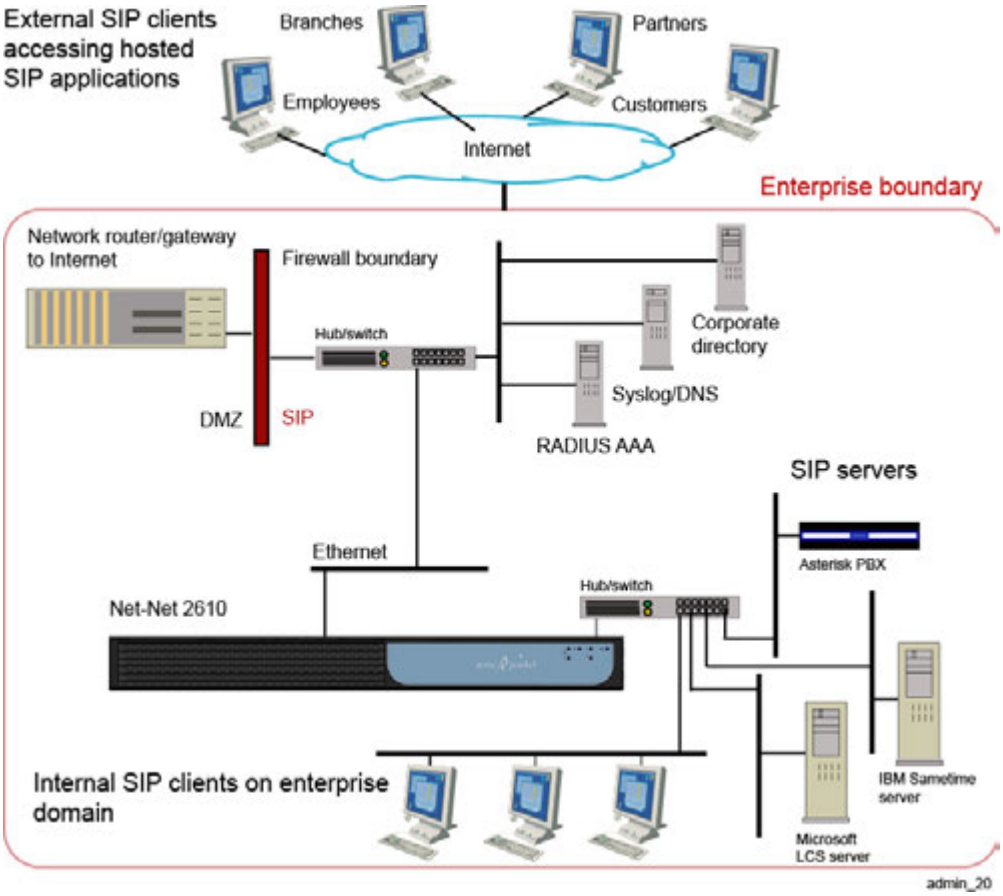
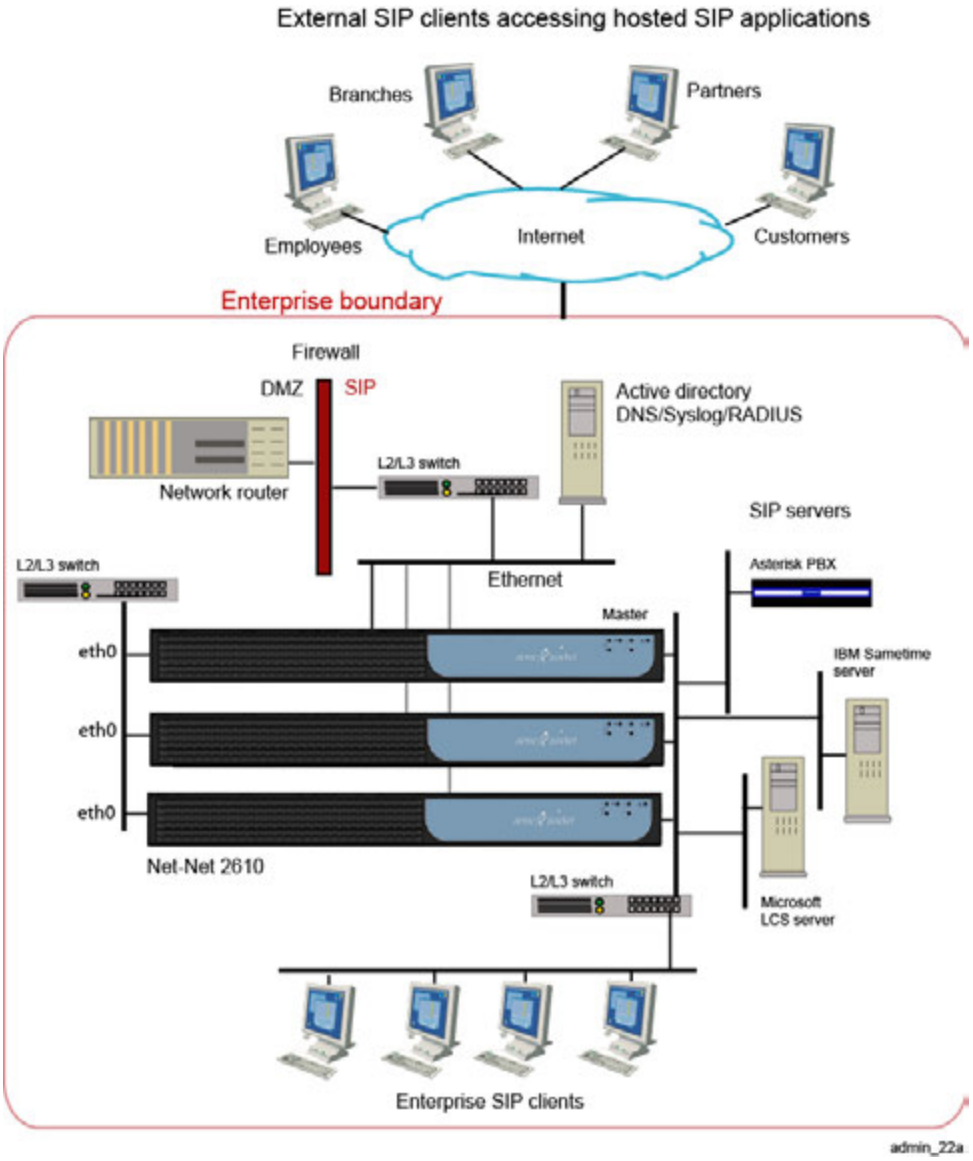


Figure 22–2 illustrates a sample enterprise that uses an ME cluster.

Figure 22–2 A Sample Enterprise that Uses an ME Cluster

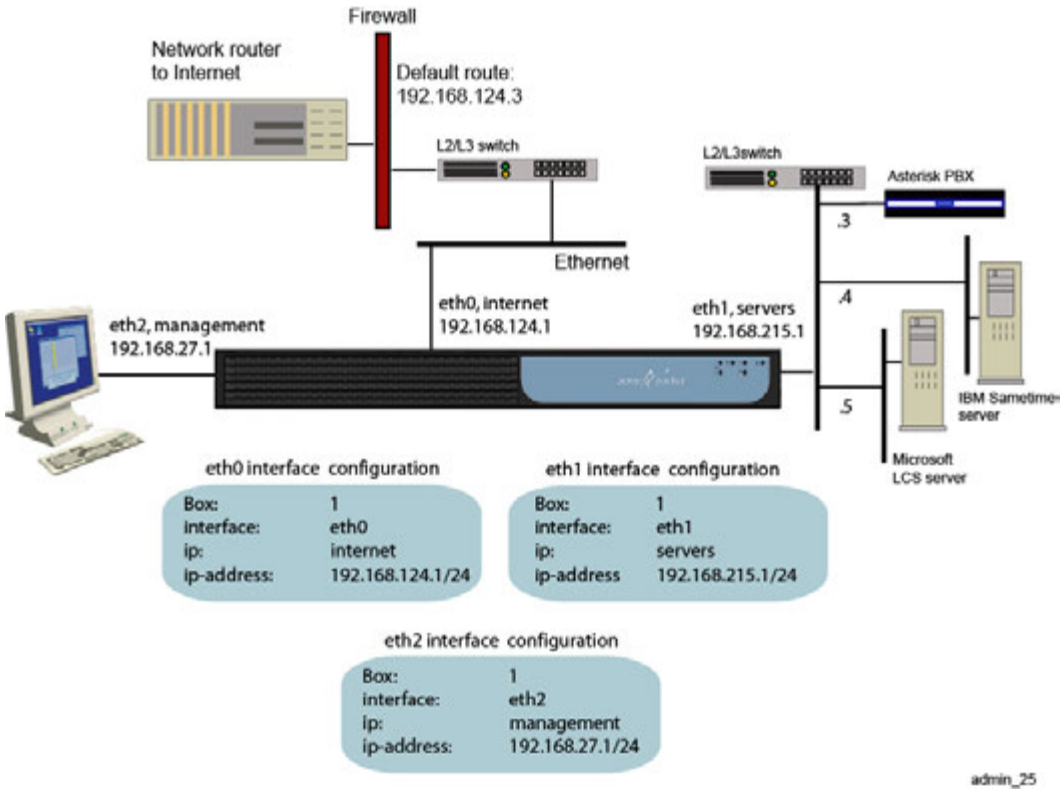


Configuring ME IP Interfaces

ME physical interfaces include multiple Ethernet 1000 Mbps auto-negotiation interfaces, such as eth0, eth1, eth2, and eth3. The number of interfaces depends on the specific platform you are using.

ME software uses IP objects, which are assigned a name by the system administrator, to uniquely identify IP connections. Each physical Ethernet interface can contain up to 255 uniquely named IP objects. Figure 22–3 illustrates a sample network with one named IP object on each physical Ethernet interface.

Figure 22–3 Sample Network with One Named IP Object on Each Ethernet Interface



CLI Session for Eth0

The network on physical interface eth0 uses the IP object that the system administrator named *internet*. The *internet* object specifies the IP address that connects to the external Internet local gateway using a default route.

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth0
config interface eth0>config ip internet
Creating 'ip internet'
config ip internet>set ip-address static 192.168.124.1/24
config ip internet>return

config interface eth0>config ip internet
config ip internet>set ip-address static 192.168.124.2/24
config ip internet>config routing
config routing>config route internetGateway
config route internetGateway>set destination default
config route internetGateway>set gateway 192.168.124.3
```

CLI Session for Eth1

The network on physical interface eth1 uses the IP object named *servers*. The static IP address points to the SIP destination servers on the same network subnet, connected over Ethernet switch.

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth1
```

```
config interface eth1>config ip servers
config ip servers>set ip-address static 192.168.215.1/24
config ip servers>return
```

CLI Session for Eth2

The network on physical interface eth2 uses the defined IP object named *management*. The management object specifies the IP address over which management traffic is carried, such as remote CLI session over Telnet, or a ME Management System session.

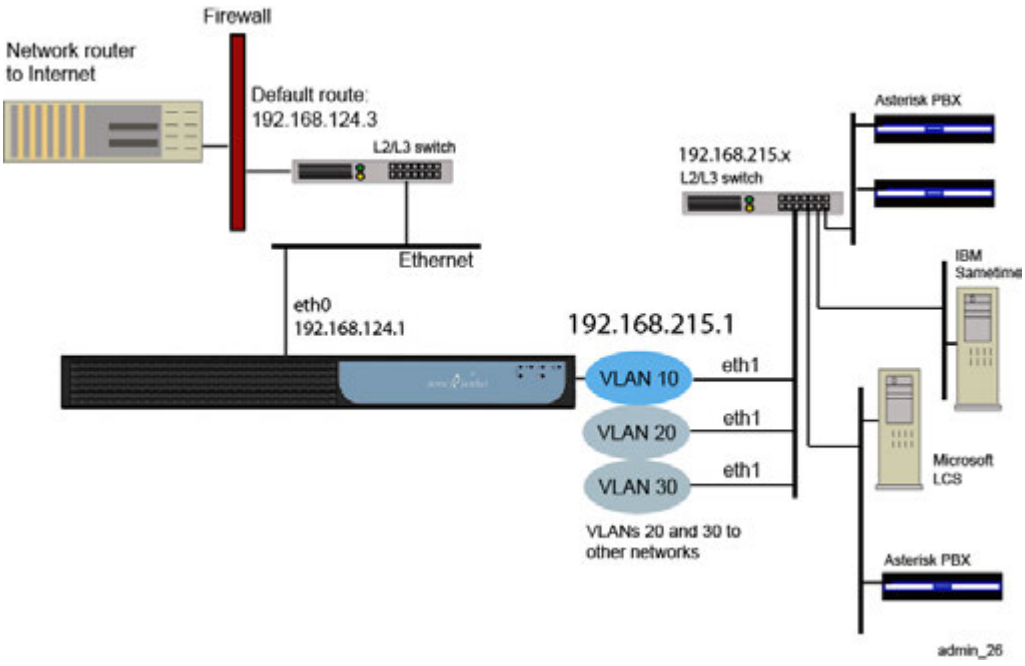
```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth2
config interface eth2>config ip management
config ip internet>set ip-address static 192.168.27.1/24
```

Creating VLANs

ME virtual LANs (VLANs) provide Layer 2 partitions to the communications servers. Creating one or more VLANs allows you to group LAN segments so that they appear to be on the same Layer 2 network. Each VLAN is identified by a VLAN ID, and ID must be unique within the physical ME system. This means that multiple logical ME systems (called VSPs) cannot use the same VLAN IDs. VLAN IDs can be in the range 1 to 4096.

Figure 22–4 illustrates a sample VLAN configuration.

Figure 22–4 A Sample VLAN Configuration



CLI Session

The following CLI session configures the VLAN 10 network. VLAN 10 supports three separate physical IP networks, and all appearing as if they are on the same Layer2 network.

```
SIP>config cluster
```

```
config cluster>config box 1
config box 1>config interface eth1
config interface eth1>config vlan 10
Creating 'vlan10'
config vlan 10>config ip servers
Creating 'ip servers'
config ip servers>set ip-address static 192.168.215.1/24
config ip servers>return
```

Configuring Media Engine Static Routes

On the ME IP interface, you can configure the following three static route types:

- host route: The route to a specific host.
- network route: The route to a specific network.
- default route: The default route to use when there is not a more specific route table match.

In the following example, the ME has an IP interface called "core" configured on network "10.33.4.22/24." This network has a gateway with IP "10.33.4.1".

The example shows a default route, network route to reach "10.44.1.0/24", and a host route to reach "10.55.1.122."

```
config ip core
set ip-address static 10.33.4.22/24
config routing
config route default
set destination default
set gateway 10.33.4.1
return
config route network
set destination network 10.44.1.0/24
set gateway 10.33.4.1
return
config route host
set destination host 10.55.1.122
set gateway 10.33.4.1
return
return
return
```

Applying Routing and Classification Tags

The system uses classification tags to classify incoming traffic and routing tags to control the egress route for a specific service type. Tags allow the IP routing table in Session Manger to be segmented into multiple routing tables. Once an interface has a configured routing tag, the interface is removed from the “null” (or system routing table).

You can create multiple routing tags on the same named IP interface. However, only one classification tag is allowed per IP interface. Both routing and classification tags are case sensitive with the following configuration properties:

- **routing-tag**: Associates all the routes configured on an interface with this **routing-tag** and creates a service route table based on the routing-tag for each service enabled on this interface. The **routing-tag** applies to the *egress* interface over which the ME forwards service traffic. Once a **routing-tag** is configured for

an interface, the service routes associated with that interface are installed in the service route table associated with the routing-tag(s).

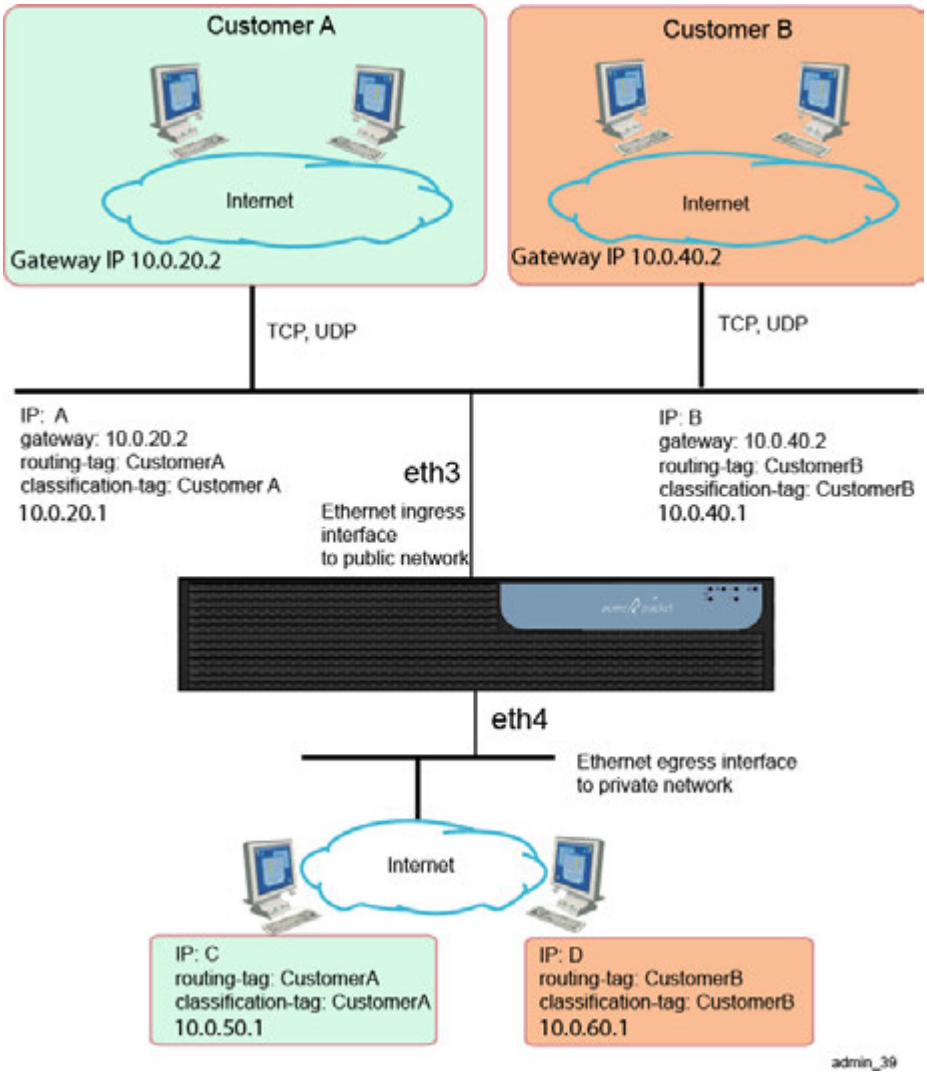
If you create an additional **routing-tag** for the interface with the name "null," the system installs the route in both the default service route table and the tag-specific service route table.

- **classification-tag**: Creates a tag associated with inbound traffic on this interface. This means that you must configure a **classification-tag** on the *ingress* interface over which the ME domain initially receives the traffic, matching the **routing-tag**. (Classification tags in the session configuration **routing-settings** object also must match this routing tag set in the **ip** object.

> **Note:** You can also configure ingress or egress classification tags through the session-config routing-settings object. If this property is configured in both places, the routing-settings configuration takes precedence.

Figure 22–5 illustrates a sample network where routing and classification tags are configured on the ingress and egress ME interfaces, followed by sample configuration sessions for ingress and egress IP instances.

Figure 22–5 Tags Configured on Ingress and Egress ME Interfaces with Sample Sessions



Note: Routing and classification tags are not required for basic ME functionality.

CLI Sessions for "IP A" and "IP B" Ingress Networks on Eth3

The following CLI sessions create the ingress side of the network illustrated in the image above, including the IP addresses, routing and classification tags, SIP settings, and a route to the IP using the gateways at IP addresses at 10.0.20.2 and 10.0.40.2. The ME uses classification tags to classify incoming traffic and routing tags to control the egress route. Configure a **classification-tag** on the incoming interface that matches the **routing-tag** on the egress interface.

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth3
Creating 'interface eth3'
config interface eth3>config ip A
Creating 'ip A'
```

```
config ip A>set ip-address static 10.0.20.1/24
config ip A>set classification-tag CustomerA
config ip A>set routing-tag CustomerA
config ip A>config sip
config sip>set admin enabled
config sip>set nat-translation enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>return
config ip A>config icmp
config icmp>return
config ip A>config routing
config routing>config route default
Creating 'route default'
config route default>set gateway 10.0.20.2
config route default>return
config routing>return
config ip A>return

SIP>config cluster
config cluster>config box 1
config box 1>config interface eth3
Creating 'interface eth3'
config interface eth3>config ip B
Creating 'ip B'
config ip B>set ip-address static 10.0.40.1/24
config ip B>set classification-tag CustomerB
config ip B>set routing-tag CustomerB
config ip B>config sip
config sip>set admin enabled
config sip>set nat-translation enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>return
config ip B>config icmp
config icmp>return
config ip B>config routing
config routing>config route default
Creating 'route default'
config route default>set gateway 10.0.40.2
config route default>return
config routing>return
config ip B>return
```

CLI Sessions for "IP C" and "IP D" Egress Networks on Eth4

The following CLI sessions create the egress side of the network illustrated in the image above, including the IP addresses, routing and classification tags, SIP settings, and a default route. The ME uses classification tags to classify incoming traffic and routing tags to control the egress route. Configure a **classification-tag** on the incoming interface that matches the **routing-tag** on the egress interface.

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth4
Creating 'interface eth4'
config interface eth4>config ip C
Creating 'ip C'
config ip C>set ip-address static 10.0.50.1/24
config ip C>set classification-tag CustomerA
config ip C>set routing-tag CustomerA
```

```
config ip C>config sip
config sip>set admin enabled
config sip>set nat-translation enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>return
config ip C>config icmp
config icmp>return
config ip C>config routing
config routing>config route default
Creating 'route default'
config route default>set destination default
config route default>return

SIP>config cluster
config cluster>config box 1
config box 1>config interface eth4
Creating 'interface eth4'
config interface eth4>config ip D
Creating 'ip D'
config ip D>set ip-address static 10.0.60.1/24
config ip D>set classification-tag CustomerB
config ip D>set routing-tag CustomerB
config ip D>config sip
config sip>set admin enabled
config sip>set nat-translation enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>return
config ip D>config icmp
config icmp>return
config ip D>config routing
config routing>config route default
Creating 'route default'
config route default>set destination default
config route default>return
```

Notes on Routing and Classification Tags

- Separate routing tables are maintained for the SIP and media service.
- IP interfaces without SIP ports enabled will not appear in the SIP table.
- IP interfaces without media ports enabled will not appear in the media table.
- SIP or media traffic that is classified by a tag will only use the routing information and interfaces that have been configured with that routing tag.
- An address of record (AOR) will be assigned an ingress tag IF the REGISTER for that AOR ingresses on an IP interface with a configured **classification-tag**.
- Matches a policy or registration-plan that applies a session configuration that has the **ingress-classification-tag** property configured. This overwrites the IP interface **classification-tag**, if configured.
- Matches a calling-group. The **classification-tag** for the calling-group is only applied if a tag has not been assigned using the IP or session configuration.
- Traffic can be assigned an egress tag as follows:
  - From an ingress tag.

- From a matching policy or dial-plan that applies a session configuration that has the **egress-classification-tag** configured. This overwrites the **classification-tag** configured on the interface.
- From a server or carrier with the routing-tag configured, overwriting all other tags.

Related Commands

To assist troubleshooting, use the following commands from the ME prompt to display information about tag-routing.

- **show services-routing**: Displays routing tables for all tags.
- **show services-routing-tables**: Displays all configured tags.
- **service-route-lookup**: To view the destination where the ME routed a call**.**

Configuring Overlapping IP Networks and Tag Routing

A preferred method for creating networks, with overlapping IPs is to configure VLANs with routing tags. A routing tag associates all the routes configured on an interface and creates a service route table based on the tag for each service enabled the interface. Routing tags apply to the egress interface over which the ME forwards service traffic.

To perform tag routing, do the following:

1. Configure a **classification-tag** on the ingress interface over which the ME initially receives service traffic. The classification tag must match the configured **routing-tag**; each IP interface can have multiple routing tags.
2. Set the **egress-classification-tag** property under the **session-config/routing-settings** when sending service traffic to servers and carriers.

> **Note:** Overlapping IP networks and tag routing are not required for basic ME functionality.

CLI Session for Ethernet Public and Private Sides of Network

The following CLI session configures the ME *public* IP Ethernet interface and SIP settings.

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth3
Creating 'interface eth3'
config interface eth3>config ip public
Creating 'ip public'
config ip public>set ip-address static 10.0.10.1/24
config ip public>config sip
config sip>set admin enabled
config sip>set nat-translation enabled
config sip>return
```

The following CLI session configures the ME *private* IP Ethernet interface and SIP settings.

```
SIP>config cluster
```

```
config cluster>config box 1
config box 1>config interface eth4
Creating 'interface eth4'
config interface eth4>config ip private
Creating 'ip private'
config ip private>set ip-address static 10.0.20.1/24
config ip private>config sip
config sip>set admin enabled
config sip>set nat-translation enabled
config sip>return
```

CLI Sessions for Customer-A and Customer-B Networks

The following CLI sessions create the VLANs to the Customer-A and Customer-B networks, including the IP addresses, routing and classification tags, SIP settings, and a route to the IP using the gateways at IP addresses at 10.0.1.50 and 10.0.1.60. The ME uses classification tags to classify incoming traffic and routing tags to control the egress route. Configure a **classification-tag** on the incoming interface that matches the **routing-tag** on the egress interface.

```
config interface eth3>config vlan 10
Creating 'vlan 10'
config vlan 10>config ip 10.0.1.1
Creating '10.0.1.1'
config ip 10.0.1.1>set ip-address static 10.0.1.1/24
config ip 10.0.1.1>set classification-tag vlan10
config ip 10.0.1.1>set routing-tag vlan10
config ip 10.0.1.1>config sip
config sip>set nat-translation enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>return
config ip 10.0.1.1>config icmp
config icmp>return
config ip 10.0.1.1>config routing
config routing>config route default
Creating 'route default'
config route default>set gateway 10.0.1.50
config route default>return
config routing>return
config ip 10.0.1.1>return

config interface eth3>config vlan 20
Creating 'vlan 20'
config vlan 20>config ip 10.0.1.1
Creating '10.0.1.1'
config ip 10.0.1.1>set ip-address static 10.0.1.1/24
config ip 10.0.1.1>set classification-tag vlan20
config ip 10.0.1.1>set routing-tag vlan20
config ip 10.0.1.1>config sip
config sip>set nat-translation enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>return
config ip 10.0.1.1>config icmp
config icmp>return
config ip 10.0.1.1>config routing
config routing>config route default
Creating 'route default'
config route default>set gateway 10.0.1.60
```

```
config route default>return
config routing>return
config ip 10.0.1.1>return
```

CLI Session for the Internal Private Network

The following CLI session creates the VLAN to the internal *private* network, including the private IP address, routing and classification tags, SIP settings, and a default route to the public IP interface at 10.0.20.1. The ME uses classification tags to classify incoming traffic and routing tags to control the egress route. Configure a **classification-tag** on the incoming interface that matches the **routing-tag** on the egress interface.

```
config interface eth4>config vlan 30
Creating 'vlan 30'
config vlan 10>config ip 10.0.20.1
Creating '10.0.20.1'
config ip 10.0.20.1>set ip-address static 10.0.20.1/24
config ip 10.0.20.1>set classification-tag MAIN
config ip 10.0.20.1>set routing-tag MAIN
config ip 10.0.20.1>config sip
config sip>set nat-translation enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>return
config ip 10.0.20.1>config icmp
config icmp>return
config ip 10.0.20.1>config routing
config routing>config route default
Creating 'route default'
config route default>set destination default
config route default>return
config routing>return
config ip 10.0.1.1>return
```

CLI Session for the session-config-pool

The following CLI session creates two session configuration entries for handling egress traffic from Customer-A and Customer-B to the ME. The session-config-pool is for any traffic routed to the private network. The **egress-classification-tag** property, which needs to match the appropriate VLAN routing-tag on VLAN 30, selects the interface to the private network.

```
config>config vsp session-config-pool
config session-config-pool>config entry "Customer-A"
Creating entry "Customer A"
config entry "Custom A">config routing-settings
config routing-settings>set egress-classification-tag MAIN
config routing-settings>return
config entry "Custom A">return
config session-config-pool>config entry "Customer-B"
Creating entry "Customer B"
config entry "Custom B">config routing-settings
config routing-settings>set egress-classification-tag MAIN
```

Configuring VRRP

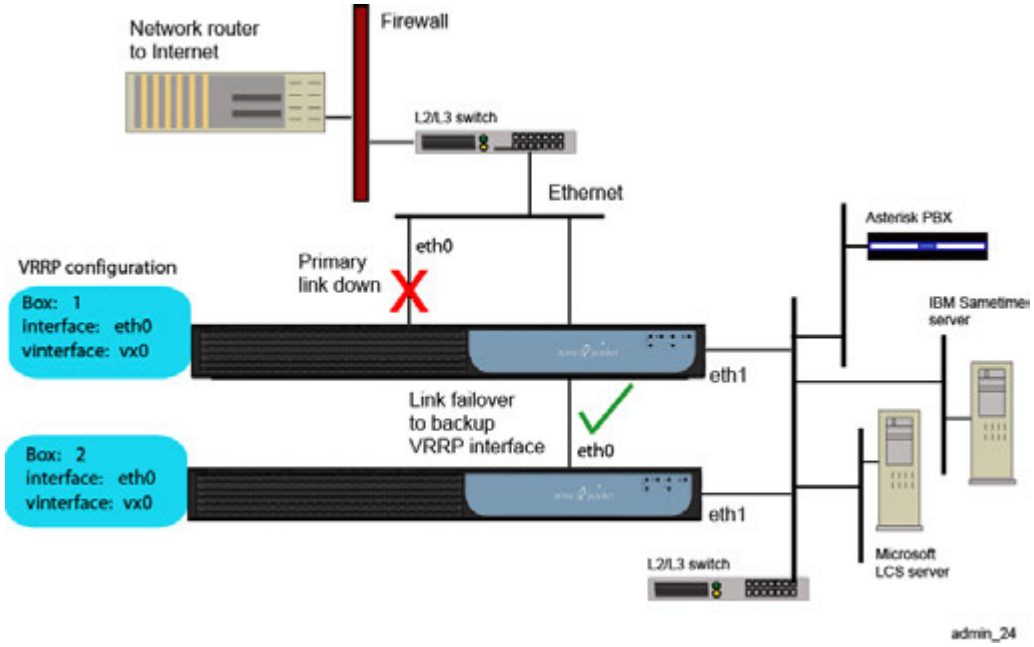
The Virtual Router Redundancy Protocol (VRRP) provides redundancy of IP interfaces within an ME cluster. The configuration for IP interfaces includes a list of

box/interface pairs. The first pair in this list is the *primary interface*. The second pair in the list is the *backup interface* and will take over if the primary goes down. You can configure additional levels of redundancy by specifying more box/interface pairs of lower priority. Priority is based on the positioning of the **set host-interface** command.

VRRP also provides redundancy of master services within a cluster. Each master service, including directory, database, and accounting, can be configured with a list of locations (box numbers within the cluster). The first location, such as box 1, is the primary; the second location (box 2) takes over if the primary fails. Specifying more locations in the list creates additional levels of redundancy.

The following image illustrates a sample network where VRRP reroutes traffic around a failed interface.

Figure 22–6 VRRP Rerouting Traffic Around a Failed Interface



If the master VRRP interface becomes unavailable, the VRRP election protocol enables a backup VRRP interface to assume mastership using the next prioritized interface in the list. However, if the original master VRRP interface (the interface with the highest priority) should once again become available, VRRP returns mastership to that interface.

See RFC 2338, *Virtual Router Redundancy Protocol*, for detailed information about this protocol.

CLI Session

The following CLI session creates two VRRP virtual interfaces (vx0 and vx1), and configures the physical host interfaces associated with each vinterface. On the vx0 interface, physical interface eth0 on box 1 will failover to eth0 on box 2, and then to eth0 on box 3. Note that each VRRP interface has its own IP (or VLAN) configuration.

```
SIP>config cluster
config cluster>config vrrp
config vrrp>config vinterface vx0
config vinterface vx0>set host-interface cluster box 1 interface eth0
config vinterface vx0>set host-interface cluster box 2 interface eth0
```

```
config vinterface vx0>set host-interface cluster box 3 interface eth0
config vinterface vx0>config ip name
Creating 'ip name'
config ip name>set ip-address static 1.1.1.1/24
config ip name>return
config vinterface vx0>return

config vrrp>config vinterface vx1
config vinterface vx1>set host-interface cluster box 3 interface eth1
config vinterface vx1>set host-interface cluster box 4 interface eth1
config vinterface vx1>config ip name
Creating 'ip name'
config ip name>set ip-address static 1.1.1.2/24
config ip name>return
config vinterface vx0>return
```

See RFC 2338, *Virtual Router Redundancy Protocol*, for detailed information about VRRP.

When configuring VRRP backing interfaces, Oracle recommends you have no more than two different MEs on the host list. You can, however, have more than one interface configured per box without any problems.

Here are some examples to illustrate acceptable and not acceptable configurations.

Not acceptable: There are interfaces from three different MEs listed for this VX interface. Oracle recommends you only have two MEs backing a VX.

```
config vrrp
 config vinterface vx10
  set group 1
  set host-interface cluster\box 1\interface eth1
  set host-interface cluster\box 2\interface eth1
  set host-interface cluster\box 3\interface eth1
  config ip 10.1.1.1
  return
 return
return
```

Not acceptable: There are interfaces from three different MEs listed for this VX interface and **preempt=true** is configured. This configuration is not supported at this time and will result in inconsistent behavior for the VX interface.

```
config vrrp
 config vinterface vx10
  set group 1
  set preempt true
  set host-interface cluster\box 1\interface eth1
  set host-interface cluster\box 2\interface eth1
  set host-interface cluster\box 3\interface eth1
  config ip 10.1.1.1
  return
 return
return
```

Acceptable: There are only two MEs listed as hosts for this VX.

```
config vrrp
 config vinterface vx10
  set group 1
  set host-interface cluster\box 1\interface eth1
  set host-interface cluster\box 2\interface eth1
  config ip 10.1.1.1
```

```
  return
 return
return
```

Acceptable: There are only two MEs listed as hosts for this VX, but each ME has two host interfaces configured on it.

```
config vrrp
 config vinterface vx10
  set group 1
  set host-interface cluster\box 1\interface eth1
  set host-interface cluster\box 1\interface eth2
  set host-interface cluster\box 2\interface eth1
  set host-interface cluster\box 2\interface eth2
  config ip 10.1.1.1
  return
 return
return
```

In either of these last two acceptable examples, it is okay to configure **preempt=true**.

Configuring Signaling Failover

The ME systems use signaling failover to preserve signaling sessions in a high-availability cluster. The cluster **master-service** maintains the signaling state of connections cluster-wide. With signaling failover, the signaling state information is transferred to the ME system taking over the signaling stream.

> **Note:** The call must be connected (at the SIP level) in order for signaling failover to take place. Signaling states prior to the "connected" state are not maintained in the cluster wide state table. Additionally, for TCP and TLS connections, the user agent must re-establish the connection once the failover has occurred. Since TCP/TLS are connection-oriented protocols, signaling state information is not maintained across failover. If TLS is used, the appropriate certificate must be loaded on the ME systems in the cluster.

Signaling information is maintained so that accurate call logs are recorded at the end of the call.

> **Note:** If there is a failure at the ME system holding the call log database, information will be lost.

Use the ME **show signaling-sessions** command to display failover state information.

CLI Session

```
SIP>config cluster
config cluster>set share-signaling-entries true
```

The **share-signaling-entries** property specifies whether or not all ME systems in a cluster exchange active SIP session information. When set to *true*, the ME systems exchange data. If the primary link then goes down, a backup link can use SIP session information from the primary device to handle existing calls.

The **share-signaling-entries** property should be set to *true* if you have configured VRRP (to provide the redundancy support). If you have VRRP enabled and configured, and if **share-signaling-entries** is set to true, signaling failover can take place.

Configuring Web Interface Settings

The Web object enables the Web server, providing access to the ME Management System graphical user interface. If you want to view SNMP traps through the GUI, you must also enable the server as a trap target. You enable and configure Web services on Ethernet and VLAN interfaces.

CLI Session

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
config ip boston1>config web
config web>set admin enabled
config web>set protocol https 443 0 "vsp tls certificate OS-E.cert.com"
config web>set trap-target enabled
```

Configuring Web Services

The **web-service** object enables the Web Services Definition Language (WSDL). WSDL is an XML-based language for describing Web services, and how to access them, in a platform-independent manner. Simple Object Access Protocol (SOAP) is the communication protocol used for communication between applications, based on XML.

A WSDL document is a set of definitions that describe how to access a web service and what operations it will perform. The ME uses it in combination with SOAP and XML Schema to allow a client program connecting to a web service to determine available server functions. The actions and data types required are embedded in the WSDL file, which then may be enclosed in a SOAP envelope. The SOAP protocol supports the exchange of XML-based messages with the ME system using HTTPS.

CLI Session

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
config ip boston1>config web-service
config web-service>set admin enabled
config web-service>set protocol https 443 0 "vsp tls certificate OS-E.company.com"
```

For detailed information on WSDL, refer to the *Net-Net OS-E – Management Tools*.

Enabling ICMP and Setting Rate Limits

The Internet Control Message Protocol (ICMP), defined in

RFC 792, is a TCP/IP protocol that determines whether a destination is unreachable. Using error and control messages between an host and an Internet gateway, ICMP verifies the validity of an IP address.

You can limit the rate at which ICMP messages are received on the ME system by setting ICMP rate and burst limits that prevent flooding of ICMP messages on the network. The rate setting is the maximum number of ICMP destination unreachable messages that the device can receive per second; the burst setting is the rate by which the number of ICMP messages that are discarded per second. Configuring the burst setting to a number lower than the rate setting will prevent ICMP message flooding.

CLI session

The following CLI session enables ICMP on the specified interface and sets ICMP rate and burst limits.

```
SIP>config cluster box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
Creating 'ip boston1'
config ip boston1>config icmp
config icmp>set admin enabled
config icmp>set limit 12 6
```

Enabling NTP and BOOTP Servers

By default, Network Time Protocol (NTP) and BOOTP services are enabled. The ME system uses NTP to synchronize time with external and local clocks using an NTP server, and the BOOTP protocol to allow an ME network client to learn its own IP address and boot information from a BOOTP server.

If addition to configuring NTP and BOOTP clients, you need to ensure that the NTP and BOOTP services are enabled on ME IP interfaces.

CLI Session

The following session enables BOOTP services on the specified ME IP interface and port number.

```
SIP>config cluster box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
Creating 'ip boston1'
config ip boston1>config bootp-server
config bootp-server>set admin enabled
config bootp-server>set port 67
```

The following session enables NTP services on the specified ME IP interface.

```
SIP>config cluster box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
Creating 'ip boston1'
config ip boston1>config ntp-server
config ntp-server>set admin enabled
```

Configuring the Network Time Protocol (NTP) Clients

The ME system uses the Network Time Protocol (NTP) to synchronize time with external and local clocks. Synchronized time across a network is important for critical functions such as packet and event time stamps or certificate validation.

You can configure an external NTP server to synchronize network time on the ME system. When you configure NTP, the system receives packets from the external NTP server that updates the local ME clock at specified NTP poll intervals.

CLI Session

The following session configures an external NTP server on the local ME NTP client. The session enables the ME NTP client, specifies the IP address of the remote NTP server, and the sets the poll-interval (in minutes) between network time updates from the NTP server.

```
config box>config ntp-client
config ntp-client>set admin enabled
config ntp-client>set server 192.168.23.76
config ntp-client>set poll-interval 5
```

Configuring the Bootstrap Protocol (BOOTP) Clients

The BOOTP commands allow you to configure the Bootstrap Protocol (BOOTP) client and server settings in an ME network cluster. BOOTP, described in RFC 951, is the Internet protocol that allows a network client to learn its own IP address and boot information from a BOOTP server.

In a network cluster, a BOOTP client requests its own IP address from the ME BOOTP server, as well as the IP address of the BOOTP server itself using the hardware MAC address. The BOOTP server responds to BOOTP client requests over the configured server port.

If a BOOTP session cannot be established between the ME client and server, BOOTP closes the session across the BOOTP interfaces after 60 seconds.

CLI Session

The following session configures a bootp client on the ME system. The session enables the bootp client, and sets the known bootp client and server ports for bootp requests and responses. UDP port 68 is the known bootp client port; UDP port 67 is the known bootp server port.

```
config box>config bootp-client
config bootp-client>set admin enabled
config bootp-client>set client-port eth1 68
config bootp-client>set server-port eth0 67
```

Configuring Session Initiation Protocol

For SIP applications running over Oracle networks, you need to enable the Session Initiation Protocol (SIP) on the ME IP interfaces. By default, the SIP protocol is enabled. However, you do need to configure the SIP operation mode, set the UDP, TCP, and TLS ports to use when listening for SIP messages, and include any certificates (generated and imported from a certificate authority) to be associated with the SIP interface.

- In *proxy* mode, the ME system only participates in SIP messages. Once the call is established, the phones send their voice traffic directly to each other without involving the proxy. SIP proxies offload tasks and simplify the implementation of end station telephones.

- The *B2BUA* is a SIP-based logical entity that receives and processes INVITE messages as a SIP User Agent Server (UAS). It also acts as a SIP User Agent Client (UAC) that determines how the request should be answered and how to initiate outbound calls. Unlike SIP proxy mode, the B2BUA maintains the call state and participates in all call requests.
- A *stateless* proxy forwards every request it receives and discards information about the request message once the message has been forwarded.

CLI Session

The following CLI session sets the SIP operation mode to "proxy."

```
SIP>config vsp
config vsp>config default-session-config
config default-session-config>config sip-settings
config sip-settings>set mode proxy
```

The following CLI session enables the SIP protocol on the specified IP interface, specifies the TCP, UDP and TLS ports to use when listening for SIP messages, and includes a certificate from an authorized certificate authority (CA).

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
Creating 'ip boston1'
config ip boston1>config sip
config sip>set admin enabled
config sip>set nat-translation enabled
config sip>set nat-add-received-from enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>set tls-port 5061
config sip>set certificate vsp tls certificate os-e.net.com
```

Load Balancing Across Media Engine Interfaces

Load balancing of SIP processing across interfaces requires both headed and backing interfaces.

The *headend* interface is the central distribution point. It does not perform SIP processing, it only forwards the calls to its configured backing interfaces. When you configure a SIP phone, configure the phone directly to the headend interface. To configure an IP interface as a headend interface, configure the **sip** object with backing interfaces. An interface is considered a headend interface if it has configured backing interfaces.

The *backing-interfaces* are identified within this **sip** object. In the **backing-interface** property, you reference previously configured IP interfaces. The backing interface is the location at which the ME terminates TCP and TLS connections (and where UDP transport messages arrive) and handles SIP processing. The ME uses round-robin load-balancing to distribute message across the configured backing interfaces.

To correctly configure load-balancing for SIP processing, you must do the following:

1. Configure the IP interfaces that will be used for both the headend and backing interfaces.

2. The SIP properties of the backing interfaces must match those of the head interface. For example, the interfaces must all use the same port assignments, and if you are using TLS, they must all use the same certificate.
3. You must enable the master services **registration** object so that the interfaces can share the registration database.

To verify your configuration, first ensure that all SIP properties match. From the CLI at the ME system that hosts the headend, execute the **show load-balance** command. This lists all associated backing interfaces (and statistics). From each box hosting a backing interface, execute **show backing-interface** to display configuration and statistics information.

CLI Session

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
config ip boston1>config sip
config sip>set admin enabled
config sip>set nat-translation enabled
config sip>set udp-port 5060
config sip>set tcp-port 5060
config sip>set tls-port 5061
config sip>set certificate "vsp tls certificate os-e.companyA.com"
config sip>set backing-interface "cluster box 1 interface eth0 ip backing1"
config sip>set backing-interface "cluster box 1 interface eth1 ip backing2"
config sip>set backing-interface "cluster box 2 interface eth0 ip
```

Configuring Media Port Pools

The **media-ports** object defines the ports and port ranges to assign to media streams on an Ethernet interface, such as NAT, media anchoring, and media recording.

CLI Session

The following CLI session enables the media-ports object, sets the starting port number, sets the total number of ports available for media streams, and enables the monitoring of idle ports (so that no traffic is sent to idle ports that are part of the media pool).

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
Creating 'ip boston1'
config ip boston1>config media-ports
config media-ports>set admin enabled
config media-ports>set base-port 20000
config media-ports>set count 5000
config media-ports>set idle-monitor enabled
```

Supported WebRTC Protocols

This section describes how to configure the Interactive Connectivity Establishment (ICE), Session Traversal Utilities for NAT (STUN), and Traversal Using Relay NAT (TURN) protocols in a WebRTC implementation.

What is Interactive Connectivity Establishment?

ICE is a protocol that establishes network paths for UDP-based media streams. It is an extension of the SDP offer/answer model and works by discovering and including all possible media transport addresses (known as candidates) in the SDP. Once SDPs are exchanged, ICE tests all possible media paths using the Session Traversal Utilities for the NAT (STUN) protocol as connectivity checks. Once the connectivity checking completes, the ICE agents settle on a final candidate pair to use for media transmission. The ME supports ICE on a per call-leg basis, meaning it can act as both the offering and answering ICE agent to satisfy this WebRTC requirement.

In addition to ICE, the ME also supports augmented ICE. In ICE the ME strips the candidates from the SDP while in augmented ICE the ME preserves all candidates received from a WebRTC endpoint. This provides the WebRTC endpoints the option to either anchor media on the ME or not.

For more information on ICE, visit http://tools.ietf.org/html/rfc5245.

What is Session Traversal Utilities for NAT?

In addition to connectivity checking, ICE relies heavily on STUN to discover all possible media candidates. During this candidate gathering phase, ICE agents perform STUN requests to discover their public IP addresses when behind a NAT device. The ME can be configured as a STUN server to satisfy these initial STUN requests.

For more information on STUN, visit http://www.ietf.org/rfc/rfc3489.

What is Traversal Using Relay NAT?

The TURN protocol assists clients located behind NAT devices to reach peers. In cases where clients and peers cannot create a direct communication path (for example, if both endpoints are behind individual NATs), it is necessary for an intermediate network device to relay data. The ME TURN Server acts as a communication-enabling alternative for such cases, relaying data between the NAT-hidden clients. When used with ICE, the ME TURN Server relay transport addresses are included in SDP ICE candidates received from clients. For more information on TURN, visit http://tools.ietf.org/search/rfc5766.

Session Traversal Utilities for NAT Required Methods

The following STUN methods are required for the ME's TURN Support:

- Allocate
- Refresh
- Send
- Data
- CreatePermission
- ChannelBind

Session Traversal Utilities for NAT Required Attributes

- CHANNEL-NUMBER
- LIFETIME
- XOR-PEER-ADDRESS
- DATA

- XOR-RELAYED-ADDRESS
- EVEN-PORT
- REQUESTED-TRANSPORT
- DONT-FRAGMENT
- RESERVATION-TOKEN

Non-Session Traversal Utilities for NAT Traversal Using Relays NAT Message

The ME supports the non-STUN ChannelData TURN message. This message carries application data between the TURN client and the server. Use of this message is optional for the client but required for the server if a channel has been bound to a remote peer.

TURN Server Long Term Credentials

The ME supports TURN server Long Term Credentials (LTC). You can configure one of the following three types of LTC:

- original: LTC authentication with username and password statically configured on the ME. These credentials are manually configured in the WebRTC browser.

  Note: For security reasons, Oracle does not recommend setting this value to **original**, as credentials can be extracted from the WebRTC browser.

- uberti: Follows the “draft-uberti-behave-turn-rest-00” document which uses a standard REST API for obtaining access to TURN services via ephemeral credentials. For more information on the “draft-uberti-behave-turn-rest-00”, see

  https://tools.ietf.org/html/draft-uberti-behave-turn-rest-00.

- SDKv1: Follows the RFC 7635 document which uses OAuth 2.0 to obtain and validate ephemeral tokens that can be used for authentication. By using ephemeral tokens, you can ensure that access to a STUN server can be controlled even if the tokens are compromised. For more information on RFC 7635, see

  https://tools.ietf.org/html/rfc7635.

  Note: SDKv1 is not completely RFC 7635 compliant due to the fact that WebRTC browser support is required and does not exist.

When you configure either the uberti or SDKv1 LTC types, you must configure a secret password. For increased security, ME uses a two-part password mechanism for passwords shared with other devices (also known as shared secrets). You must configure both a password and a tag. An enterprise or RADIUS server, for example, probably has a configured password that ME must use to access the server. This shared secret is the password. The tag is not the password itself, but rather a user-configurable name used to access the real password. By managing shared secrets, you can maintain the secrecy of the other passwords on other devices. An administrator can set up the tags and passwords; end users can work with the configuration files and use the password tag, without having access to the password itself.

ME uses a password store to maintain the actual password known to the other device. Using a password store allows the shared passwords to be stored outside of, and not displayed in, the configuration file. Password tags are stored in the ME configuration.

This password mechanism applies only to cases of ME using a shared secret. It does not apply to passwords created for users under the access object. (These are stored as hashed data, never as plaintext.)

Purging Traversal Using Relays Around the NAT Allocations

The **turn-allocation-purge** action allows you to manually remove TURN Allocations. Per RFC5766, TURN clients that no longer want to use an Allocation are encouraged to delete the Allocation via a TURN Refresh request with a requested lifetime of 0. However, some TURN clients currently do not remove Allocations and these remain in the ME until they expire.

> **Note:** Ensure you remove only unused Allocations. Removing valid and in-use Allocations disrupts a WebRTC call using the ME's TURN server.

Syntax

```
turn-allocation-purge [turn-client]
```

Arguments

- [*turn-client*]: The TURN client's IP address and port.

> **Note:** By default, the **turn-allocation-purge** action purges all TURN Allocations, unless otherwise specified.

Media Engine Encryption

Secure Real-Time Transport Protocol (SRTP) is secure Real-Time Transport Protocol (RTP) designed to provide encryption, authentication, and integrity to RTP streams. Used along with Source Description RTCP (SDES), encryption keys are exchanged in the SDP offer and answer using the crypto attribute. The ME supports SDES-SRTP encryption and decryption on a per call-leg basis to satisfy this WebRTC requirement.

For more information on SDES, visit http://tools.ietf.org/html/rfc4568.

In addition to SDES-SRTP, the ME also supports Datagram Transport Layer Security (DTLS) as a method for encryption. DTLS works similarly to SDES-SRTP in that encryption keys are exchanged in the SDP offer and answer using the crypto attribute and the ME supports DTLS on a per call-leg basis.

For more information on DTLS, visit http://tools.ietf.org/html/rfc4347.

Data Channel Support

The ME supports data channels for anchored media.

Data channels use the SCTP protocol as a generic transport service which allows web browsers and native mobile applications to exchange non-media data between peers. For more information on data channels, visit

https://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-11.

Note: You must have the **session-config > media > anchor** property set to **enabled** for data channels to work.

Configuring Interactive Connectivity Establishment

To configure ICE on the ME, you must enable session-wide media anchoring.

You must also enable symmetric RTP, which returns RTP based on the source IP address and UDP port in the received RTP. NAT modifies data in the IP header only and the SDP payload is left unchanged. By using the source IP address and UDP port from the received RTP, the ME sends traffic back to the NAT device instead of the untranslated addresses in the SDP.

In addition to these session-wide settings, you must also configure ICE for incoming and outgoing WebRTC sessions.

To enable system-wide media anchoring and symmetric RTP:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **media**.
3. **anchor**: Set to **enabled** to enable media anchoring for this media session. Media anchoring forces the SIP media session to traverse the ME.
4. Click **Configure** next to **nat-traversal**.
5. **symmetricRTP**: Set to **true** to enable symmetric RTP for this media session. When enabled, symmetric RTP returns RTP based on the source IP address and UDP port in the received RTP. NAT modifies data in the IP header only and the SDP payload is left unchanged.
6. Click **Set**. You are returned to the **media** object.
7. Click **Set**. Update and save the configuration.

To enable ICE for incoming WebRTC sessions:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **in-ice-settings**.
3. **admin**: Set to **enabled** to enable ICE on this call leg.
4. **connectivity-check-max-retransmits**: Specify the number of times the ME retransmits ICE STUN connectivity checks before labeling a candidate pair as Failed. To achieve maximum interoperability with Chrome, set this value to no less than **200**.
5. Click **Set**. Update and save the configuration.

To enable ICE for outgoing WebRTC sessions:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **in-ice-settings**.
3. **admin**: Set to **enabled** to enable ICE on this call leg.
4. **delay-stun-responses**: (*Advanced*) Set to **enabled** so that the ME does not respond to STUN until the 200 OK is received.

Note: To view Advanced properties, click the **Show advanced** button.

5. **suppress-re-invites**: (*Advanced*) Set to **enabled**. When enabled so that the ME does not send a re-INVITE when ICE completes successfully.
6. Click **Set**. Update and save the configuration.

Configuring Augmented Interactive Connectivity Establishment

The ME supports augmented ICE. In ICE the ME strips the candidates from the SDP while in augmented ICE the ME preserves all candidates received from a WebRTC endpoint. This provides the WebRTC endpoints the option to either anchor media on the ME or not.

If you are configuring the ME for augmented ICE you must complete the configuration procedure for ICE plus some additional configuration. For details on configuring ICE, see "Configuring Interactive Connectivity Establishment".

Note: By default, augmented ICE is disabled on the ME. If you are using augmented ICE for a particular session, enable it on that named **session-config-pool > entry** only. Leave the **default-session-config** object's augmented ICE setting disabled so as not to affect all named sessions, which can cause an adverse negative impact.

To configure augmented ICE:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **media**.
3. **augmented-ice**: Set to **enabled** to enable augmented ICE.
4. Click **Set**. You are returned to the **media** object.
5. Click **Configure** next to **in-encryption**.
6. **mode**: Select **pass-thru** from the drop-down list.
7. Click **Set**. You are returned to the **media** object.
8. Click **Configure** next to **out-encryption**.
9. **mode**: Select **pass-thru** from the drop-down list.
10. Click **Set**. Update and save the configuration.

Configuring Trickle Interactive Connectivity Establishment

The ME supports trickle ICE, a draft extension to RFC 5245 that allows ICE agents to incrementally exchange remote candidate information. Trickle ICE support considerably reduces call setup time by allowing ICE to run before the candidate harvesting phase has completed by sending empty or partial media candidate lists in the SDP.

To configure trickle ICE:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.

2. Click **Configure** next to **in-ice-settings**.
3. **trickle-ice**: Set to **enabled** or disabled to determine if trickle ICE is offered and supported on each call leg.
4. Click **Configure** next to **out-ice-settings**.
5. **trickle-ice**: Set to **enabled** or disabled to determine if trickle ICE is offered and supported on each call leg.
6. Click **Set**. Update and save the configuration.

Three show commands allow you to view trickle ICE information: **show ice-local-candidates**, **show ice-remote-candidates**, **show ice-candidate-pair-status**.

show ice-local-candidates

Displays ICE information for the local candidates used by each state machine.

Sample Output

```
SIP>show ice-local-candidates
session-id       leg checklist  transport  componentID type priority foundation
-----------      --- ---------  ---------  ----------- ---- -------- -----------
0x8c4ef6081de8c26 1 172.44.10.55:20676 UDP 1 host 2130706431 1172.44.10.55:20677
```

Properties

- session-id: The ID of the session that owns the ICE state machine.
- leg: The call-leg on which the ICE state machine is running.
- checklist: The checklist number that owns the candidate. This is also known as the media description index.
- transport: The IP, port, and transport protocol of the candidate.
- componentID: The ICE component ID. This value is an integer.
- type: The ICE candidate type. This can be either host, srflx, prflx, or relay.
- priority: The candidate priority.
- foundation: The foundation string.

show ice-remote-candidates

Displays ICE information for the remote candidates received from the remote peer.

Sample Output

```
SIP>show ice-remote-candidates
session-id leg checklist transport componentID type priority foundation
---------- --- --------- --------- --------------- -------- ----------
0x8c4ef6081de8c26 1 0 172.44.10.57:22656 UDP 1 host 2130706431 1
172.44.10.57:22657 UDP 2
```

Properties

- session-id: The ID of the session that owns the ICE state machine.
- leg: The call-leg on which the ICE state machine is running.
- checklist: The checklist number that owns the candidate. This is also known as the media description index.
- transport: The IP, port, and transport protocol of the candidate.
- componentID: The ICE component ID. This value is an integer.

- type: The ICE candidate type. This can be either host, srflx, prflx, or relay.
- priority: The candidate priority.
- foundation: The foundation string.

show ice-candidate-pair-status

Displays information and state for each ICE candidate pair.

Sample Output

```
SIP>show ice-remomte-candidates
session-id leg checklist transport componentID type priority foundation
---------- --- --------- --------- ----------- ---- -------- ----------
0x8c4ef6081de8c26 1 0 172.44.10.57:22656 UDP 1 host 2130706431 1
```

Properties

- session-id: The ID of the session that owns the ICE state machine.
- leg: The call-leg on which the ICE state machine is running.
- checklist: The checklist number that owns the candidate. This is also known as the media description index.
- transport: The IP, port, and transport protocol of the candidate.
- componentID: The ICE component ID. This value is an integer.
- type: The ICE candidate type. This can be either host, srflx, prflx, or relay.
- priority: The candidate priority.
- foundation: The foundation string.

show ice-candidate-pair-status

Displays information and state for each ICE candidate pair.

Sample Output

```
SIP>show ice-candidate-pair-status
session-id leg checklist local remote state componentID nominated
---------- --- --------- ----- ------ ----- ----------- ---------
0x8c4ef6081de8c26 10 172.44.10.55:20676 UDP 172.44.10.57:22656 UDP Succeeded1 true
```

Properties

- session-id: The session ID on which ICE is running.
- leg: The call leg on which ICE is running.
- checklist: The checklist that owns the candidate pair. This is also known as the media description index.
- local: The local candidate in the pair.
- remote: The remote candidate in the pair.
- state: The pair state. This can be either Frozen, Waiting, Succeeded, or Failed.
- componentID: The componentID of the pair. This value is an integer.
- nominated: Specifies whether or not this pair has been nominated for media transmission.

Configuring Session Traversal Utilities For the NAT

In addition to an ICE server, the ME can also be configured as a STUN server.

To configure the ME as a STUN server:

1. Click the **Configuration** tab and select the **cluster > box > interface > ip** object.
2. Click **Configure** next to **stun-server**.
3. **admin**: Set to **enabled** to enable the ME as a STUN server.
4. Click **Add port** to configure a port for the STUN server.
5. **transport**: Select from the drop-down list the transport protocol over which STUN messages are exchanged between a SIP endpoint and the ME STUN server. Valid values **UDP**, **TCP**, and **TLS**. The default value is **UDP**.
6. **port**: Specify the port over which STUN messages are exchanged between a SIP endpoint and the ME STUN server. The default value is **3478**.
7. Click **Create**. You are returned to the **stun-server** object.
8. Click **Set**. Update and save the configuration.

For more information on the stun-server object, see the *WebRTC Session Controller Media Engine Object Reference*.

Configuring Traversal Using Relay NAT

To enable the TURN server on the ME, you must enable STUN and configure several properties within **stun-server** object.

To enable TURN:

1. Click the **Configuration** tab and select the **cluster > box > interface > ip** object on which you are configuring TURN.
2. Click **Configure** next to **stun-server**.
3. **admin**: Set to **enabled** to enable STUN.
4. **allow-turn**: Set to **enabled**.
5. **relay-interface**: Select an interface provisioned with media ports.
6. **type:** Specifies the type of LTC authentication to use. The ME supports the following three types of LTC authentication:
   - original: LTC authentication with username and password statically configured on the ME. These credentials are manually configured in the WebRTC browser.

     Note: For security reasons, Oracle does not recommend setting this value to **original**, as credentials can be extracted from the WebRTC browser.

   - uberti: Follows the "draft-uberti-behave-turn-rest-00" document which uses a standard REST API for obtaining access to TURN services via ephemeral credentials. For more information on "draft-uberti-behave-turn-rest-00", see https://tools.ietf.org/html/draft-uberti-behave-turn-rest-00.

Note: SDKv1 is not completely RFC 7635 compliant due to the fact that WebRTC browser support is required and does not exist.

- SDKv1: Follows the RFC 7635 document which uses OAuth 2.0 to obtain and validate ephemeral tokens that can be used for authentication. By using ephemeral tokens, you can ensure that access to a STUN server can be controlled even if the tokens are compromised. For more information on RFC 7635, see https://tools.ietf.org/html/rfc7635.

7. **ltc-authentication-realm**: Specify the realm to use for STUN LTC authentication.
8. **ltc-auth-provider-shared-secret**: Enter the shared secret with the TURN LTC Authentication Provider.
9. **ltc-auth-provider-pw-ttl**: Specify the lifetime length for the shared password.
10. Click **Set**. Update and save the configuration.

Three show commands allow you to view TURN server information: **show turn-allocations**, **show turn-destinations**, **show turn-server**.

show turn-allocations

Provides information for each TURN client allocated server relay port. WebRTC endpoints typically allocate a relay port for each media stream.

Sample Output

```
SIP>show turn-allocations
server-port: 172.44.10.60:3478
user: TurnMike@TurnRealm
client: 10.1.26.32:56863
client-transport: UDP
relay-port: 172.44.10.60:20975
relay-transport: UDP
destination-count: 1
client-to-peer-packets: 61489
client-to-peer-bytes: 5822176
peer-to-client-packets: 61585
peer-to-client-bytes: 5512977
bandwidth-max: 150 kbits-per-second
allocation-time: 07:12:02.147705 Tue 2014-01-21
duration: 1177 seconds
remaining: 503 seconds
```

Properties

- server-port: The IP and port of the TURN server listener.
- user: The user and realm of TURN LTC.
- client: The IP and port of the TURN client.
- client-transport: The transport method used for client/server communication.
- relay-port: The IP and port of the TURN relay for this Allocation.
- relay-transport: The transport method used for server/peer communication.
- destination-count: The number of TURN destinations for this Allocation.
- client-to-peer-packets: The number of packets relayed from client to peer for this Allocation.

- client-to-peer-bytes: The number of bytes relayed from client to peer for this Allocation.
- peer-to-client-packets: The number of packets relayed from peer to client for this Allocation.
- peer-to-client-bytes: The number of bytes relayed from peer to client for this Allocation.
- bandwidth-max: Currently not supported.
- allocation-time: The time the Allocation was created and/or refreshed.
- duration: The duration of the TURN Allocation.
- remaining: The time remaining for the TURN Allocation.

show turn-destinations

Provides information for each TURN peer associated with a TURN client.

Sample Output

```
SIP>show turn-destinations
index: 1
turn-client: 10.1.26.32:56864
turn-allocation: 0xd1c27f75
turn-relay: 172.44.10.60:20927
relay-transport: UDP
turn-peer: 172.44.10.60:20972
channel-number: 16384
chan-expire-time: 07:32:02.972915 Tue 2014-01-21
chanRemaining: 561 seconds
dest-permissions: Allowed
perm-expire-time: 07:32:02.972915 Tue 2014-01-21
permRemaining: 261 seconds
dest-anchored: true
```

Properties

- index: The index of this Destination. An Allocation can have multiple destinations.
- turn-client: The IP and port of the TURN client.
- turn-allocation: The handle of the Allocation owning this Destination.
- turn-relay: The IP and port of the TURN relay for this Destination.
- relay-transport: The transport used for server/peer communication.
- turn-peer: The IP and port of the TURN relay for this Destination.
- channel-number: The TURN channel number for this Destination (a value of 0 means unused).
- chan-expire-time: The time the TURN channel expires.
- chanRemaining: The time remaining before the TURN channel expires.
- dest-permissions: Permissions installed by the TURN client for this Destination.
- perm-expire-time: The time Permissions expire.
- permRemaining: The time remaining before Permissions expire.
- dest-anchored: Indicates if media is anchored for this TURN relay.

show turn-server

Provides information regarding the configured TURN server.

Sample Output

```
NNOS-E>show turn-server
                 index: 0
               ifindex: 1
             transport: UDP
            ip-address: 10.138.236.34
                  port: 3478
           turn-server: enabled
         relay-ifindex: 0
         relay-address: 10.138.236.34
relay-allocation-count: 0
           rx-requests: 0
          tx-responses: 0
    tx-error-responses: 0
              discards: 0
           ltcAuthType: sdkv1
          ltcSecretTag: diwakarExample
              ltcPwTtl: 3600
```

Properties

- index: This index of this TURN server.
- ifindex: The interface index used for this TURN server.
- transport: The transport method used for this TURN server.
- ip-address: The IP address used for this TURN server listener.
- port: The port used for this TURN server listener.
- turn-server: The name of the server.
- relay-ifindex: The interface index used for the TURN server relay.
- relay-address: The TURN server address.
- relay-allocation-count: The number of TURN Allocations in use by the TURN server.
- rx-requests: The number of STUN/TURN requests received.
- tx-responses: The number of STUN/TURN success responses sent.
- tx-error-responses: The number of STUN/TURN error responses.
- discards: The number of STUN/TURN messages discarded.
- ltcAuthType: The ME LTC Authentication type for this server.
- ltcSecretTag: The tag that references the ME's LTC provisioned secret.
- ltcPwTtl: The configured lifetime of the LTC provisioned secret.

Configuring Static Datagram Transport Layer Security Certificates

When using DTLS in a WebRTC implementation, you must configure a static certificate via the default-dtls-settings configuration object.

To configure a static DTLS certificate:

1. Click the **Configuration** tab and select the **vsp > tls** object.
2. Click **Configure** next to **default-dtls-settings**.

3. **certificate-file**: Specify the name of the certificate file used to establish connections made with this object. The ME supports the following certificate formats: PKCS#12: Public Key Cryptography Standard #12 format, often from Microsoft IIS Version 5 (binary), PEM: Privacy Enhanced Mail format, from any Open SSL-based web server (ASCII).
4. **passphrase-tag**: Specify the passphrase associated with the certificate file. Use this property if the certificate file is encrypted to have its private key information protected. This passphrase must match the string that the certificate was encrypted with.
5. Click **Set**. Update and save the configuration.

Execute the **show certificates -v** action to verify that the certificate is working.

> **Note:** For more information on configuring certificates and viewing certificate statistics, see the *WebRTC Session Controller Media Engine Object Reference*.

Configuring Encryption

Although the ME supports encryption, it does not require it from WebRTC endpoints. If an endpoint does not support encryption, it does not include a crypto key in its answer SDP and RTP is automatically used to transport media.

Because the ME always sends media encrypted out, you must configure the in-leg to allow encryption and the out-leg to require it.

You can configure the ME to use SDES-SRTP, DTLS, or specify multiple and let the WebRTC endpoint decide which type of encryption to use.

> **Note:** If you configure **encryption-preferences** but do not have type set to **multiple**, it does not work. If you specify **multiple** but do not configure **encryption-preferences**, you receive an error.

To configure in-leg encryption:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **in-encryption**.
3. **mode**: Select **allow** from the drop-down list. This allows the ME to receive encryption on the in-leg.
4. **type**: Select the type of encryption you want to use from the drop-down list.
   - RFC3711: Use the SDES-SRTP protocol for encryption.
   - DTLS: Use the DTLS protocol for encryption.
   - multiple: Both SDES-SRTP and DTLS are offered for encryption. Using the **encryption-preferences** property, assign each protocol a priority and the type of encryption used depends upon the WebRTC endpoint.
5. If you set **type** to **multiple**, click **Add encryption-preferences** and click **Edit**.
6. **priority**: Set to **1**.
7. **type**: Select **DTLS** from the drop-down list and click **Set**.

Note: Always give DTLS a priority of 1 and RFC-3711 a priority of 2.

8. CLick **Add encryption-preferences** and click **Edit**.
9. **priority**: Set to **2**.
10. **type**: Select RFC3711 from the drop-down list.
11. Click **Set**. Update and save the configuration.

To configure out-leg encryption:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **out-encryption**.
3. **mode**: Select **require** from the drop-down list. This allows the ME to offer encryption.
4. **type**: Select the type of encryption you want to use from the drop-down list.
   - RFC3711: Use the SDES-SRTP protocol for encryption.
   - DTLS: Use the DTLS protocol for encryption.
   - multiple: Both SDES-SRTP and DTLS are offered for encryption. Using the **encryption-preferences** property, assign each protocol a priority and the type of encryption used depends upon the WebRTC endpoint.
5. If you set **type** to **multiple**, click **Add encryption-preferences** and click **Edit**.
6. **priority**: Set to **1**.
7. **type**: Select **DTLS** from the drop-down list and click **Set**.

Note: Always give DTLS a priority of 1 and RFC-3711 a priority of 2.

8. CLick **Add encryption-preferences** and click **Edit**.
9. **priority**: Set to **2**.
10. **type**: Select **RFC3711** from the drop-down list.
11. Click **Set**. Update and save the configuration.

The **show ice-dtls-status** show command provides information per call-leg for sessions using DTLS encryption.

```
SIP>show ice-dtls-status
        session-id: 0x4c40106b423123b
               leg: 1
            stream: 0
           address: 172.30.12.82:24472
            remote: 172.30.12.82:24352
              type: 1-RTP
              role: Passive
             state: Succeed
```

Properties:

- session-id: The unique ID of the ME session.
- leg: Specifies in-leg (0) or out-leg (1).

- stream: The media stream index, either audio (0) or video (1).
- address: The local ME IP and port for this DTLS socket.
- remote: The remote peer IP and port for this DTLS socket.
- type: Specifies the type of ICE port, either RTP (1) or RTCP (2).
- role: Specifies the DTLS role, either Passive or Active.
- state: The state of the DTLS socket, either Connected, Listening, Succeeded, or Closed.

Disabling the Datagram Transport Layer Security Cookie Exchange

For the ME to work properly in a WebRTC environment, you must configure it to stop exchanging cookies during the DTLS negotiation.

To stop the DTLS cookie exchange:

1. Click the **Configuration** tab and select the **vsp > tls** object.
2. Click **Configure** next to **default-dtls-settings**.
3. **dtls-cookie-exchange**: Set to **disabled** to stop exchanging cookies during the DTLS negotiation.
4. Click **Set**. Update and save the configuration.

Real-Time Transport Protocol/Real-Time Control Protocol Multiplexing

The ME supports RTP/RTCP Multiplexing which, when enabled, bundles all of the RTP and RTCP media through the same port.

When initiating a bundled call, the ME inserts the necessary information into the INVITE message's SDP in the following format:

```
m=RTP <Port>
a=rtcp=<RTCP Port>
a=rtcp-mux
```

If the recipient supports RTP/RTCP multiplexing, it returns the following in the SDP of its 200 OK response:

```
m=RTP/RTCP <Port>
a=rtcp-mux
```

If the recipient does not support RTP/RTCP multiplexing, it returns its own RTO and RTCP port numbers in the SDP without a=rtcp-mux and multiplexing is not used.

The ME does not support audio and video multiplexing, audio and video streams bundled to the same port. To ensure the recipient that the ME is talking to knows this, you must strip out any Synchronization Source (SSRC) information from the SDP.

To configure RTP/RTCP multiplexing for incoming WebRTC calls:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **in-sdp-attribute-settings**.
3. **rtcp-mux**: Enables or disabled RTP/RTCP multiplexing. By default this property is **disabled**.
4. **ssrc-in-sdp**: Set to **strip** to strip out any SSRC information from the SDP.

5. **patch-audio-group**: (*Advanced*) Set to **enabled**. When the ME receives an offer SDP with both audio and video and the line a=group BUNDLE audio video and a response with only audio, it must perform certain functions in order to get the audio to work.

   When enabled, the ME performs the following modifications:

   - The ME performs RTP/RTCP multiplexing on the in-leg, regardless of the user configuration.
   - The ME adds bundling information by adding the following to the SDP.

     ```
     a=group BUNDLE audio
     a=mid:audio
     ```
   - The ME generates WebRTC-style SSRC values and adds them to the SDP as well as the RTP/RTCP stream.

   Note: To view Advanced properties, click the **Show advanced** button.

6. Click **Set**. Update and save the configuration.

To configure RTP/RTCP multiplexing for outgoing WebRTC calls:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **out-sdp-attribute-settings**.
3. **rtcp-mux**: Enables or disabled RTP/RTCP multiplexing. By default this property is **disabled**.
4. **ssrc-in-sdp**: Set to **strip** to strip out any SSRC information from the SDP.
5. Click **Set**. Update and save the configuration.

Configuring SDP Regeneration

To ensure the ME represents itself properly in the SDP, it must regenerate incoming SDPs to list the attributes it supports and strip out unsupported attributes. To do this, you must configure the **sdp-regeneration** object.

Note: If the ME forwards an SDP containing attributes it does not support, the WebRTC call does not work.

To configure SDP regeneration:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **sdp-regeneration.**
3. **regenerate**: Set to **enabled** to regenerate the SDP, with the configured settings, before forwarding it along.
4. **add-rtpmaps**: Set to **enabled** so the ME includes rtpmap attributes for well-known CODECs when the rtpmap is not included in the SDP by the original endpoint.
5. **pass-attributes**: Click **Edit pass-attribute**.

6. Enter the attributes to be included in the SDP. The following attributes must be added:

- ice-ufrag
- ice-pwd
- candidate
- remote-candidates
- rtcp
- rtcp-mux
- You must enter attributes one at a time. After entering an attribute and clicking **Add**, a new field to enter the next attribute appears.

> **Note:** These attributes do not appear in the drop-down list and must be entered into the provided blank field.

Media Steering For Unknown Endpoints

When the SE creates an ME session to anchor media, the network on which the two endpoints (caller and callee) reside may not be known by the ME. In these cases, nominal network addresses can be used to steer media correctly through the ME by performing media service route lookups. This ensures that the media resources are allocated from a media interface that can reach the endpoint.

The following named variables can be used to configure the nominal network addresses (and optionally ports) used to steer media through the ME.

- inleg.source.ip
- inleg.source.port
- outleg.source.ip
- outleg.source.port

These named variables are used to steer media through the ME by providing nominal network addresses to perform a media service route lookup, allocating media resources on an interface that can reach the remote endpoint. The **inleg.source.ip** and **outleg.source.ip** values can be set to the IP address of the ME media interface to force media resource allocation from that specific interface. These values can also be used to specify a network IP address (for example, 1.1.1.1) for cases where the ME has multiple media interfaces on the same subnet for load balancing purposes.

> **Note:** For information on configuring named variables, see Using Regular Expressions in the *WebRTC Session Controller Media Engine Objects and Properties Reference* guide.

Configuring a Browser to SIP Call

When an Internet browser makes a call to a SIP phone residing on the customer core network, the ME uses the "web-to-sip" session-config call flow. Adding these named variables to the existing "web-to-sip" session-config steers the media from the access to the core networks.

Note: The endpoint initiating the call resides on the "inleg" and the endpoint receiving the call is on the "outleg".

To steer the media correctly for this call flow, the **inleg.source.ip** can be configured as 1.1.1.1 and the **outleg.source.ip** can be configured as 2.2.2.2. Configuring the named variables this way forces the allocation of media resources from the access interface (1.1.1.1) to reach the browser, and the core interface (2.2.2.2) to reach the SIP phone.

Example 22–1 shows how to add named variables to an existing "web-to-sip" session-config.

Example 22–1 Web-to-SIP

```
config vsp
 config session-config-pool
  config entry web-to-sip
   config named-variables
    config named-variable inleg.source.ip
     set value 1.1.1.1
    return
    config named-variable outleg.source.ip
     set value 2.2.2.2
    return
```

Configuring a SIP to Browser Call

When a SIP phone makes a call to an Internet browser residing on the customer core network, the ME uses the "sip-to-web" session-config call flow. Adding these named variables to the existing "sip-to-web" session-config steers the media from the core to the "access" networks.

Note: The endpoint initiating the call resides on the "inleg" and the endpoint receiving the call is on the "outleg".

To steer media correctly for this call flow, the **inleg.source.ip** can be configured as 2.2.2.2 and the **outleg.source.ip** can be configured as 1.1.1.1. Configuring the named variables this way forces the allocation of media resources from the core interface (2.2.2.2) to reach the SIP phone, and the access interface (1.1.1.1) to reach the browser.

Example 22–2 shows how to add named variables to an existing "sip-to-web" session-config.

Example 22–2 SIP-to-Web

```
config vsp
 config session-config-pool
  config entry sip-to-web
   config named-variables
    config named-variable inleg.source.ip
     set value 2.2.2.2
    return
    config named-variable outleg.source.ip
     set value 1.1.1.1
    return
```

Configuring a Browser to Browser Call

When an Internet browser makes a call to another Internet browser, the ME uses either the "web-to-web-anchored" or "web-to-web-anchored-conditional" session-config call flow. If you require media steering for these calls, use the **inleg.source.ip** and **outleg.source.ip** named variables.

> **Note:** The endpoint initiating the call resides on the "inleg" and the endpoint receiving the call is on the "outleg".

Since both endpoints in this call flow reside on the Internet, specify the ME's access interface (1.1.1.1) for both the **inleg.source.ip** and **outleg.source.ip**.

Example 22–3 shows how to add named variables to an existing "web-to-web-anchored" session-config call flow.

Example 22–3 'Web-to-Web-Anchored'

```
config vsp
 config session-config-pool
  config entry web-to-web-anchored
   config named-variables
    config named-variable inleg.source.ip
     set value 1.1.1.1
    return
    config named-variable outleg.source.ip
     set value 1.1.1.1
    return
```

Message Session Relay Protocol Interworking

The ME supports the Message Session Relay Protocol (MSRP) interworking. MSRP interworking allows communication between WebRTC and Rich Communication Suite (RCS) endpoints. This protocol is used for transmitting a series of instant message chats and file transfers within the context of a session.

For more information on MSRP, see https://tools.ietf.org/html/rfc4975.

Configuring MSRP Interworking

To enable MSRP interworking on the ME, you must configure the **in-msrp-session-leg** and **out-msrp-session-leg** objects.

To configure in-leg MSRP interworking:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **in-msrp-session-leg**.
3. **admin**—Set to **enabled** to enable MSRP interworking.
4. **msrp-leg-transport**—Specify the MSRP transport method for WebRTC or RCS.
5. **default-media-interface**—Specify the local media interface to use for MSRP if svc-routing fails to locate the appropriate interface.
6. Click **Set**. Update and save the configuration.

To configure out-leg MSRP interworking:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to **out-msrp-session-leg**.
3. **admin**: Set to **enabled** to enable MSRP interworking.
4. **msrp-leg-transport**: Specify the MSRP transport method for WebRTC or RCS.
5. **default-media-interface**: Specify the local media interface to use for MSRP if svc-routing fails to locate the appropriate interface.
6. Click **Set**. Update and save the configuration.

The ME allows you to choose the criteria on which to validate received MSRP messages. When the ME receives a MSRP message, the MSRP session manager checks the configured match criteria to verify the message belongs to the MSRP session associated with the connection transporting the MSRP message.

You can choose to match on the following criteria.

- scheme
- host
- port
- transport
- sessionId

To configure flexible MSRP message matching:

1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click **Configure** next to either **in-msrp-session-config** or **out-msrp-session-leg**.
3. **message-match-criteria**: Select the criteria on which you want to match MSRP messages.
4. Click **Set**. Update and save the configuration.

There are four show commands that allow you to view MSRP interworking statistics: **show active-msrp-sessions**, **show msrp-connections**, **show msrp-listeners**, and **show msrp-stats**.

The **show active-msrp-sessions** command displays information regarding active MSRP session statistics.

```
SIP>show active-msrp-sessions
Active MSRP Sessions:
------------------------------------------------------------------------
session-handle: 0xC7C634F3
inleg-type: Msrp
inleg-state: CONNECTED
outleg-type: Msrp
outleg-state: CONNECTED
caller-session-id: mhnb1ad02f
caller-path: msrp://wscAddress.invalid:2855/mhnb1ad02f;ws
called-session-id: 2511644601
called-path: msrp://10.138.238.49:53847/2511644601;tcp
create-time: 12:09:59.163681 Thu 2014-10-30
duration: 24 seconds
Total Active MSRP Sessions: 1
```

Properties

- session-handle: The handle for this session.
- inleg-type: The type of endpoint of the in-leg session.
- inleg-state: The state of the in-leg session endpoint.
- outleg-type: The type of endpoint of the out-leg session.
- outleg-state: The state of the out-leg session endpoint.
- caller-session-id: The session ID of the calling endpoint.
- caller-path: The path of the calling endpoint.
- called-session-id: The session ID of the called endpoint.
- called-path: The path of the called endpoint.
- create-time: The time this session was created.
- duration: The length, in seconds, of this session.

The **show msrp-connections** command displays statistics regarding all of the connections used by the current MSRP sessions.

```
SIP>show msrp-connections
----------------------------------------------------------------------
Process Proto LocalAddress RemoteAddress State Direction RefCount
----------------------------------------------------------------------
SIP TCP 10.138.236.35:23365 10.138.238.49:53847 Connected Answer 1
SIP WS 10.138.236.35:23385 10.138.238.49:53848 Connected Originate 1
```

Properties

- Process: The signaling process being used for this connection.
- Proto: The media transport protocol being used for this connection.
- LocalAddress: The local IP address and port.
- RemoteAddress: The remote IP address and port.
- State: The state of the connection.
- Direction: The current direction of media transfer.
- RefCount: Not currently supported. This value should always be 1.

The **show msrp-listeners** command displays information listing all ports on the ME interface that are waiting for MSRP connections.

```
SIP>show msrp-listeners
-------------------------------------------------------------
Process Proto Address Connections Rejected Current Timeouts
-------------------------------------------------------------
SIP     WS    10.138.236.35:23385 0 0       1       0
```

Properties

- Process: The signaling process being used for this port.
- Proto: The media transport protocol being used for this port.
- Address: The IP address for this port.
- Connections: The number of connections available on this port.
- Rejected: The number of connections rejected by this port.
- Current: The number of current connections on this port.
- Timeouts: The number of timeouts that have occurred on this port.

The **show msrp-stats** command displays information regarding MSRP interworking statistics.

```
SIP>show msrp-stats
totalSessions: 4
totalConnections: 2
totalActiveConnections: 1
totalPassiveConnections: 1
RxRequests: 4
RxResponses: 4
TxRequests: 4
TxResponses: 4
RxMessagesDiscarded: 0
RxMessagesPartialRead: 0
RxMessagesFailed: 0
TxMessageRetries: 0
TxTcpWriteErrors: 0
TxMessagesFailed: 0
ListenerErrors: 0
SessionEstTimeouts: 0
UserMsgsExpired: 0
```

Properties

- totalSessions: The total number of MSRP sessions since the system was last started.
- totalConnections: The total number of connections since the system was last started.
- totalActiveConnections: The total number of connections created by the ME.
- totalPassiveConnections: The total number of connections initiated by MSRP.
- RxRequests: The total number of MSRP request messages received by the ME.
- RxResponses: The total number of MSRP response messages received by the ME.
- TxRequests: The total number of MSRP request messages forwarded by the ME.
- TxResponses: The total number of MSRP response messages forwarded by the ME.
- RxMessagesDiscarded: The total number of MSRP messages discarded by the ME regardless of reason.
- RxMessagesPartialRead: The total number of partial MSRP messages read. If this value is anything but zero, the ME is using partial-forwarding.
- RxMessagesFailed: The total number of MSRP messages the ME has been unable to read.
- TxMessageRetries: The total number of attempts to forward MSRP messages (usually due to slow connection establishment).
- TxTcpWriteErrors: The total number of times the ME encountered an error while attempting to forward an MSRP message.
- TxMessagesFailed: The total number of MSRP messages not forwarded by the ME due to an error condition.
- ListenerErrors: The total number of MSRP listener-related errors.
- SessionEstTimeouts: The total number of times an MSRP session failed to be established.
- UserMsgsExpired: Not currently supported.

Configuring Kernel Filtering

Kernel filter rules provide a security mechanism that allows or denies inbound traffic on ME IP interfaces. The filter controls access to resources on the enterprise servers based on source IP address and/or subnet, source port, and protocol. When the ME processes kernel rules, it first interprets deny rules, then allow rules. In this way, you can deny a subnet access, and then allow specific endpoints.

The ME acts on kernel rules before the other, higher level rules such as DOS policy rules. This stops traffic from known problems early, tying up fewer processing resources.

CLI Session

The following CLI session creates and enables a deny rule named *evil-badguy* from source IP address 215.200.40.8, source port 56, over UDP.

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
config ip boston1>config kernel-filter
config kernel-filter>config deny-rule rule1
Creating 'deny-rule rule1'
config deny-rule evil-badguy>set admin enabled
config deny-rule evil-badguy>set source-address/mask 215.200.40.8/24
config deny-rule evil-badguy>set source-port 56
config deny-rule evil-badguy>set protocol udp
```

Configuring Messaging

Messaging is the mechanism from which the ME system communicates with other systems in the cluster. Messaging sets up a listening socket on an interface, enabling the interface to receive messaging traffic and participate in clustering and media partnering.

In a cluster, the master looks through the configurations of all ME systems to find out which interface is used for messaging. (If multiple interfaces are configured, the master only communicates with one: the first it finds.) The master then communicates with the identified interface to share configuration and data.

In media partnering, you configure a specific IP address (on a different box) as a partner. On the box that owns that IP address, you need to configure and enable messaging for media partnering to operate.

CLI Session

The following CLI session configures messaging on box 1, interface eth0.

```
SIP>config cluster
config cluster>config box 1
config box 1>config interface eth0
config interface eth0>config ip boston1
config ip boston1>config messaging
config messaging>set admin enabled
config messaging>set certificate vsp tls certificate name
config messaging>set port 13002
config messaging>set protocol tls
```

For detailed information on ME clusters and media partnering, refer to the *WebRTC Session Controller Installation Guide*.

23

Enabling ME Services

This chapter describes the services that you can enable on the ME platforms.

Enabling Services on the ME Master

There are administrative services available on the ME master that are enabled by default. These master services are:

- Cluster-Master Services
- Directory Services
- Accounting Services
- Authentication Services
- ME Database
- Registration Services
- Server Load
- Call Failover (Signaling and Media)
- Load-Balancing
- File-Mirror
- Route Server
- Sampling
- Third-Party-Call-Control (3PCC)

If you are not using any of these services, you can globally disable them to conserve memory and system resources on the ME master.

Cluster-Master Services

The **cluster-master** services object configures the ME system that maintains the master configuration for the cluster. The master is responsible for providing configuration changes and updates to other devices in the cluster. If a different device becomes the cluster-master during a failover, this device then sends out its configuration to the other devices in the cluster.

CLI Session

```
NNOS-E> config
config> config master-services
config master-services> config cluster-master
```

```
config cluster-master> set admin enabled
config cluster-master> set host-box cluster box 2
config cluster-master> set host-box cluster box 1
```

Accounting Services

When enabled, accounting services supports RADIUS accounting, system logging (syslog), DIAMETER protocol services, the accounting database, archiving, and the accounting file-system.

You can configure one or more of these accounting mechanisms for capturing the ME network accounting activity and SIP call detail records under the VSP (Virtual System Partition) configuration object.

CLI Session

The following session enables the ME global accounting services on the master.

```
NNOS-E> config master-services
config master-services> config accounting
config accounting> set admin enabled
config accounting> set host-box cluster box 3
config accounting> set host-box cluster box 1
config accounting> set group 1
```

ME Database

The master-services **database** object allows you to configure maintenance and other settings for the ME system database. The database is the local repository for call accounting records and media files.

CLI Session

The following session enables ME database maintenance and sets the local maintenance time at 6 a.m. daily.

```
NNOS-E> config master-services
config master-services> config database
config database> set admin enabled
config database> set maintenance time-of-day 06:00
```

Server Load

The master-services **server-load** object configures the ME to calculate server load. This object must be enabled if your dial plan arbiter settings use least-load as the routing algorithm option. (The arbiter rules property sets the criteria by which the ME selects the server to which it forwards calls.)

CLI Session

The following session enables the server load functionality on the ME master.

```
NNOS-E> config master-services
config master-services> config server-load
config server-load> set admin enabled
config server-load> set host-box "cluster box 2"
config server-load> set host-box "cluster box 3"
```

Call Failover (Signaling and Media)

The master-services **call-failover** object configures failover for both the media and signaling streams across an ME cluster. Enabling **call-failover** ensures that there is an active copy of the database on another box in the cluster in the event of a failure. The first **host-box** property defines the primary ME system. Configure backup boxes in the event of primary failure by re-executing the **host-box** property.

CLI Session

The following session enables call-failover of the media and signaling streams.

```
NNOS-E> config master-services
config master-services> config call-failover
config call-failover> set admin enabled
config call-failover> set host-box cluster box 1
config call-failover> set host-box cluster box 2
```

The call must be connected at the SIP level for signaling failover to succeed. States prior to the "connected" state are not maintained in the cluster-wide state table. For TCP and TLS connections, the user agent (UA) must reestablish the connection after the failover, since TCP and TLS are connection-oriented protocols that do not maintain state information. If TLS is used, the appropriate certificate must be loaded on both devices in the cluster.

Accurate call logs are recorded at the end of the call. However, if the ME system maintaining the call log database fails over to the other ME system in the cluster, call information will not be recorded.

Use the ME **show signaling-sessions** action to view cluster-wide signaling state information.

```
NNOS-E> show signaling-sessions

session-id: 342946641025485482
fromURI: <sip:1234@dial-plan.com>
toURI: <sip:5678@dial-plan.com>
inLegCallID: 3c2a54ca1fbd-7intxouoq8zo@172-30-0-176
inLegFromTag: xqkhmbwmiv
inLegToTag: b432a8c0-13c4-454a1124-102dd42a-164adf67
outLegCallID:
CXC-279-61b29378-b432a8c0-13c4-454a1124-102dd42b-7023adbd@dial-plan.com
outLegFromTag: b432a8c0-13c4-454a1124-102dd42b-749c0b03
outLegToTag: 152jkzyt73
origInFromURI:
origInToURI:
origOutFromURI:
origOutToURI:
vthreadID: 278
initialMethod: 0
Box: 0.0.0.0
```

Load-Balancing

The master-services **load-balancing** object configures the ME systems to host the load-balancing master service. These devices (boxes) are responsible for keeping the rule database up to date. They do not need to be the same devices that host the head-end interfaces, although it is common to do so. (You can, for example, configure devices in the cluster that only serve as host devices without any head-end interfaces or backing interfaces.)

For more information on the load-balancing object, refer to the *WebRTC Session Controller Media Engine Object Reference*. For more information on configuring load-balancing across ME interfaces, see Load Balancing Across ME Interfaces.

CLI Session

The following CLI session enables load balancing on the master, specifies box 1 as the master box on which the rule database runs (subsequent host boxes 2 and 3 serve as backup) and associates the load balancing service with preconfigured VRRP group 1.

```
NNOS-E> config master-services
config master-services> config load balancing
config load-balancing> set host-box cluster box 1
config load-balancing> set host-box cluster box 2
config load-balancing> set host-box cluster box 3
config load-balancing> set group 1
```

Sampling

The master-services **sampling** object opens the mechanism for setting the interval at which the ME samples operational aspects of the system for either:

- Display in the ME Management System, or
- For sending to an IBM Tivoli server.

By setting sampling for a status provider, you can view data for that provider over a specified period of time. The ME supports two sampling targets: a Postgres SQL database and an IBM Tivoli server. (Set the provider data sent to the target using the **status** and **provider** objects. See *WebRTC Session Controller Media Engine Object Reference* for more information on configuring these objects.)

When you execute a status-provider command from the CLI, the system just displays the results of the request at the time it was issued.

Once you have enabled sampling, the master service stores the samples in its local database. You can select a status provider underneath Trends in the Status tab of the ME Management System. The GUI trends graphs pull data from the database on the sampling master service box to display a time series graph of the results. Changes to the interval setting in the sampling subobjects do not effect the CLI results.

> **Note:** If you have limited storage space, and are not using this feature, disable it. Otherwise, polling data is continuously written to the status database.

CLI Session

The following CLI session enables sampling services on the ME master:

```
NNOS-E> config master-services
config master-services> config sampling
config sampling> set admin enabled
config sampling> set host-box cluster box 1
config sampling> set host-box cluster box 2
config sampling> set host-box cluster box 3
config sampling> set group 1
config sampling> return
config master-services> return
config>
```

Enabling Event Logging Services

The ME event logger allows you to configure how event messages are filtered and captured. You can direct event messages to a remote syslog server (by IP address), to a named event log file stored on the ME system, or to the local ME database.

CLI Session

The following session configures the event logger to direct event messages to a remote syslog server.

```
NNOS-E> config services
config services> config event-log
config event-log> config syslog 192.168.124.89
```

The following session configures the event logger to direct event messages to a named file and sets the event log operational parameters: direct all messages to the file, limit the event log file size to 20 Mbytes, and set the maximum number of event log files to create when log files reaches the maximum size in megabytes.

```
NNOS-E> config services
config services> config event-log
config event-log> config file eventfile1
config file eventfile1> set admin enabled
config file eventfile1> set filter all error
config file eventfile1> set size 20
config file eventfile1> set count 5
```

The following session configures the event logger to direct event messages to the local ME database and sets the event log operational parameters: direct only SIP messages to the local database, and set the maximum number of days over which event messages are logged to the local database before the database is cleared and restarted.

```
NNOS-E> config services
config services> config event-log
config event-log> config local-database
config local-database> set admin enabled
config local-database> set filter sip error
config local-database> set history 50
```

Configuring Threshold Monitors

The **services/monitors** configuration object allows you to monitor the following statistics and thresholds for logging and SNMP trap generation:

- CPU usage
- Memory usage
- TLS connections statistics

Polling intervals are in minutes, memory and CPU usage in percent, and TLS connections and failures in actual numbers. At the specified polling interval(s), the ME checks memory and CPU usage, and TLS statistics. If a parameter setting is exceeded, the ME logs an event and an SNMP trap.

CLI Session

```
NNOS-E> config services
config services> config monitors
config monitors> config monitor usage
Creating 'monitor usage'
```

```
config monitor usage> set interval 60
config monitor usage> set parameter cpu-usage 90
config monitor usage> set parameter memory-usage 95
config monitor usage> return
config monitors> config monitor tls
Creating 'monitor tls'
config monitor tls> set interval 30
config monitor tls> set parameter tls-connections 1000
config monitor tls> set parameter tls-failures 10
```

Configuring Data and Archiving Locations

The **services/data-locations** configuration object allows you to specify the directory and path locations on the ME system where you are to save certain types of information. This information includes:

- RTP media (for call recording). Use the **rtp-recorded** property to select a location on the system disk for local archiving of call detail records and call recordings.
- RTP mixed (for playback of recorded calls). Use the **rtp-mixed** property to set the location for playback of recorded calls.
- File transfer. Use the **file-transfer-recorded** property to set the location for file transfer records.
- Log files. Use the **log** property to set the location for log files.

If you choose not to create specific locations for saved files, the ME provides default directory path locations. For example, the directory path */cxc_common* on hard-drive-1 is the default location for recorded RTP files and file transfers. You can display the default directory file paths using the **show** command.

CLI Session

```
config> config services data-locations
config data-locations> show

services
 data-locations
  rtp-recorded[1] /cxc_common/rtp_recorded
  rtp-recorded[2] /cxc/recorded
  rtp-mixed[1] /cxc_common/rtp_mixed
  rtp-mixed[2] /cxc/mixed
  rtp-mixed[3] /cxc/admin/archives
  file-transfer-recorded[1] /cxc_common/ft_recorded
  file-transfer-recorded[2] /cxc/recorded
  log /cxc_common/log
```

The following CLI session changes the default logging path from */cxc_common/log* to */cxc/admin/logfiles*.

```
config> config services data-locations
config data-locations> set log /cxc/admin/logfiles
```

The following CLI session sets the location for "mixed" RTP files to the directory */cxc/admin/RTPmixed;* the location for storing file transfer records is set to */cxc/admin/FTrecords.*

```
config> config services data-locations
config data-locations> set rtp-mixed /cxc/admin/RTPmixed
config data-locations> set file-transfer-recorded /cxc/admin/FTrecords
```

Configuring an External Database

If you want to use a database other than the one that is provided with the ME system, you can configure the ME to use an external database to store event logs, call detail records, and other accounting data. Depending on your network remote SQL server databases, for example, can provide large storage and resource capabilities.

To configure an external database, you will need the Open Database Connectivity (ODBC) driver name associated with the database, as well the user name and secret tags (and password) needed for the ME to access the database. Consult your database administrator for this information before configuring the remote database on the ME system.

The following CLI session configures the database driver named "My SQL Server" and sets the username, secret-tag, and password/password confirmation for this database.

CLI Session

```
config> config services database external
config database external> set driver "My SQL Server"
config database external> set username cxc
config database external> set secret-tag 123
password: *********
 confirm: *********
config database external>
```

The following CLI session configures the event log to direct *snmp* events with the severity *warning* to the SQL database named *corpDatabase* for a period of 150 days. The ME automatically associates the external database name to the services/database configuration.

```
config services> config event-log
config event-log> config external-database corpDatabase
config corpDatabase> set admin enabled
config corpDatabase> set filter snmp warning
config corpDatabase> set history 150
```

For more information on the services/database object, refer to the *Net-Net OS-E – Objects and Properties Reference*.

Setting ME Disk Thresholds

The **storage-device** object allows you to set warning and failure thresholds for remaining disk space on ME hard drives. When a disk drive reaches the configured **fail-threshold** property setting, the ME begins WRITE operations to the next available disk drive. Warning messages are logged (in minutes) whenever disk threshold settings are matched.

The **storage-device** object operates on all installed disk drives. If all disk drives match the configured thresholds, media call recording, file transfers, and log files will no longer be written to the ME disks.

> **Note:** Currently, the ME systems support two 250GB internal disk drives.

The following CLI session sets the fail thresholds for all installed disk drives. Writing to that disk fails when the remaining disk space drops to 20 GB.

CLI Session

```
config> config services
config services> config storage-device
config storage-device> set fail-threshold 20000
```

Scheduling Regularly Performed Tasks

The ME can automatically perform tasks on a configured schedule. This means that you do not have to physically execute an action at a specific time; the ME does it for you. Use the **set action?** command to display the current list of tasks from which you can choose.

The following CLI session configures a directory reset for the *Boston1* enterprise directory at 3 pm.

CLI Session

```
config> config services
config services> config tasks
config tasks> config task directoryReset
config task 1> set action ?
archive         Run the archiving task for a given vsp
 directory-reset Reset an enterprise directory

config task 1> set action directory-reset
config task 1> set schedule time-of-day 15:00:00
```

Performing Database Maintenance

The ME automatically runs a database maintenance script daily, at 3:00 A.M. This "normal" database maintenance purges (removes old files preventing ME disks from becoming too full), "vacuums" (reclaims unused disk space), reindexes, and analyzes the database. You can also selectively schedule periodic database maintenance or force database maintenance at any time.

Along with normal daily database maintenance, Oracle recommends that you perform a "vacuum-full" on the database monthly to reclaim unused disk space.

This section describes how to do the following database maintenance tasks:

- Set normal maintenance time-of-day.
- Schedule periodic database maintenance.
- Force manual database maintenance.
- Perform the database "vacuum-full" process (recommended monthly, in addition to normal maintenance).

Note: As a guideline, Oracle recommends that you perform database archiving more frequently than database maintenance. For example, archiving on a daily basis and performing maintenance every 7-days allows records in the database to age without the risk of removing records before those records are archived. See Enabling and Configuring Local Archiving for information.

Setting Normal Database Maintenance Time-of-Day

The ME automatically runs database maintenance daily, at 3:00 A.M.

If you want to change the actual time-of-day when the ME runs normal database maintenance, use the **master-services** database object. If old records are found, the ME purges those records from the database. Optionally, you can configure a time period in hours, such as every 3 hours, if you want run maintenance at multiple time periods during a 24-hour day.

CLI Session

```
config> config master-services
config> config database
config database> set maintenance time-of-day 02:00:00
```

Verifying Normal Database Maintenance

To verify that normal maintenance has successfully completed, and that a table has been vacuumed automatically, view the system log file. The log file should display a message similar to the one shown below:

```
2008-04-16T05:39:45+12:00[notice] 1:SIP[system] An automatic VACUUM FULL
was performed on database table SipMessage to reclaim 895300 unused
pointers
```

Scheduling Periodic Database Maintenance

The VSP **database** object allows you to configure the number of days to elapse before the ME purges old records from the database. You can selectively configure the number of days for each of the following database records:

- accounting
- call details
- media
- file transfer
- instant messages

Whenever records in the database become older than the configured number of days, the next maintenance natively purges the old files. The following CLI session configures the number of days to elapse for each database record type before the ME deletes the old records from the system disk.

CLI Session

```
config> config vsp
config> config database
config database> set accounting-history 7 days
config database> set call-details 10 days
config database> set media-history 10 days
config database> set file-transfer-history 3 days
config database> set im-history 2 days
```

Forcing Database Maintenance

Use the **database-maintenance normal** command to run a specific database maintenance operation at any time. This forces a database cleanup of any old database

entries if you did not previously configure the VSP database settings. Use the show **database-tables** command to display the database contents after the cleanup.

CLI Session

```
NNOS-E> database-maintenance normal

Starting database maintenance as a background operation.

 -- this may take a very long time --

Please check database-maintenance-status for notification when this
operation is complete.
```

Performing Database Vacuum-Full

The normal (daily) vacuum process attempts to reclaim any unused space in the database (this is analogous to a hard drive defragmentation process, but on the database files) without locking any of the tables.

The **database vacuum-full** process locks each table, one at a time, and reclaims all possible disk space. Note that a table lock prevents the table in question from being written to by an application; i.e. the ME.

Oracle recommends performing a **database vacuum-full** on a monthly basis by scheduling a "maintenance/outage window." You should only run the process during a maintenance window because the process will lock tables, preventing them from being written to by the ME. This can affect the ability of a DOS rule from being triggered, and at the same time, affecting call-logs, recording of accounting data, and any other data that is written to the database. (However, running **database vacuum-full** will not affect the ability of the ME to pass sip/media traffic, accept/delegate registrations, route calls, and perform other directly service-related tasks).

If your site is logging a large volume of data, you may wish to perform a **vacuum-full** on a more frequent (e.g. weekly) basis.

Note the following **database vacuum-full** implementation tips:

- You perform a vacuum-full on the entire database (global) using the **database vacuum-full system** command.
- You vacuum a specific table using the **database vacuum-full system <table-name>** command. For example, you may wish to use this process if the ME logs a message stating that a specific table needs to be vacuumed.

Performing Other Database Maintenance Tasks

You use the VSP **database** object to perform other database maintenance tasks, as described below:

- **delete**: Purges the database of entries contained in the specified database, or entries in the table within the database. The **database delete** action (without qualifiers) deletes all rows in all tables in the database.
- **vacuum**: Based on SQL's VACUUM command, reclaims storage occupied by deleted entries and makes it available for re-use. The system can read and write to the table while the process is occurring (versus more extensive **vacuum-full process** during which the table is not available for read/write operations during the process).

- **drop**: Deletes all data stored in the specified table and removes the table definition from the database schema.
- **repair**: Initiates database repair options. If you select the **data-recovery** option, the system recovers data that was removed by the ME when it corrected a corrupted database. The **translate** option migrates earlier databases to a format compatible with ME Release 3.2 and later.
- **initialize**: Deletes all data and reinitializes the database.
- **snapshot**: Captures an archive of the database at a certain point(s) in time.

Refer to the *Net-Net OS-E – Objects and Properties Reference*for full description of how to use each of these database objects.

Managing Oracle Communications 2600 Database Size

This section describes ways to manage the size of the ME database. That is, it describes ways to reduce the amount of data that is being written to the database. You may wish to try one of these procedures if your database is growing too large, or if it is responding too slowly:

- Disable the logging of REGISTER messages.
- Configure a policy to prevent logging of NOTIFY messages.

> **Note:** Backup the current ME configuration before attempting any of the procedures described in this section.

Disabling REGISTER Message Logging

To disable the logging of REGISTER messages in order to reduce the amount of data store in the database, do the following steps:

1. From the ME Management System, select **vsp->default-session-config-> log-alert**.
2. Set **message-logging** to **no-registers**, then click **Set** to save your changes. This change will take effect immediately.
3. Repeat Step 1 for any **session-configs** that have a **log-alert** configured (e.g. in session-config-pool entries, policies, dial/registration-plans, etc.).

To verify that REGISTER messages are no longer being logged, do the following steps:

1. From the ME Management System, click on the **Call Logs** tab.
2. From the left side of the window, click **SIP Messages**.
3. Click on **Advanced Search**.
4. Enter in **REGISTER** in the **Request Message** field.

   The ME searches for messages of type REGISTER. None should be found.

Preventing NOTIFY Message Logging

If you want to further reduce the amount of data that is being logged to the database, you can configure a policy to prevent logging of specific message types. For example, you may want to prevent the logging of NOTIFY messages that are received from phones (i.e. being received on the public IP interface). These messages are often used as “keep-alive” messages from the end device.

To configure a policy to prevent logging of NOTIFY messages from phones, do the following steps:

1. From the ME Management System, select **vsp->policies**.
2. Scroll to **session-policies**, then click **Add policy**.
3. Name the new policy "default" and then click **Create**.

   If there is already a default-policy configured under **vsp->policies** skip to step 4, keeping in mind that in this example the default-policy is named "default."
4. Specify the "default" policy as your default-policy under **vsp->policies**-> **session-policies.**
5. Under **vsp->policies**-> **session-policies->policy default**, add a new rule. Name the new rule something obvious, for example, "NoLog-NOTIFY."
6. Under vsp->policies-> session-policies->policy default->rule-> NoLog-NOTIFY, configure the condition-list as follows:
   - Set the **default operation** to **AND**
   - Set the **mode** to **evaluate**
7. Under **vsp->policies-> session-policies->policy default->rule-> NoLog-NOTIFY->condition list**, add a **sip-message-condition**, as follows:
   - Set an **attribute** of **request-method**
   - Set the **match** to **match**
   - For the **request-method**, select **NOTIFY** (You could also select other SIP message types.)
8. Under **vsp->policies-> session-policies->policy default->rule-> NoLog-NOTIFY->condition list**, add a **sip-message-condition,** as follows:
   - Set an **attribute** of **local-ip**
   - Set the **match** to **match**
   - For the **local-ip**, enter the IP Address of the ME public interface, including the "slash" subnet mask notation. For example: 1.1.1.1/32.
9. Under **vsp->policies->session-policies->policy default->rule-> NoLog-NOTIFY->condition list**, add a **sip-message-condition**, as follows:
   - Set an **attribute** of **direction**
   - Set the **match** to **match**
   - For the **value**, select **RX**
10. Under **vsp->policies-> session-policies->policy default->rule-> NoLog-NOTIFY**, create a **session-config** container.
11. **Under vsp->policies-> session-policies->policy default->rule-> NoLog-NOTIFY->session-config**, configure a **log-alert** container, then set **message-logging** to **disabled**.

To check to see if the rule is being enforced, perform a "show rules" from the CLI. For example:

```
NNOS-E> show rules
name: policy default/rule NOTIFY
admin: enabled
evaluations: 10008082
```

```
successes: 8336316

NNOS-E> show rules
name: policy default/rule NOTIFY
admin: enabled
evaluations: 10008127
successes: 8336356
```

If the number of "successes" is increasing, then the condition-list "entry criteria" are causing SIP messages to be affected by the rule's session-config.

Backing Up the Database

The **database-backup backup** command allows you to create a backup file of the database, and save it to the ME.

> **Note:** Performing the database backup procedure increases the load on the ME, slowing down the device. Therefore, Oracle recommends performing this task for debugging purposes only.

The database backup file is saved in /cxc/pg_dump/*name*, where *name* is the file name that you specify. When you enter the name for the backup database file, make certain to specify a path that begins with /cxc/pg_dump/.

For example, /cxc/pg_dump/database1 is correct. However, if you specify /cxc/database1, the operation will fail.

Note that by default the ME uses BZIP2 compression. This format is optimized for size, but can take longer to produce. If you would prefer to use GZIP compression, which is faster but results in a 30-40% larger archive, you can do so by supplying the gz suffix when you initiate the action. The following table provides examples of using the gz suffix:

Table 23–1 Archive Types

Enter this filename at the command line	Get an archive of this type
DBbackup	DBbackup.bz2
DBbackup.gz	DBbackup.gz

To create a database backup file and store it on the ME, perform the following steps:

1. Use the **show mounts** command and **shell** command to verify that you have enough storage space on the disk (preferably /mnt/hd2), as shown in the following sample CLI session:

```
NNOS-E> show mounts
device       device-name   mount-point   filesystem    disk-size  percent-free
------       -----------   -----------   ----------    ---------  ------------
cdrom        /dev/cdrom                                0          0
usb          /dev/usb1                                 0          0
hard-drive-1 /dev/root     /             reiserfs      234448     96
hard-drive-2 /dev/sdb                                  0          0
hard-drive-3

NNOS-E> shell
bash-3.00# du -sh /var/lib/pgsql
296M    /var/lib/pgsql
```

```
bash-3.00# exit
exit
```

2. You must create the /cxc/pg_dump directory due to the fact that this procedure is used most often as a debugging tool and is not present during the initial ME installation. Use the **mkdir** command as shown in the following CLI session:

CLI Session

```
NNOS-E> shell
bash-3.00# mkdir pg_dump
bash-3.00# exit
exit
NNOS-E>
```

1. Execute the **database-backup backup** command, specifying a filename for the database backup file.

   For example, the following CLI session creates a database backup file named **DBbackup.bz2** where **system** is the system database where call logs and accounting records are stored:

```
NNOS-E> database-backup backup system /cxc/pg_dump/DBbackup
Are you sure (y or n)?
Starting database backup as a background operation.
 -- this may take a very long time --
Please check database-maintenance-status for notification when this operation is
complete.
```

Restoring a Database

Use the **database-backup restore** command to restore a saved database backup file from the /cxc/pg_dump directory to the ME system.

Any restore action adds entries from that file to the database. If your goal is to overwrite the database, then you should first use the **database delete** action, and then use the **database-backup restore** action.

The following CLI session restores the backup file *backup.bz2.*

CLI Session

```
NNOS-E> database-backup restore system /cxc/pg_dump/backup
Are you sure (y or n)? y
Starting database restore as a background operation.
-- this may take a very long time --
Please check database-maintenance-status for notification when this operation is
complete.
```

Enabling and Configuring Local Archiving

Local archiving allows you to store call accounting records and media files at regular intervals on the ME platform before the records are removed by the database maintenance interval, as described in the previous section. Other archiving options "push" the data to alternate locations.

You can specify the types of information to store with the *include-* properties. If you do not include any of the message types, the archive will contain just the meta data (To, From, setup/connect/disconnect times, and call ID). All message types are included by default.

When archiving, the ME creates both a .zip file and an XML file of the archive contents. The XML file contains all of the XML data for the call except for the SIP messages. The .zip file contains the XML file and an additional file called sip.xml,which contains the SIP messages.

You enable local archiving using the **vsp\accounting\archiving object**. In addition, you must configure a server in one of the archiving sub-objects for the archiving mechanism to work.

- **windows-share**
- **ftp-server**
- **smtp-server**
- **db-server**
- **local**

The following CLI session enables archiving on the ftp server named *ftp1*.

CLI Session

```
NNOS-E> config vsp
config vsp> config accounting
config accounting> config archiving
config archiving> config ftp-server ftp1
config ftp-server ftp1> set admin enabled
config ftp-server ftp1> set username admin
config ftp-server ftp1> set password-tag xyz123abc
password: ************
confirm: ************
config ftp-server ftp1> set directory /archives
config ftp-server ftp1> set server 192.168.10.10
config ftp-server ftp1> set port 1998
config ftp-server ftp1> set timeout 100000
```

To locally archive on a scheduled basis, you need to schedule the archiving task.

```
config> config services
config services> config tasks
config tasks> config task archive
config task archive> set action archive
config task archive> set schedule time-of-day 15:00:00
```

For more information on archiving and archiving to multiple server locations away from the ME system, refer to Chapter 5, "Configuring ME Accounting and Archiving," and the *WebRTC Session Controller Media Engine Object Reference*.

Media Loss Detection

The ME supports media loss detection, which determines when there is a loss of media on a session call-leg. When enabled, all relevant media sessions anchored on the ME are monitored for media activity at a configured interval. When the ME detects a loss of media, an event is generated indicating the session-id, call-leg, media stream index, type of media stream, and the timestamp of the last RTP packet received. An additional event is generated once media has resumed.

Media loss is assumed when there is no change to the Rx packet count during a media monitor interval.

You can either configure media loss detection on a per-session basis or on-demand via the **call-control-media-loss-start** and **call-control-media-loss-stop** actions.

> **Note:** Media streams placed on hold are not monitored for the duration that they are on hold. Monitoring resumes once the session is taken off hold.

Configuring Media Loss Detection

You can configure media loss detection for a session via the **session-config > in-media-loss-detection** and **out-media-loss-detection** properties. You can also use on-demand media loss detection using the **call-control-media-loss-start** and **call-control-media-loss-stop** actions.

> **Note:** The call-control-media-loss-start action takes precedence over the session-config media loss detection configuration.

Configuring Media Detection Loss for a Session-config

The **session-config > in-media-loss-detection** and o**ut-media-loss-detection** objects configure call-leg media loss monitoring. When these properties are enabled for a session, any media session call-legs matching this session-config are monitored for a loss of media.

A session's media must be anchored on the ME for media loss detection to work.

To enable in-leg and out-leg media detection loss:

1. Click the **Configuration** tab and select either **default-session-config** or session-config-pool > entry.
2. Click **Configure** next to **in-media-loss-detection**.
3. **admin:** Set to **enabled** to enable in-leg media loss detection. The default value is **disabled**.
4. **interval:** Set the interval, in seconds, to monitor for a loss of media on the in-leg. This value dictates how quickly the loss-of-media or resumption-of-media is detected and an event is generated. The default interval is **5** seconds.
5. Click **Set**. You are returned to the **session-config** object.
6. Click **Configure** next to **out-media-loss-detection**.
7. **admin:** Set to **enabled** to enable out-leg media loss detection. The default value is **disabled**.
8. **interval:** Set the interval, in seconds, to monitor for a loss of media on the out-leg. This value dictates how quickly the loss-of-media or resumption-of-media is detected and an event is generated. The default interval is **5** seconds.
9. Click **Set**. Update and save the configuration.

Initiating and Terminating On-Demand Media Loss Detection

You can initiate and terminate on-demand media loss detection via the **call-control-media-loss-start** and **call-control-media-loss-stop** actions.

call-control-media-loss-start

Initiates on-demand media loss detection for the call-leg of a specified session.

Syntax

`call-control-media-loss-start <handle> [interval]`

Arguments

- <*handle*>: Specify the call-leg handle on which to start monitoring for a loss of media.
- [*interval*]: Specify the interval, in seconds, to monitor for a loss of media. This value indicates how quickly the loss of media or resumption of media is detected and an event generated. The default value is **5** seconds.

call-control-media-loss-stop

Terminates the on-demand monitoring of a specified call-leg for the loss of media.

Syntax

`call-control-media-loss-stop <handle>`

Arguments:

- <*handle*>: Specify the call-leg handle on which to stop monitoring for a loss of media.

24

Configuring ME Accounting and Archiving

This chapter describes the ME methods for capturing SIP call detail records (CDRs) and other accounting records associated with SIP sessions

Accounting System Overview

The ME system uses industry-standard accounting targets where SIP call detail records are forwarded. The supported accounting targets are:

- RADIUS
- Database
- Syslog
- File system
- DIAMETER
- Archiving

Accounting records are written to directories on the file system, providing a large storage queue for call records as they are written. The accounting software then reads and distributes the call records to the configured accounting target destination(s).

In the event that an accounting target is unable, call records are automatically resent when the accounting target destination(s) become available and when all targets have been updated successfully. Use the **accounting reapply** action to resend call records in the file-system that met the date range to the target regardless if they previously were sent to the target successfully (or not).

The following directory structure store accounting records prior to their distribution to the various accounting targets.

/cxc/accounting/

Subdirectories: #

Files: #-sessionid

Base directory: The root location on the ME system for storing CDRs, such as */cxc/accounting*.

Subdirectories: A series of numbered subdirectories each containing the number of files specified by accounting **subdirectory-size** property. The naming convention is # - *sequential value*.

Files: Each entry is a discrete CDR record. The naming convention is # - *sequential value* followed by the session identifier.

As the accounting software reads and processes files in the subdirectories, it creates, updates and deletes the following status markers:

- **complete**: Indicates that the directory has been fully populated and that all of the files in the directory have been successfully processed.
- **lastprocessed**: Indicates that the directory is currently being populated and that all of the files have been processed successfully.
- **pending**: Indicates that the accounting software has selected the directory for processing and that processing has not yet begun.
- **inprogress**: Indicates the files in the directory are currently being processed.
- **reapply**: Indicates that the directory is currently being evaluated by the **accounting reapply** action.

The **services\data-locations** object contains the **accounting-root-directory** property to specify the directory where accounting records will be placed prior to being sent to the various accounting targets. The default location is the */cxc_common/accounting* directory.

Configuring the Accounting Settings

General accounting settings are available under the **vsp\accounting** configuration object.

- admin: Enables or disables all configured accounting targets.
- **retention-period**: Specifies how many days the accounting records should be retained before being purged from the file system. The default setting is 7. The range is 0 to 21 days.
- **subdirectory-size**: Specifies the number of records to be recorded in each of the sub-directories. The default is 1000. The range is 100 to 2000.
- **purge-criteria**: Specifies he criteria to be used when deleting records from the file system. The **purge-always** setting indicated that records should be deleted even if they have not been saved to all of the defined enabled targets. The **purge-only-when-complete** setting indicates that even expired CDRs should be retained if they have not been sent to all of the defined targets.
- **report**: Creates a named CDR summary report containing the specified field, match, and category criteria.

The **accounting purge** action forces an immediate purge and clears all CDRs on the file system that are eligible for deletion.

The **accounting reapply** action accepts a date range and selected groups and marks qualifying records on the file system back to an unprocessed state. The records are picked up and reapplied (resubmitted) to the configured accounting targets. Use this action if CDR data is lost for a selected target and the data needs to be recovered. This action is limited to data within the current retention period.

The **show accounting-status** command provides a summary of current accounting and processing information for existing targets, including any target exceptions.

Configuring RADIUS Groups

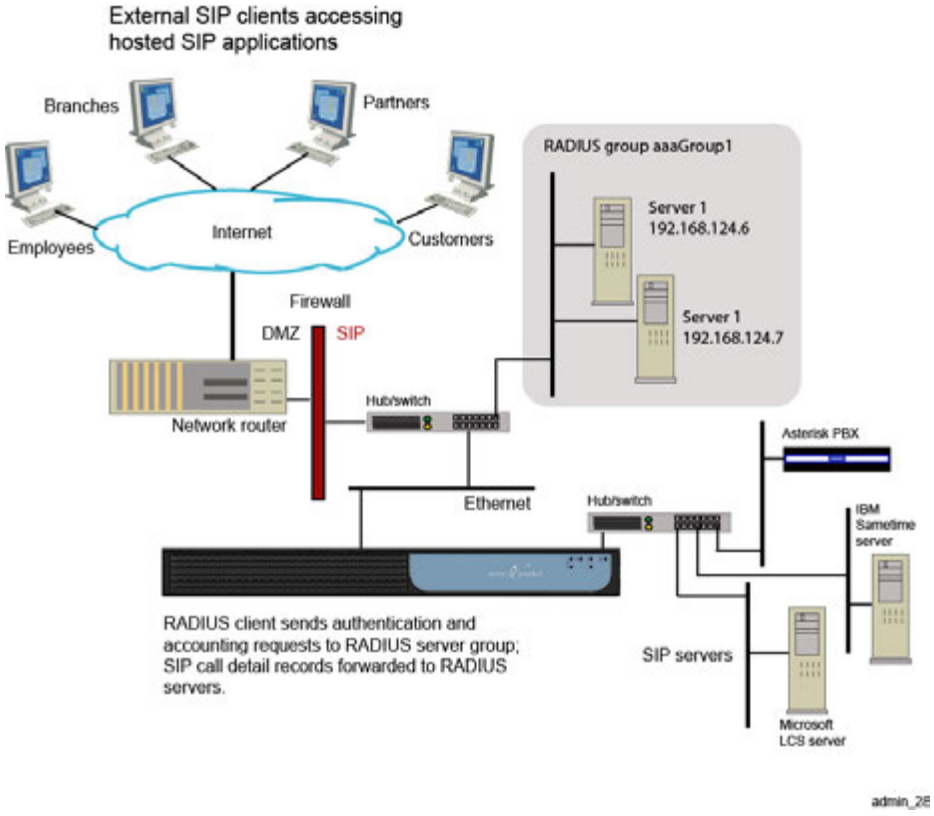
The Remote Authentication Dial In User Service (RADIUS) implementation allows the ME system to operate as a RADIUS client that directs SIP call detail records to a RADIUS accounting server. The RADIUS accounting server receives the accounting

request and returns a response to the client indicating that it has successfully received the request.

A RADIUS group is a uniquely named object that defines the authentication and accounting services associated with a group of RADIUS servers. Including a RADIUS group in one or more VSP configurations allows the ME system (the RADIUS client) to perform user authentication and forward accounting and SIP call detail records to RADIUS servers.This means that you have flexibility to create as many unique RADIUS groups as you need, and include them with the VSPs of your choice.

Within a RADIUS group, you set the RADIUS authentication and accounting modes that you are using, the type of RADIUS accounting format, and whether the RADIUS group is to be included as a default authentication and accounting group for SIP traffic that is not governed by configured authentication and accounting policies.

The following image illustrates a sample network using a RADIUS accounting group.



CLI Session

The following CLI session creates the RADIUS accounting group named **aaaGroup1** and sets the group operational properties.

```
NNOS-E> config vsp
config vsp> config radius-group aaaGroup1
Creating 'radius-group aaaGroup1'
config radius-group aaaGroup1> set admin enabled
config radius-group aaaGroup1> set accounting-mode duplicate
config radius-group aaaGroup1> set authentication-mode failover 3
config radius-group aaaGroup1> set type Cisco
```

In this session, the authentication and accounting modes are RADIUS operational algorithms. The duplicate algorithm issues multiple duplicate accounting requests to

all servers in the RADIUS accounting group. A duplicate accounting request uses the same client source IP address and source UDP port. If you configure multiple authentication servers in the RADIUS group, the failover algorithm forwards authentication requests to secondary servers should the current authentication session fail. You can specify up to 256 failover attempts to other servers.

The default accounting method is cisco accounting, and the aaaGroup1 RADIUS group is a default group for all non-policy governed RADIUS requests between the ME system and the RADIUS servers.

Configuring the RADIUS Servers

You can configure multiple RADIUS servers in the RADIUS group, and you identify each server using a unique number and IP address, authentication port, accounting port, and other operational settings.

CLI Session

The following CLI session creates two numbered RADIUS servers and sets the operational properties for RADIUS requests and responses between the ME system and the RADIUS servers.

```
NNOS-E> config vsp
config vsp> config radius-group aaaGroup1
config radius-group aaaGroup1> config server 192.168.147.6
config server 192.168.147.6> set admin enabled
config server 192.168.147.6> set authentication-port 1800
config server 192.168.147.6> set accounting-port 1801
config server 192.168.147.6> set secret-tag abc123xyz
config server 192.168.147.6> set timeout 1500
config server 192.168.147.6> set retries 3
config server 192.168.147.6> set window 255
config server 192.168.147.6> set priority 2
config server 192.168.147.6> return

config vsp> config radius-group aaaGroup1
config radius-group aaaGroup1> config server 192.168.147.7
config server 192.168.147.7> set admin enabled
config server 192.168.147.7> set authentication-port 1800
config server 192.168.147.7> set accounting-port 1801
config server 192.168.147.7> set secret-tag abcXYZ123
config server 192.168.147.7> set timeout 1500
config server 192.168.147.7> set retries 3
config server 192.168.147.7> set window 255
config server 192.168.147.7> set priority 2
config server 192.168.147.7> return
```

For additional information on configuring RADIUS groups and servers, refer to the *Net-Net OS-E – Objects and Properties Reference*.

Including the RADIUS Group

When you configure RADIUS groups, you include one or more groups with the VSP RADIUS accounting configuration. This tells the VSP what RADIUS servers to use when forwarding RADIUS accounting requests.

CLI Session

The following CLI session includes the RADIUS groups named **aaaGroup1** and **aaaGroup2** with the VSP RADIUS accounting configuration.

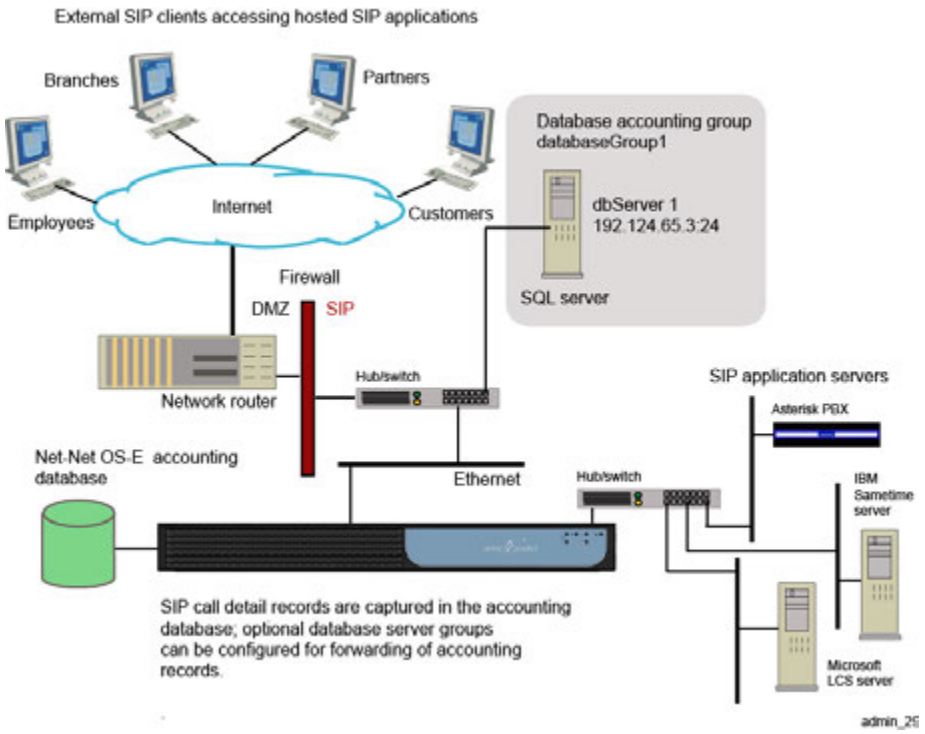
```
NNOS-E> config vsp
config vsp> config accounting
config accounting> config radius
config radius> set admin enabled
config radius> set group vsp radius-group aaaGroup1
config radius> set group vsp radius-group aaaGroup2
config radius> show

vsp
 accounting
  radius
   admin enabled
   group vsp\radius-group aaaGroup1
   group vsp\radius-group aaaGroup2
```

When using the set group command, specify the CLI path where you created the Radius group.

Configuring the Accounting Database

The ME accounting database is a subsystem that captures and stores SIP call detail records. If configured, these records can be forwarded to remote SQL database servers such as Oracle and Postgres where the call detail records are used with other accounting and billing applications. Access to a remote database group and server is restricted by configured user names and passwords.

Accounting policies direct SIP call detail records to specific accounting groups and servers. If you do not configure one or more remote database groups and servers, the SIP call detail records are stored in the ME accounting database only. The following image illustrates a sample network with a database server group.



CLI Session

The following CLI session creates the accounting database group named **databaseGroup1,** creates the associated server named **dbServer1**, and sets the group and server operating properties.

```
NNOS-E> config vsp
config vsp> config accounting
config accounting> config database
config accounting> set admin enabled
config database> config group databaseGroup1
Creating 'group databaseGroup1'
config group databaseGroup1> set admin enabled
config group databaseGroup1> set mode duplicate

config group databaseGroup1> config server dbServer1
Creating 'server dbServer1'
config group databaseGroup1> set admin enabled
config group databaseGroup1> set type sqlserver 192.124.65.3 24 srvr1
config group databaseGroup1> set username frank
config group databaseGroup1> set password-tag kj3k2
```

In this session, the duplicate mode algorithm issues a duplicate accounting request to all servers in the accounting group. A duplicate accounting request uses the same client source IP address and source UDP port. If you configure multiple database servers in the database group, the fail-over algorithm forwards one accounting request to each secondary servers should the current session fail.

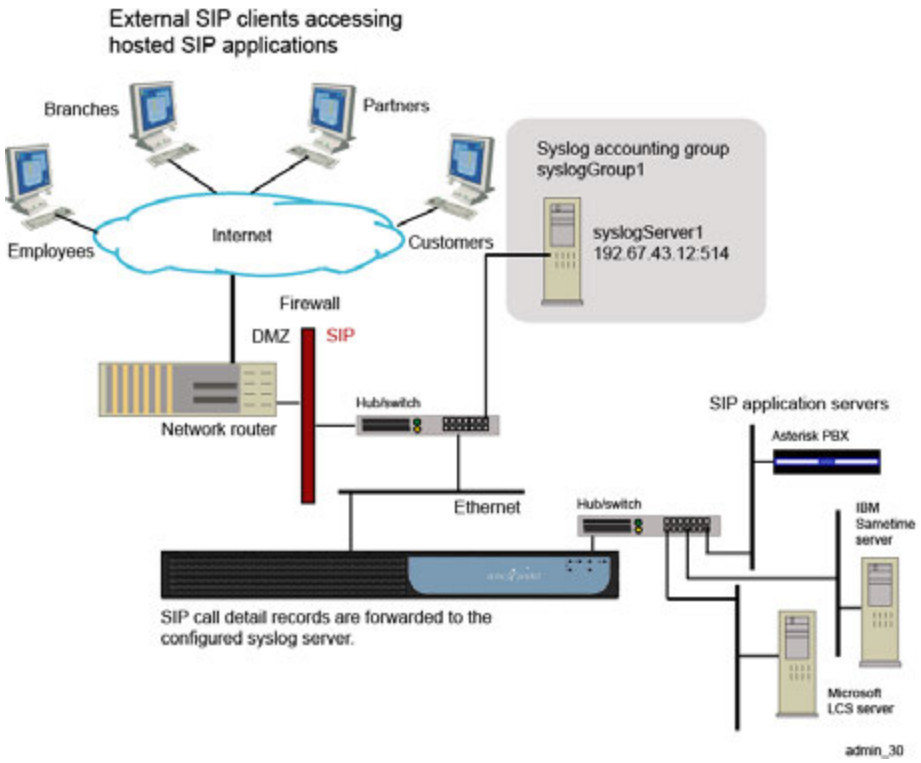
The **databaseGroup1** accounting group is a default group for all non-policy governed accounting database requests between the ME system and the database servers.

> **Note:** If you set the server type to local while using the local database as the accounting target, set the username and the password-tag to postgres. If you edit the username and password-tag properties to anything other than postgres, data will not be written to the database.

For additional information on configuring accounting database groups and servers, refer to the *WebRTC Session Controller Media Engine Object Reference*.

Configuring Syslog

Syslog allows you to log accounting information to a remote server using the configured syslog format: Oracle, CSV, tabular, or XML format. When enabled, SIP call detail records are forwarded to the specified syslog accounting group and server. The following image illustrates a sample network.



CLI Session

The following CLI session creates the syslog accounting group named **syslogGroup1,** specifies the associated syslog server at **192.167.43.12** on port **514**, and sets the syslog group and server operating properties.

```
NNOS-E> config vsp
config vsp> config accounting
config accounting> config syslog
config syslog> set admin enabled
config syslog> config group syslogGroup1
Creating 'group syslogGroup1'
config group syslogGroup1> set admin enabled
config group syslogGroup1> set format csv

config group syslogGroup1> config server 192.167.43.12:514
Creating 'server 192.167.43.12:514'
config server 192.167.43.12:514> set admin enabled
config server 192.167.43.12:514> set name syslogserver1
config server 192.167.43.12:514> set facility local0
config server 192.167.43.12:514> set priority info
config server 192.167.43.12:514> set include-timestamp true
```

In this session, **syslogGroup1** uses Comma-Separated Values (CSV) format. CSV format is a generic file format used for importing data into databases or spreadsheets, such as Microsoft Access or Excel (or several other database systems). CSV uses the .CSV file extension. The **syslogGroup1** accounting group is a default group for all non-policy governed accounting database requests between the ME and the syslog servers.

The syslog server at IP address and port **192.67.43.12:514** is enabled with the operator-defined name **syslogserver1**. The facility (local0 to local7) specifies where SIP call detail records are logged. Syslog facilities help isolate the origin of messages written to the syslog server. The syslog priority (info, emergency, alert, etc.) sets the

message priority to be associated SIP call detail records. All ME accounting and SIP call detail records are assigned this priority before they are forwarded to the syslog server. A time stamp can also be applied to each accounting record.

For additional information on configuring accounting database groups and servers, refer to the *WebRTC Session Controller Media Engine Object Reference*.

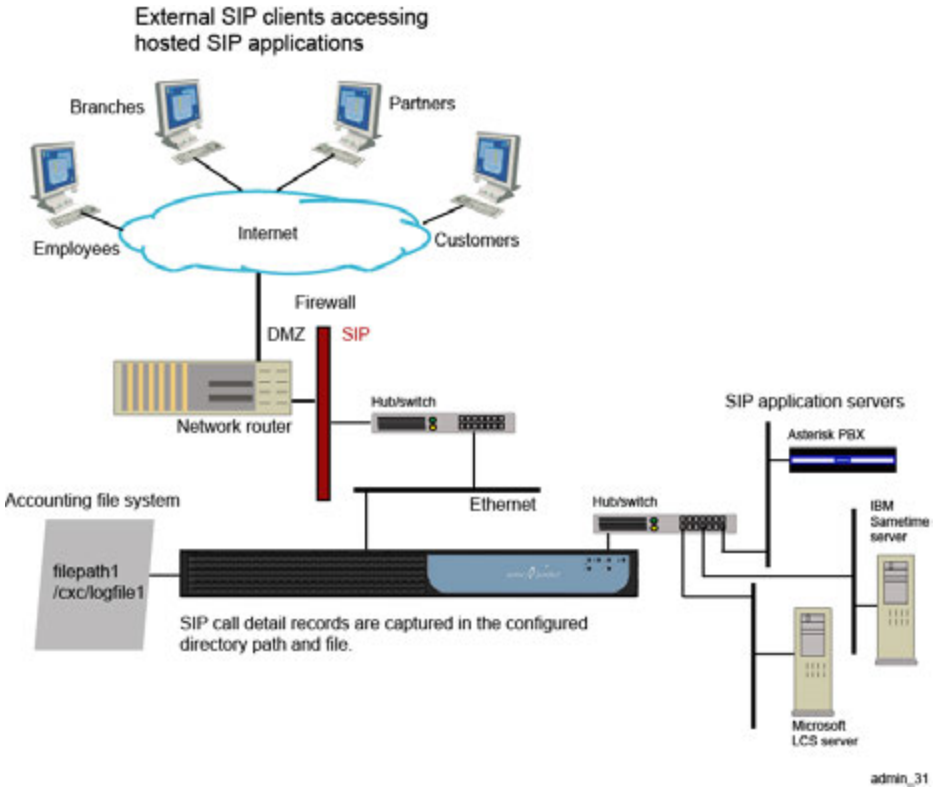
Configuring the File System

The accounting file system allows you to direct SIP call detail records to a named directory path and file using a specified format: CSV, tabular., Oracle text file format, or to a temporary output file in the case of postgres format.

There are two states that the file system cycles through as it processes raw CDRs and writes to the output file.

- Clear: The target is ready to write.
- Writing: The target is writing to the output file.

The following image illustrates a sample network.



CLI Session

The following CLI session creates the file system group named **filePath1,** specifies the format, file path, and target file name, and sets the file system operational properties.

```
NNOS-E> config vsp
config vsp> config accounting
config accounting> config file-system
config file-system> set admin enabled
config file-system> config path filePath1
```

```
Creating 'path filePath1'
config path> set admin enabled
config path> set format csv
config path> set call-field-filter recorded
config path> set file-path \cxc\logfile1.csv
config path> set roll-over never
config path> set purge-old-logs true
config path> set retention-period 1 days
```

In this session, **filePath1** uses Comma-Separated Values (CSV) format. CSV format is a generic file format used for importing data into databases or spreadsheets, such as Microsoft Access or Excel (or several other database systems). CSV uses the .CSV file extension. The ME target file path is **\cxc\logfile1.csv**, where **logfile1.csv** is the name of the file to which SIP call detail records are forwarded.

The **roll-over** property maintains and keeps the original time as it was first applied to the log file. The log file will continue to build under this time stamp. The **filePath1** file system accounting group is a default target group for capturing all non-policy governed SIP call detail records.

For additional information on configuring accounting database groups and servers, refer to the *WebRTC Session Controller Media Engine Object Reference*.

Configuring an External File System Target

The external-file-system target allows you to send accounting records from the ME to a remote system. The target is able to read raw CDRs and write this information to a temporary output file in the format you specify during configuration.

There are four states that the external target cycles through as it processes raw CDRs, writes to the output file, and sends it to the remote system.

- Clear: The target is ready to write.
- Writing: The target is currently writing to the temporary file.
- Sending: The target is sending a file. At this time, the file can also be writing to a temporary file that will become the next file to send once the current file is successfully sent.
- Blocked: The target has one file in the middle of sending and another one ready to send. The target will not process anymore requests from the accounting server, but will send retries to the server giving retry interval based on its best estimate of when the retry can work.

If the configuration is modified or deleted, any files currently being processed are sent immediately and without retries. If the target is in the blocked state, there are two files immediately sent and if the target is in the sending or writing states, one file is sent. The modification or deleted is applied only after the send completes, successfully or not.

If there is a failure when sending a file to the external target, the send is retried every 30 seconds for an hour. After an hour, the send is retried once every hour until it succeeds.

The following is the format of the output file:

```
<target-name>-<yyyy-mm-dd-hh-mm>-<processingtype>-<seq-no>.<xtn>
```

- target-name: Name specified in the configuration.
- yyyy-mm-dd-hh-mm: The timestamp when the output file is created.
- processingtype: Hourly, daily, never.

- xtn: .csv, .tab, .cov, or .pg

CLI Session

The following CLI session creates the external file system target, sets the target format, URL address, and CDR processing.

```
NNOS-E>config vsp
config vsp>config accounting
config accounting>config external-file-system
config external-file-system>config url 7
Creating 'url test'
config url 7>
config url test>set admin enabled
config url test>set format csv
config url test>set url ftp://lalenchery:BillGates#1@10.33.5.10:/acct/test/
config url test>set cdr-processing batch 10
config url test>
```

For additional information on configuring external file system targets, refer to the *WebRTC Session Controller Media Engine Object Reference*.

Configuring Diameter

The Diameter protocol, as described in RFC 3588, provides Authentication, Authorization and Accounting (AAA) services for applications such as IP mobility and SIP multimedia communications sessions. An ME system (SIP proxy), operating as Diameter client, sends an accounting request to the Diameter server where the Diameter server returns an accounting response to the Diameter client indicating that it has received and processed the accounting request.

Diameter is also an essential component for the Oracle route-server functionality.

Creating the Diameter Accounting Group

Like RADIUS, a Diameter group is a uniquely named object that defines the authentication and accounting services associated with a group of Diameter servers. Including a Diameter group in one or more VSP configurations allows the ME system (the Diameter client) to perform user authentication and forward SIP call detail records to Diameter servers.This means that you have flexibility to create as many unique Diameter groups as you need, and include them with the VSPs of your choice.

CLI Session

The following CLI session creates the Diameter accounting group named **diameterGroup1** and sets the group operational properties.

```
NNOS-E> config vsp
config vsp> config diameter-group 1
Creating 'diameter-group 1'
config diameterGroup1> set admin enabled
config diameterGroup1> set authentication-mode round-robin
config diameterGroup1> set application sip
config diameterGroup1> set origin-host text
config diameterGroup1> set origin-realm text
config diameterGroup1> set default-destination-realm text
```

In this session, the **authentication-mode**, sets the Diameter group authentication operational algorithm. This example allows continued authentication requests to primary and secondary servers until a valid authentication response is received (round-robin).

The **application** setting specifies the target application for the servers in this Diameter group. Choose **SIP** for standard AAA activities, **3GPPRx** for inter-operation with the Camiant policy server (enabled with the Rx object), and **Routing** for least-cost-routing between clusters.

The **origin-host** specifies the text written to the Origin-Host attribute field in any Diameter *requests* it sends. This should be the ME domain name.

The **origin-realm** specifies the text written to the Origin-Realm attribute field in any Diameter requests it sends. This should be the ME domain name.

The **default-destination-realm** specifies the text written to the Destination-Realm attribute field in any Diameter responses it sends. This setting operates with the 3Gpp Rx application.

Configuring Diameter Servers

You can configure multiple Diameter servers in the Diameter group, and you identify each server using a unique name, authentication port, and other operational settings.

CLI session

The following CLI session creates two numbered Diameter servers and sets the operational properties for Diameter requests and responses between the ME system and the Diameter peers.

```
NNOS-E> config vsp
config vsp> config diameter-group 1
Creating 'diameter-group 1'
config diameterGroup1> set admin enabled
config group diameterGroup1> config server diameterServer1
Creating 'server diameterServer1>
config diameterServer 1> set admin enabled
config diameterServer 1> set port 3868
config diameterServer 1> set transport tcp
config diameterServer 1> set authentication-port 3868
config diameterServer 1> set request-timeout 2
config diameterServer 1> set window 8
config diameterServer 1> set priority 1

NNOS-E> config vsp
config vsp> config diameter-group 1
Creating 'diameter-group 1'
config diameterGroup1> set admin enabled
config group diameterGroup1> config server diameterServer2
Creating 'server diameterServer2>
config diameterServer 2> set admin enabled
config diameterServer 2> set port 3868
config diameterServer 2> set transport tcp
config diameterServer 2> set authentication-port 3868
config diameterServer 2> set request-timeout 2
config diameterServer 2> set window 8
config diameterServer 2> set priority 1
```

For additional information on configuring Diameter groups and servers, refer to the *WebRTC Session Controller Media Engine Object Reference*.

Configuring Diameter Interfaces and Ports

The **diameter** configuration object under the **box\interface\ip object** identifies the IP interface on which the Diameter server application resides. This is the ME interface

that listens for incoming Diameter connections. This interface must be configured on each ME domain that is referenced by a server in a Diameter group.

CLI Session

```
config box> config interface eth3
config interface eth3> config ip A

config ip A> config diameter
config diameter> set admin enabled
config diameter> set origin-host text
config diameter> set origin-realm text

config diameter> config port 3868
Creating 'port 3868'
config port 3868> set admin enabled
config port 3868> set transport tcp
config port 3868> set application sip
config port 3868> set peer-access-control transport
config port 3868> set peer ipaddress
```

The **origin-host** setting specifies the text written to the Origin-Host attribute field in any Diameter *responses* it sends. This should be the DNS name of the ME domain you are configuring.

The **origin-realm** specifies the text written to the Origin-Realm attribute field in any Diameter *responses* it sends. This should be the ME domain name.

The **port** configuration specifies properties for incoming Diameter connections. The **application** setting sets the application that the incoming connection must be running to use this port.

Choose **SIP** for standard AAA activities, **3GPPRx** for inter-operation with the Camiant policy server (enabled with the Rx object), and **Routing** for least-cost-routing between clusters.

The **peer-access-control** setting specifies how the ME controls incoming peer connections. You can select to allow incoming connection from all peers or from peers on a configured list based on address or Host-IP-Address AVP.

The **peer** setting specifies the list of peers that are allowed to connect to this port. This property is not applied if the peer-access-control property is set to **none. Indicate the peer by specifying the peer IP address.**

Configuring Archiving

The **accounting/archiving** object allows you to configure an archiving location for SIP call detail records. Archiving is the persistent storage of the contents of the call (as opposed to the database or syslog server, which just records the placement of the call).

You must configure an archiving server in one of the archiving sub-objects for the archiving mechanism to work:

- **windows-share**: Archiving of accounting and SIP call records to a selected Windows server partition
- **ftp-server**: Archiving of accounting and SIP call records to a selected FTP server
- **http-server**: Archiving of accounting and SIP call records to a selected HTTP server

- **smtp-server**: Enables archiving of accounting and SIP call records to a selected Simple Mail Transfer Protocol (SMTP) server. When enabled, the ME sends out the archives in the form of an email attachment to the specified destination mailbox.
- **db-server**: Archiving of accounting and SIP call records to a selected database server
- **local**: Archiving of accounting and SIP call records to a location on the ME system

The following CLI session configures a remote database server for archiving of SIP call detail records.

CLI Session

```
NNOS-E> config vsp
config vsp> config accounting
config accounting> config archiving
config archiving> config db-server database1
Creating 'db-server database1'
config db-server database1> set admin enabled
config db-server database1> set username admin
config db-server database1> set password-tag xyz123abc
config db-server database1> set server 192.168.10.10
config db-server database1> set url www.companyABC.com
config db-server database1> set driver-class com.oracle.jdbc.Driver
```

If you are archiving using the **http-server** method, a server-side script designed to be run with Apache 2.0 and perl 5.8.5 on Linux is needed to handle the POST requests that are sent from the ME to transfer the archive zip files to the server. The following is an example:

```
#!/usr/bin/perl
#---Modify the above line to match the location of perl on your system---

#---This script has been tested running with OS-E software version 3.5.2 sending
#to Apache 2.0.52 running on Redhat EL4 Linux with perl 5.8.5---

#---Make sure to modify file permissions for this script so that it can
#be executed by the user running the httpd daemon.---

#---Note this script is provided as an example, which makes no attempt to validate
#the values pulled from the HTTP POST to ensure execution security---

#---Require strict syntax---
use strict;
use warnings;

#---Use the CGI library provided with perl - CGI.pm---
use CGI;
#---The below lines are an example of code, provided as-is, used to take
#the multipart/form-data from an HTTP POST to this script, which
#apache presents on STDIN and write it out to the disk in the
#directory specified in the variable above, using the same filename
#presented in the HTTP POST---

#---Instantiate CGI object---
my $cgi = new CGI;
my %params = $cgi->Vars;

#---Get proper filehandle from unknown file param name---
my $filehandle;
my $anon_param;
```

```
foreach my $param (keys %params) {
$anon_param = "$params{$param}" if (("$param" ne "name") && ("$param" ne "path"))
};

$filehandle = $cgi->param($anon_param);

#---Pull target directory from "path" cgi variable; this comes from the
"directory"
#in the OS-E config. Note: leave off the trailing slash------

#---Make sure to modify file permissions for target directory so that it can
#be executed and written to by the user running the httpd daemon.---
my $dir = $cgi->param('path');
#---Pull target filename from "name" cgi variable
#---Assemble directory and filename---
my $name = $cgi->param('name');
my $fullname = "$dir/$name";

#---Write out the file from the HTTP POST---
open(LOCAL, ">$fullname") or die $!;
binmode LOCAL;
while(<$filehandle>) { print LOCAL $_; }
close(LOCAL);

#---Needed for 200OK response---
print $cgi->header( "text/plain" ), "File received.";
```

The following example displays the way the ME must be configured for the http-server archiving to work:

```
config archiving> config http-server server1
config http-server server1> set admin enabled
config http-server server1> set directory /tmp/archives
config http-server server1> set url http://10.0.0.1/cgi-bin/archive_http_upload_
example.pl
config http-server server1> set timeout 60000
```

- The server needs to be configured to allow CGI scripts.
- The script needs to be placed in the “cgi-bin” directory and given execute permission for the user running the server.
- The URL needs to include the name of the script.
- The directory needs to have “write” permissions for the user running Apache. This argument gets passed through the HTTP POST to the scripted. It is used to determine to which directory on the server the archive file is written.

For additional information on archiving accounting records, refer to the *Net-Net OS-E – Objects and Properties Reference*.

The ME also supports archiving as an accounting target, configured under the **accounting** object. Archiving targets can be configured as either **archive-local** or **archive-external**.

Once the archiving functionality is enabled and configured on the ME, the archiver listens for requests from the accounting server. A request from the server tells the archiver that there are calls that needs to be archived. The archiver creates a task for each CDR. This task gathers data to put in the archive by executing actions and status requests and querying databases.

The archiving target cycles through two states:

- Clear: The target is ready to handle requests.

- Blocked: The target has reached the maximum number of files it can save. You must remove saved archives to enable the target to start processing again.

When the ME sends an archive to a remote location and the send fails, the ME retries sending the archive as many times as it is configured to do so. If all retries fail, the ME saves the archive in the archive-save-folder and logs a message similar to the following.

```
Warning: "Target archive-test, saved 1234.zip containing records 1000 to 1000 as
/cxc_common/archive/saved/1234.zip (failure was: Connect timed out)"
```

You can configure the number of archives that can be saved in the archive-save-folder via the **max-saved-on-send-failure** property under the archive-external and archive-local objects. Once the ME hits this threshold, the target enters the "Blocked" state and stops processing any more CDRs until the saved archives are removed from the folder. When this condition is reached, the ME logs a message similar to the following:

```
Critical: "Target archive-test cannot process any more CDRs because the maximum of
200 archives that can be saved locally on failure is met or exceeded. Delete saved
archives to enable further processing.
```

Note that the number of saved archives may be slightly higher than the configured number. This is because archives are not created in order and it is possible that some newer CDRs finished processing earlier than the archive that finally blocked the target.

Due to accounting server purges, there may be missing CDRs. The ME handles missing records by skipping over them and continuing the process. Missing records are logged and can be viewed in the status provider.

During an HA failover, the target on the new master ME picks up from where the previous master ME left off.

You configure the archiving targets under the **vsp > accounting** object.

```
vsp
 accounting
  admin enabled
  duration-type default
  retention-period 0 days 00:01:00
  subdirectory-size 100 records
  purge-criteria purge-always
  radius
  database
  syslog
  file-system
  external-file-system
  archiving
  purge-check-interval 0 days 01:00:00
  purge-disk-utilization-percent 90 %
  archive-local
  archive-external
archive-worker-threads automatic
  archive-max-inprogress 120
  archive-tries 2
  archive-name-format[1] recordID
  compatible-archives false
  server-idle-timeout 300
```

For more information on the new archiving configuration properties, see the *WebRTC Session Controller Media Engine Object Reference*.

The target can then be applied to a session-config via the **session-config > accounting** object.

```
config vsp>config default-session-config
config default-session-config>config accounting
config accounting>set target archive-external-file-system
"vsp\accounting\archive-external\url""archivetest""
```

You can view information regarding archive targets using the following status providers.

The **show accounting-targets** action is a previously existing status provider that displays summary data from all accounting targets. This status action now includes archiving targets.

```
NNOS-E>show accounting-targets

                  type: archive-external
                  name: url archive-day1
              received: 641 CDRs
             processed: 641 CDRs
              failures: 0
        missing-records: 0
      last-acked-record: 1495276
   acked-pending-record: 1495276
average-processing-time: 2278 milliseconds/CDR
```

The **show accounting-targets-archive-tasks** action displays information about currently running archiving tasks on the ME.

```
NNOS-E>show accounting-targets-archive-tasks

name record               errors     in-progress
------                 ------     -----------
nnose-backup                        1170995             2          (send)
nnose-backup                       1171000             2          (send)
nnose-backup                       1171001             2          (send)
```

For more information on these status providers, see the *Webrtc Session Controller Media Engine Object Reference*.

Free-Form Accounting for CDRs

The ME supports free-form accounting for CDRs, meaning you have the ability to completely customize the list of columns created in CDRs by using the session-config's named variable table. These custom CDRs are supported for all accounting target types except internal database.

You still have the ability to use the pre-existing (default) accounting record columns. This is the ME's default behavior. Each target type supports both forms of accounting, but each individual target can have only one or the other. A target can have either the default accounting fields or custom accounting fields.

This feature differs from the existing CDR custom data fields because you create all of the columns yourself. In releases previous to 3.6.0m5, you could only get existing fields and filter those that you did not want. There also existed one column named **custom-field** that contained user-specified data.

To enable free-form accounting for CDRs:

1. Select the **Configuration** tab and click the **vsp > accounting** object.
2. Click **Configure** next to the type of target for which you want to create free-form accounting.
3. **admin:** Set to **enabled** and provide either a **group** or a **path** for the target (depending on which type of target you are configuring).

4. **custom-accounting**: Set to **enabled**.

> **Note:** The **custom-accounting** property overrides the **call-field-filter** property, where you configure the default accounting records.

5. Click **Set**. Update and save the configuration.

To create free-form CDRs, one mechanism to populate free-form CDRs is to use named variables in the session-config. Named variables can be added to sessions on the ME in multiple ways. They can be added via the **session-config > named-variables** object. For more information on configuring named-variables in the session-config, see Configuring Session Configuration Objects in the *WebRTC Session Controller Media Engine Object Reference*.

Named-variables can also be added via the **named-variable-add** action. For more information on this action, see the Named Variable Actions section of this guide.

The ME offers a list of pre-defined variables for you to use in free-form CDRs. These can be broken down into three types: **acct**, **cdr**, and **session**.

The **acct** class of named variables is derived from items that are already available through the **accounting-data > custom-data-field**.

For a complete list of named variables available on the ME, see the *WebRTC Session Controller Media Engine Objects and Properties Reference* guide.

Once you have the named-variables configured in the session-config, you can add them to your free-form CDRs. You can add named-variables to free-form CDRs via the **accounting > targets > named-variable-entries** property.

To add named-variables to free-form CDRs:

1. Access the *vsp* > *accounting* > <*target*> **> named-variable-entries** object where you have the **custom-accounting** property set to **enabled**. Click **Configure** next to **named-variable-entries**.

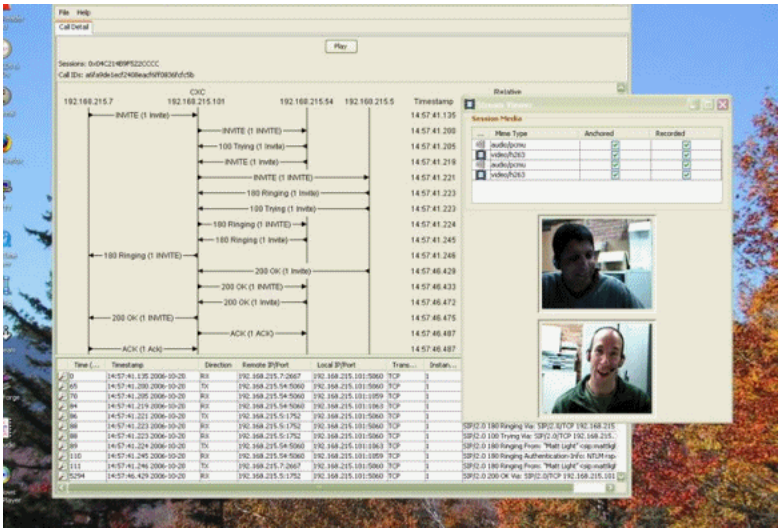
   Or, access the **vsp > session-config-pool > entry > accounting-data** object and click **Add named-variable-entry**.

2. Click **Add entry**.
3. From the **variable-name** drop-down list, select the named-variable to include.
4. **display-name**: Enter the name you want to be displayed for this column. This value is required if the accounting target is a database and the display-name complies with the column name rule of the corresponding database.
5. Click **Create**. Repeat this process for as many named-variables as you want to include.
6. Click **Set**. Update and save the configuration.

Using the ME Archive Viewer

The archive viewer is a standalone utility that displays information and plays video recordings from archive files that have been stored locally on a client PC. The viewer allows you to see the call diagram and message details without having to run the ME Management System.

The following image illustrates a sample Archive Viewer display.



The Archive Viewer is contained in a ZIP file included with the ME release software. The file is named **nnSE360-av.zi**p.

Perform the following steps on a Windows PC, which is the only supported platform for the Archive Viewer:

1. Download the **nnSE360-av.zi**p file to a location on your PC. The file is available from the Oracle support site.
2. Double-click the .ZIP file, then select **Extract All**. A separate folder will be created using the same name, minus the .ZIP extension.
3. Open the folder that you just created, then double-click the **nnSE360-av.exe** file. This will launch the Archive Viewer.
4. Select **File->Open Archive**, or **File->Stream Viewer** to browse for the archived file. The Stream Viewer replays and mixes the two audio streams (one in each direction) with the video streams (one in each direction).

Note: You must configure the ME with both accounting and media recording enabled. You can enable archiving to periodically send the recorded files to a workstation, or you can create individual session archives on demand from the ME Management System Call Logs screen.

Call Detail Record Field Descriptions and Data Types

The following table lists and describes the fields and data types that make up detail record.

Table 24–1 CDR Field Descriptions and Data Types

CDR Field	MS-SQL Data Types	PostgreSQL Data Types	Oracle Data Types	Description
ani	type="String"	type="name"	**VARCHAR2 (256)**	The caller ID for the ANI after any manipulation is done by the ME.
call-dest-cr-name	type="String"	type="name"	**VARCHAR2 (256)**	The name of the dial plan that forwarded the call.
call-dest-realm-name	type="String"	type="name"	**VARCHAR2 (256)**	The destination domain name to which the call was forwarded.
call-dest-regid	type="String"	type="name"	**VARCHAR2 (256)**	The server name if available, or user portion of the To: URI.
call-id	type="String"	type="name"	**VARCHAR2 (256)**	The unique call identifier of the inbound call leg.
call-id-2	type="String"	type="name"	**VARCHAR2 (256)**	The secondary call identifier for the outgoing leg.
call-pdd	type="uint32"	type="int4"	**NUMBER**	The post dial delay between the initial INVITE and the 180/183 RINGING.; calculated in msec.
call-source-realm-name	type="String"	type="name"	**VARCHAR2 (256)**	The source domain name from which the call was received.
call-source-regid	type="String"	type="name"	**VARCHAR2 (256)**	The server name if available, or user portion of the From: URI.
call-type	type="String"	type="name"	**VARCHAR2 (256)**	The type of call, such as IV for Inbound Voice.
called-party-after-src-calling-plan	type="String"	type="namev"	**VARCHAR2 (256)**	The called party number after any manipulation on leg 1, but before any manipulation on leg 2.
cdr-type	type="String"	type="name"	**VARCHAR2 (256)**	The call record type, either START or STOP.
codec-on-dest-leg	type="String"	type="name"	**VARCHAR2 (256)**	The CODEC associated with the outbound call leg.
codec-on-src-leg	type="String"	type="name"	**VARCHAR2 (256)**	The CODEC associated with the inbound call leg.
connect-time	type="Time"	type="timestamp"	**TIMESTAMP**	The time at which the SIP was connected to the SIP call destination in the format hour:minutes:seconds.milliseconds: weekday year-month-day.
creation-timestamp	type= "Time"	type="timestamp"	**TIMESTAMP**	The time the accounting record was written to the accounting target.

Table 24–1 (Cont.) CDR Field Descriptions and Data Types

CDR Field	MS-SQL Data Types	PostgreSQL Data Types	Oracle Data Types	Description
custom-data	type="String"	type="name"	**VARCHAR2 (256)**	Custom data field as defined by the **accounting-data** object.
disconnect-cause	type="Disconnect Type"	type="int4"	**NUMBER**	The reason for the call disconnection, such as BYE.
disconnect-error-type	type="String"	type="name"	**VARCHAR2 (256)**	The type of error that caused the disconnection.
disconnect-time	type="Time"	type="timestamp"	**TIMESTAMP**	The time at which the SIP was disconnected from the SIP call destination in the format hour:minutes:seconds.milliseconds: weekday year-month-day.
dnis	type="String"	type="name"	**VARCHAR2 (256)**	Dialed Number Identification Service
duration	type="uint32"	type="int4"	**NUMBER**	Duration of the call in seconds.
from	type="String"	type="name"	**VARCHAR2 (256)**	The string in the From URI:field of SIP header.
header	type="String"	type="name"	**VARCHAR2 (256)**	An arbitrary header associated with the SIP call.
hunting-attempts	type="uint32"	type="int4"	**NUMBER**	The number of times the ME used the arbiter to select a dial-plan and a failure occurred (including subsequent attempts).
in_anchor_dst	type="IPPort"	type="name"	**VARCHAR2 (256)**	The IP address and port at the ME where the inbound call leg was received from the source peer.
in_anchor_src	type="IPPort"	type="name"	**VARCHAR2 (256)**	The IP address and port at the ME where the inbound call leg was forwarded to the destination peer.
in_peer_dest	type="IPPort"	type="name"	**VARCHAR2 (256)**	The IP address and port of the destination phone to which the ME forwarded the inbound call leg.
in_peer_src	type="IPPort"	type="name"	**VARCHAR2 (256)**	The IP address and port of the source phone that contacted the ME over an inbound call leg.
incoming-request-uri	type="String	type="name"	**VARCHAR2 (256)**	The Request URI on the inbound call leg.
incoming-uri-stripped	type="String"	type="name"	**VARCHAR2 (256)**	The stripped down version of the incoming request URI.

Table 24–1 (Cont.) CDR Field Descriptions and Data Types

CDR Field	MS-SQL Data Types	PostgreSQL Data Types	Oracle Data Types	Description
last-pkt-timestamp-on-dest-leg	type="Time"	type="timestamp"	**TIMESTAMP**	The time of the last media packet on the destination leg.
last-pkt-timestamp-on-src-leg	type="Time"	type="timestamp"	**TIMESTAMP**	The time of the last media packet on the source leg.
last-status-message	type="uint16"	type="int4"	**NUMBER**	An integer indicating SIP message type last status message (omitting "200 OK") and therefore call progress.
latency-on-dest-leg	type="uint32"	type="int4"	**NUMBER**	The total processing time of the outbound call leg.
latency-on-src-leg	type="uint32"	type="int4"	**NUMBER**	The total processing time of the inbound call leg.
max-jitter-on-dst-leg	type="uint32"	type="int4"	**NUMBER**	The maximum jitter on the destination leg.
max-jitter-on-src-leg	type="uint32"	type="int4"	**NUMBER**	The maximum jitter on the source leg.
max-latency-on-dst-leg	type="uint32"	type="int4"	**NUMBER**	The maximum latency on the destination leg.
max-latency-on-src-leg	type="uint32"	type="int4"	**NUMBER**	The maximum latency on the inbound call leg.
method	type="String"	type="name"	**VARCHAR2 (256)**	The SIP method, such as INVITE or REGISTER, that initiated the call session.
mimetype-on-dest-leg	type="String"	type="name"	**VARCHAR2 (256)**	The MIME type associated with the outbound call leg, such as audio/pcmu.
mimetype-on-src-leg	type="String"	type="name"	**VARCHAR2 (256)**	The MIME type associated with the inbound call leg, such as audio/pcmu.
mos-fmt-dest-leg	type="String"	type="name"	**VARCHAR2 (256)**	The formatted MOS calculation on the oubound call leg.
mos-fmt-on-src-leg	type="String"	type="name"	**VARCHAR2 (256)**	The formatted MOS calculation on the inbound call leg.
mos-on-dest-leg	type="uint32"	type="int4"	**NUMBER**	The MOS calculation * 1000 on the oubound call leg.
mos-on-src-leg	type="uint32"	type="int4"	**NUMBER**	The MOS calculation * 1000 on the inbound call leg.
new-ani	type="String"	type="name"	**VARCHAR2 (256)**	The caller ID for the ANI after any manipulation is done by the ME.
newDnis	type="String"	type="name"	**VARCHAR2 (256)**	New Dialed Number Identification Service

Table 24–1 (Cont.) CDR Field Descriptions and Data Types

CDR Field	MS-SQL Data Types	PostgreSQL Data Types	Oracle Data Types	Description
next-hop-dn	type="String"	type="name"	**VARCHAR2 (256)**	The fully qualified domain name (FQDN) or IP address of the next network node handling the call forwarded by the ME.
next-hop-ip	type="IPHost"	type="int4"	**NUMBER**	The IP address of the next hop; the next network node handling the call forwarded by the ME device.
origGW	type="String"	type="name"	**VARCHAR2 (256)**	The name of the originating gateway associated with the call.
origin	type="String"	type="name"	**VARCHAR2 (256)**	The ORIGIN header associated with the SIP call.
out_anchor_dst	type="IPPort"	type="namev"	**VARCHAR2 (256)**	The IP address and port at the ME where the outbound (responding) call leg was received from the destination peer.
out_anchor_src	type="IPPort"	type="name"	**VARCHAR2 (256)**	The IP address and port at the ME where the outbound call leg was forwarded back to the source peer.
out_peer_dst	type="IPPort"	type="name"	**VARCHAR2 (256)**	The IP address and port of the destination phone to which the ME forwarded the outbound (return) call leg.
out_peer_src	type="IPPort"	type="name"	**VARCHAR2 (256)**	The IP address and port of the responding destination phone from which an outbound call leg was returned to the ME.
outgoing-request-uri	type="String"	type="name"	**VARCHAR2 (256)**	The Request URI on the outbound leg.
packets-discarded-on-dest-leg	type="uint32"	type="int4"	**NUMBER**	The total number of packets discarded on the outbound call leg.
packets-discarded-on-src-leg	type="uint32"	type="int4"	**NUMBER**	The total number of packets discarded on the inbound call leg.
packets-lost-on-dest-leg	type="uint32"	type="int4"	**NUMBER**	The total number of packets lost on the outbound call leg.
packets-lost-on-src-leg	type="uint32"	type="int4"	**NUMBER**	The total number of packets lost on the inbound call leg.

Table 24–1 (Cont.) CDR Field Descriptions and Data Types

CDR Field	MS-SQL Data Types	PostgreSQL Data Types	Oracle Data Types	Description
packets-received-on-dest-leg	type="uint32"	type="int4"	**NUMBER**	The total number of packets received on the outbound call leg.
packets-received-on-src-leg	type="uint32"	type="int4"	**NUMBER**	The total number of packets received on the inbound call leg.
previous-hop-ip	type="IPHost"	type="int4"	**NUMBER**	The IP address of the previous hop; the last network node handling the call before received at the ME device.
previous-hop-via	type="String"	type="name"	**VARCHAR2 (256)**	The VIA header from the previous hop.
pvd-on-dest-leg	type="uint32"	type="int4"	**NUMBER**	The packet delay variation (jitter) associated with the call on the outbound call leg.
pvd-on-src-leg	type="uint32"	type="int4"	**NUMBER**	The packet delay variation (jitter) associated with the call on the inbound call leg.
recorded	type="Boolean"	type="int4"	**NUMBER**	The true or false indication as to whether the SIP call was recorded.
rfactor-on-dest-leg	type="uint16"	type="int4"	**NUMBER**	The R-factor integer used in the MOS score compilation on the destination call leg.
rfactor-on-dest-leg-times-1000	type="uint32"	type="int4"	**NUMBER**	The R-factor integer * 1000 this is used in the MOS score compilation on the destination call leg.
rfactor-on-src-leg	type="uint16" or type="uint32"	type="int4"	**NUMBER**	The R-factor integer used in the MOS score compilation on the inbound call leg.
rfactor-on-src-leg-times-1000	type="uint32"	type="int4"	**NUMBER**	The R-factor integer * 1000 this is used in the MOS score compilation on the inbound call leg.
scp-name	type="String"	type="name"	**VARCHAR2 (256)**	The ME virtual system partition (VSP) that handled the call.
session-id	type="uint64" format="hex" key="index"	type="int8"	**NUMBER**	The unique session identifier in hexadecimal format, unassigned 64-bit integer.

Table 24–1 (Cont.) CDR Field Descriptions and Data Types

CDR Field	MS-SQL Data Types	PostgreSQL Data Types	Oracle Data Types	Description
setup-time	type="Time" key="index"	type= "timestamp"	**TIMESTAMP**	The time at which the SIP was set up at the ME in the format hour:minutes:seconds.milliseconds: weekday year-month-day.
setup-time-integer	type="uint64"	type="int8"	**NUMBER**	The call setup time indicated as an integer.
termGW	type="String"	type="name"	**VARCHAR2 (256)**	The name of the gateway where the call was terminated.
to	type="String"	type="name"	**VARCHAR2 (256)**	The string in the To URI: field of the SIP header.

25

Configuring Domain Name Systems (DNS)

This chapter covers DNS configurations on the ME system.

Domain Name System (DNS) Overview

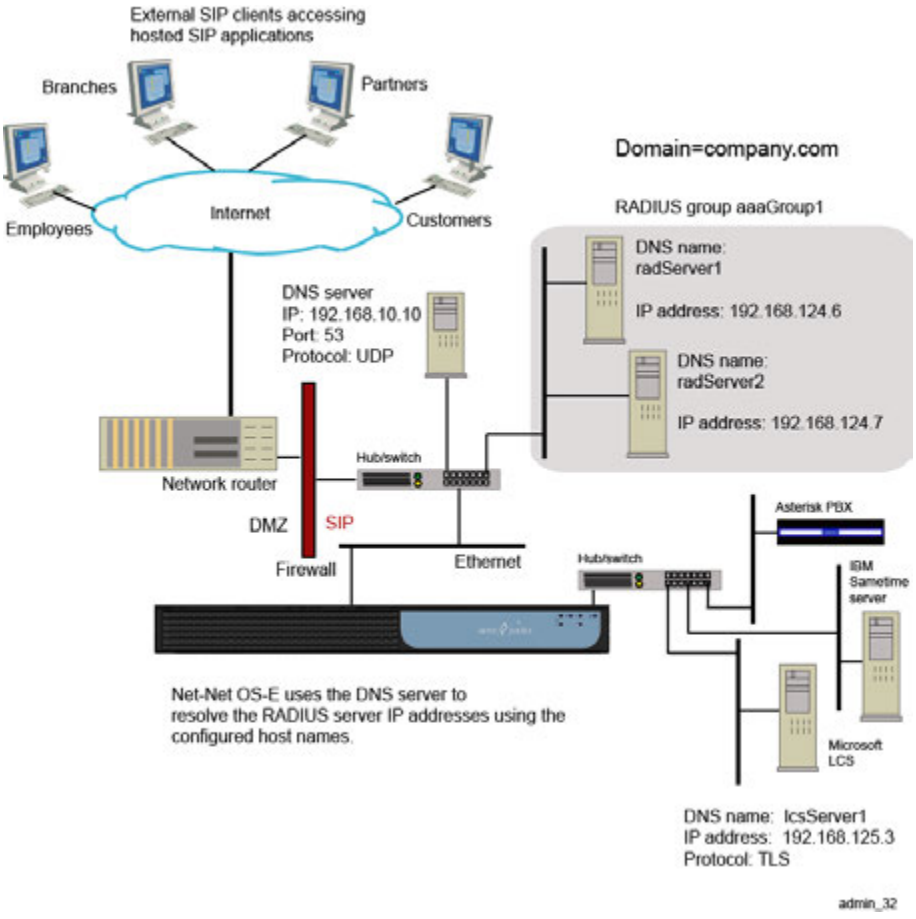
Domain Name System (DNS) servers are responsible for translating Internet domain and host names to IP addresses. DNS converts the name entered on a Web browser address bar to the IP address of the Web server that hosts that particular Web site. DNS uses a distributed database to store this name and address information for all public hosts on the Internet.

When an Internet client issues a request that involves an Internet host name, a DNS server determines the host's IP address. If the DNS server cannot service the request, it forwards the request to other DNS servers until the IP address is resolved, completing the Internet client request.

The ME maintains a cache of query responses: positive responses were successful and negative (reject) responses failed. This response is the DNS resource record, allowing the ME to consult its cache for mapping information before querying a server.

RADIUS and Diameter group accounting configurations, for example, require that you configure DNS to resolve the IP addresses associated with RADIUS and Diameter servers being used to capture call detail records.

The following image illustrates a sample network with a DNS server that resolves RADIUS server IP addresses using the domain name.



Configuring the DNS Resolver

The ME system functions as a DNS client (resolver) that forwards requests for IP address resolutions, but does not act as a server in accepting requests. As a resolver, the ME obtains resource records from DNS servers on behalf of resident or requesting applications. You must configure the resolver function before other objects within the DNS configuration object.

> **Note:** You must configure the settings of the resolver object before setting other objects under DNS.

The DNS object configures the ME domain name, one or more DNS servers, and static mapping between host names and addresses. You can also configure static service locations, naming authority pointers, and how to resolve negative entries.

CLI Session

The following CLI session configures and enables the DNS resolver, sets the domain name to be used for DNS mappings, sets the DNS server IP address, port number and transport protocol, and the DNS query properties.

```
NNOS-E> config vsp
config vsp> config dns
config dns> config resolver
config resolver> set admin enabled
```

```
config resolver> set server 192.168.10.10 UDP 54
config resolver> set query-timeout 10
config resolver> set query-retries 5
config resolver> set cache-poll-interval 60
```

The **query-timeout** property specifies the time, in seconds (between 1 to 10), that a DNS lookup can go unanswered before it times out. The **query-retries property** specifies the number of DNS query (lookup) retries to execute if a DNS query times out. Enter a number of retries between 0 to 5, where 0 indicates no retries.

The **cache-poll-interva**l property specifies the number of seconds that the ME waits between refreshing the cache. The interval controls the rate at which the ME polls the location-cache to purge stale location bindings.

Configure as many DNS servers as you need. Refer to the *WebRTC Session Controller Media Engine Object Reference* for information on additional settings.

Configuring DNS Hosts and IPs

For each host in your network domain, you need to statically map IP addresses to host names. The **host** object requires that you supply a **name** variable. This is the name of an Internet node, for example, a SIP server, a RADIUS server, or a PC in your network.

You can enter:

- An existing name and new address; the corresponding address is mapped to the name for use in DNS lookups, or
- A new name and existing address; the system creates a named entry for DNS use.

CLI Session

The following DNS session configures the DNS host name for the RADIUS server named **radServer1** and sets the IP address to be returned in DNS lookups.

```
NNOS-E> config vsp
config vsp> config dns
config dns> config host radServer1
Creating 'host radServer1'
config host radServer1> set address 192.168.124.6
```

The following DNS session configures the DNS host name for the SIP server named **lcsServer1** and sets the IP address to be returned in DNS lookups.

```
NNOS-E> config vsp
config vsp> config dns
config dns> config host lcsServer1
Creating 'host lcsServer1
config host lcsServer1> set address 192.168.125.3
```

Mapping SIP Services

The DNS **service** object allows you to statically map SIP services to specific SIP servers. Using a configured rule, DNS resolves the SIP service and maps the service to a specific SIP server. By adding DNS server resource (SRV) records for each SIP service, SRV records provide contacts for the specific DNS servers.

The **rule** property establishes the preference level for selecting a named SIP service if you configure multiple SIP service mappings. Configuring the **service** object for each SIP service establishes the sequence to use when contacting the configured SIP servers.

CLI Session

The following CLI session maps the TLS service on the **company.com** domain. DNS resolves the TLS service to **lcsServer1** using the configured rule (port, priority, and weight settings).

```
NNOS-E> config vsp
config vsp> config dns
config dns> config service company.com tls
Creating 'service company.com tls'
config service company.com> set rule lcsServer1.company.com
5001 10 5
```

Configuring NAPTR

The Naming-authority pointer (called NAPTR) creates a static mapping of service information to a specific server or domain name. This mapping performs DNS lookups for requests in cases where the ME system cannot determine either the protocol or port of the destination.

Naming-authority pointer (NAPTR) records contain rules for converting each request to the correct configured service. Because each transport service over SIP is viewed as a different service (TCP, UDP, or TLS), they establish three different NAPTR records. This object configures the preference for use of an appropriate service for each domain.

Set one rule for each protocol: UDP, TCP, and TLS. Before a request can be forwarded on, the system must know the protocol and the port for the destination.

CLI Session

The following CLI session sets the NAPTR rules (protocol, order, preference) for SIP TLS, TCP and UDP services on the **company.com** domain. DNS uses the configured SIP services (TLS, TCP, UDP) to resolve the destination SIP server, using exact matching of the **company.com** domain name.

```
NNOS-E> config vsp
config vsp> config dns
config dns> config naptr company.com
Creating 'naptr company.com'
config naptr company.com> set match exact
config naptr company.com> set rule TLS 1 10
config naptr company.com> set rule TCP 2 10
config naptr company.com> set rule UDP 3 10
```

For more information on NAPTR and DNS on the ME system, refer to the *WebRTC Session Controller Media Engine Object Reference*.

Configuring DNS Rejections

You can instruct DNS to ignore lookups that involve certain domain names. The DNS **reject** object requires that you supply a host name, service name, domain name, or IP address. Any request containing the specified name will be rejected.

Set the **type** property to identify which record type you are entering:

- A: IPv4 address
- AAAA: IPv6 address
- PTR: Address to name mapping
- NAPTR: NAPR rule

CLI Session

The following CLI session rejects DNS lookups that involve the domain named **evilBadGuy.com**., using the IPv4 address, matching the exact domain name as entered.

```
NNOS-E> config vsp
config vsp> config dns
config dns> config reject badNetwork.com naptr
Creating 'reject badNetwork.com naptr'
config reject badNetwork.com> set match exact
```

For more information on DNS rejections on the ME system, refer to the *WebRTC Session Controller Media Engine Object Reference*.