Oracle® Communications Application Session Controller

Web Services SOAP/REST API

Release 3.7.0

May 2016

Contents

About This Guide

Overview

Net-Net ASC Web Service is a WSDL/REST Application Programming Interface (API) enabling enterprises, service providers, and third-party developers to streamline business processes by integrating their applications with IP communications services.

Audience

This guide is written for application developers and network administrators, and provides information about the Net-Net ASC WSDL/REST-based Web Services implementation.

For information about Net-Net system training, contact your Oracle sales representative directly or email support@acmepacket.com

About Net-Net OS-E Documentation

The Net-Net OS-E references in this documentation apply to the Net-Net OS-E operating system software that is used for the following Oracle and third-party SBC products:

- Oracle Communications Application Session Controller (ASC)
- Oracle Communications WebRTC Session Controller (WSC)
- Oracle Communications OS-E Session Director (SD) Session Border Controller (SBC)
- Oracle Communications 2600 Session Director (SD) Session Border Controller (SBC)
- Third-party products that license and use Net-Net OS-E software on an OEM basis

Unless otherwise stated, references to Net-Net OS-E in this document apply to all of the Oracle and third-party vendor products that use Net-Net OS-E software.

The following documentation set supports the current release of the OS-E software.

- *Oracle Communications Application Session Controller System and Installation Commissioning Guide*
- *Oracle Communications Application Session Controller System and Installation Commissioning Guide Release 3.7.0M4*
- *Oracle Communications Application Session Controller Management Tools*
- *Oracle Communications Application Session Controller System Administration Guide*
- *Oracle Communications Application Session Controller Session Services Configuration Guide*
- *Oracle Communications Application Session Controller Objects and Properties Reference*

- *Oracle Communications Application Session Controller System Operations and Troubleshooting*
- *Oracle Communications Application Session Controller Release Notes*
- *Oracle Communications Application Session Controller Single Number Reach Application Guide*
- *Oracle Communications Application Session Controller Web Services SOAP REST API*
- *Oracle Communications WebRTC Session Controller Installation Guide*

Revision History

This section contains a revision history for this document.

Date	Revision Number	Description
June 28, 2013	Revision 1.00	• Initial release of the OS-E 3.7.0 software.
October 31, 2013	Revision 1.10	• Adds Appendix C ASC Web Services Samples.
February 27, 2015	Revision 1.11	• Updates “Specifying Output and Callback” and “Set Configuration” sections to include more thorough descriptions.
October 27, 2015	Revision 1.20	• Adds Appendix C ASC Call Reconnect SDK.
May 17, 2016	Revision 1.21	• Adds *Oracle Communications Application Session Controller System Installation and Commissioning Guide Release 3.7.0M4* to the 3.7.0 doc set. • Adds a note to Appendix C ASC Web Services Samples regarding sample support in release 3.7.0M4. • Adds a note regarding a necessary argument with the **mix-session** and **mix-session-threaded** actions in the context of on-demand recording.

1 About the Web Service Interface

Introduction

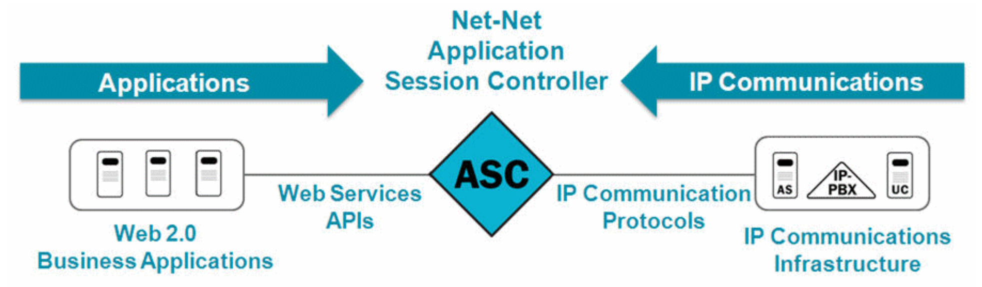
The Net-Net ASC Web Service is a SOAP/REST Application Programming Interface (API) which enables enterprises, service providers, and third-party developers to streamline business processes by integrating their applications with IP communications services.

A web service is a software system that supports interoperable machine-to-machine interaction over a network using HTTP/HTTPS transport.

This document provides a full description of the individual interface definitions that make up the ASC API.

What is the ASC?

The Net-Net ASC is a programming platform that enables enterprises, service providers, and third-party developers to streamline business processes by integrating their applications with IP communications services. The ASC implements both a SOAP-based web service interface, as well as a RESTful web service interface for invoking remote web services.



What Are SOAP-Based Web Services?

SOAP is a protocol that uses XML for exchanging structured information in the implementation of web services. A SOAP message consists of three parts:

- An envelope that defines what is included in the message and how to process it.
- A set of encoding rules which define data objects and types.
- The convention that is used to represent call and response procedures.

What is WSDL?

The ASC uses Web Service Description Language (WSDL) to define its available actions and types for SOAP-based web service.

What is REST?

REST is an API style supported by the ASC for web service which implements a URI using HTTP and a collection of resources with three defined aspects:

- The base URI for the web service.
- The format of the data returned by the REST URL. This is usually either XML or JavaScript Object Notation (JSON).
- A set of ASC web service operations.

There are two action and status report request formats available when using RESTful web service, flat and hierarchical. When possible, Oracle recommends using the hierarchical format, which is a simpler way to encode REST requests.

What is WADL?

For RESTful web service, the ASC uses Web Application Description Language (WADL) to define its available actions and types.

Specifying Output and Callback

When you are using REST, the default format returned by the REST URL is XML. However, you can request to receive the output in the JavaScript Notation (JSON) format instead by specifying this in the URI.

To specify the format in which you want the responses to REST requests:

```
output=<response_format>
```

Options are:

- json: Use JSON format
- xml: (default) Use XML format

If you choose JSON, the ASC supports Javascript callbacks when using REST. To configure a callback, specify the JavaScript method name in the URI. The JavaScript method is called with the JSON output string as a parameter.

```
callback=<JavaScript_method_name>
```

If you choose XML, you must specify an XML output format.

```
_format=<xml_format>
```

Options are:

- simplified
- legacy (default)

Note: For more information on legacy and simplified schema, see “Legacy and New Schema” on page 1-17.

Accessing the ASC

The ASC web service interfaces are platform-agnostic. Any application environment, programming language, or development environment capable of sending HTTP requests may be used, including:

- Programming languages (ie., C#, Java)
- Mobile platforms (ie., iOS, Android)
- Purely web-based languages (ie., JavaScript, PHP, Python)

To access the web services homepage, the default is

```
http://x.x.x.x:8080
```

where x.x.x.x is the IP static-address where the **web-services** configuration is enabled.

The ASC web services homepage is where all user documentation and samples are located.

Supported ASC Functionality

The ASC API supports retrieving and setting all configuration objects, invoking all actions, and retrieving all status reports available on the Net-Net OS-E. Configuration, action, and status objects are referred to in this document and in the API as objects and sub-objects.

Terminology

The following terms are used throughout the document:

- *Object* – Configuration, status, or action data.
- *Property* – Attribute of an object
- *Alias* – Display name of an object or property

Authentication

The ASC requires authentication of client endpoints for security purposes. When a request is sent by a web services application to the ASC, a session cannot be established without authentication being performed.

The ASC can perform either basic authentication, which requires HTTP basic authentication for client connections, or it can perform certificate-based authentication. This requires an HTTPS certificate for authentication of client connections. Upload a unique certificate via the **vsp > tls** object.

NOTE: In order for authentication information to be encrypted, you must be using HTTPS.

When SOAP-based messages are used to send requests to the ASC and access permissions have been configured, the SOAP client endpoint sending the request must also send the username and password with the request. Basic HTTP authentication is supported, as well as certificate-based HTTPS authentication.

REST requests can be authenticated using basic HTTP authentication, or can use the REST-specific login action, defined in all WADLs published by the ASC.

The ASC communicates with web services applications in “sessions”. A session timeout is not configurable and is hard-coded to 30 minutes.

Configuring Access

For authentication to work, you must have at least one user configured under the **access** object, with **access > permissions > web-services** set to **enabled**.

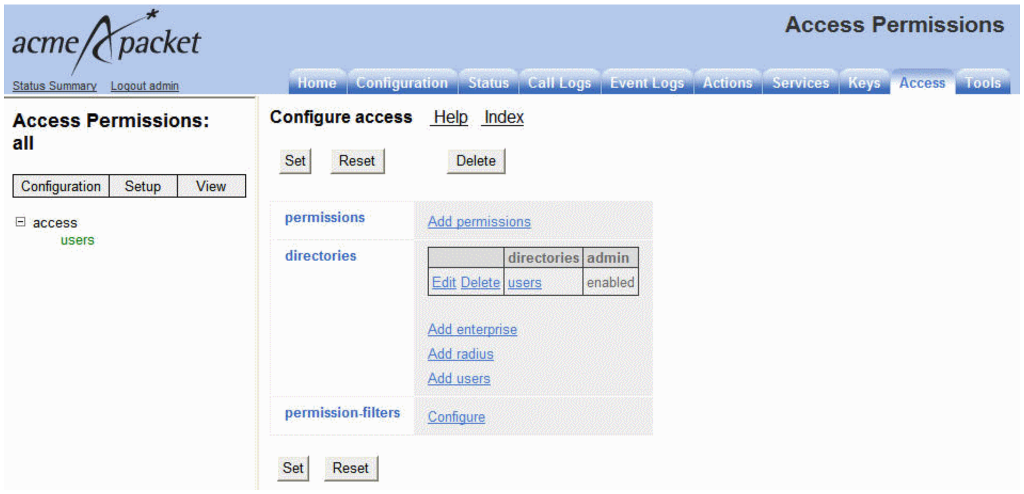
NOTE: Users with the web-services permission enabled have access to the entire ASC system (all configuration objects, statuses, and actions).

The first step is to create a permission set with **web-services** enabled. Once this has been done, create a user and assign that user the web-services enabled permission set.

To create a web-services permission set:

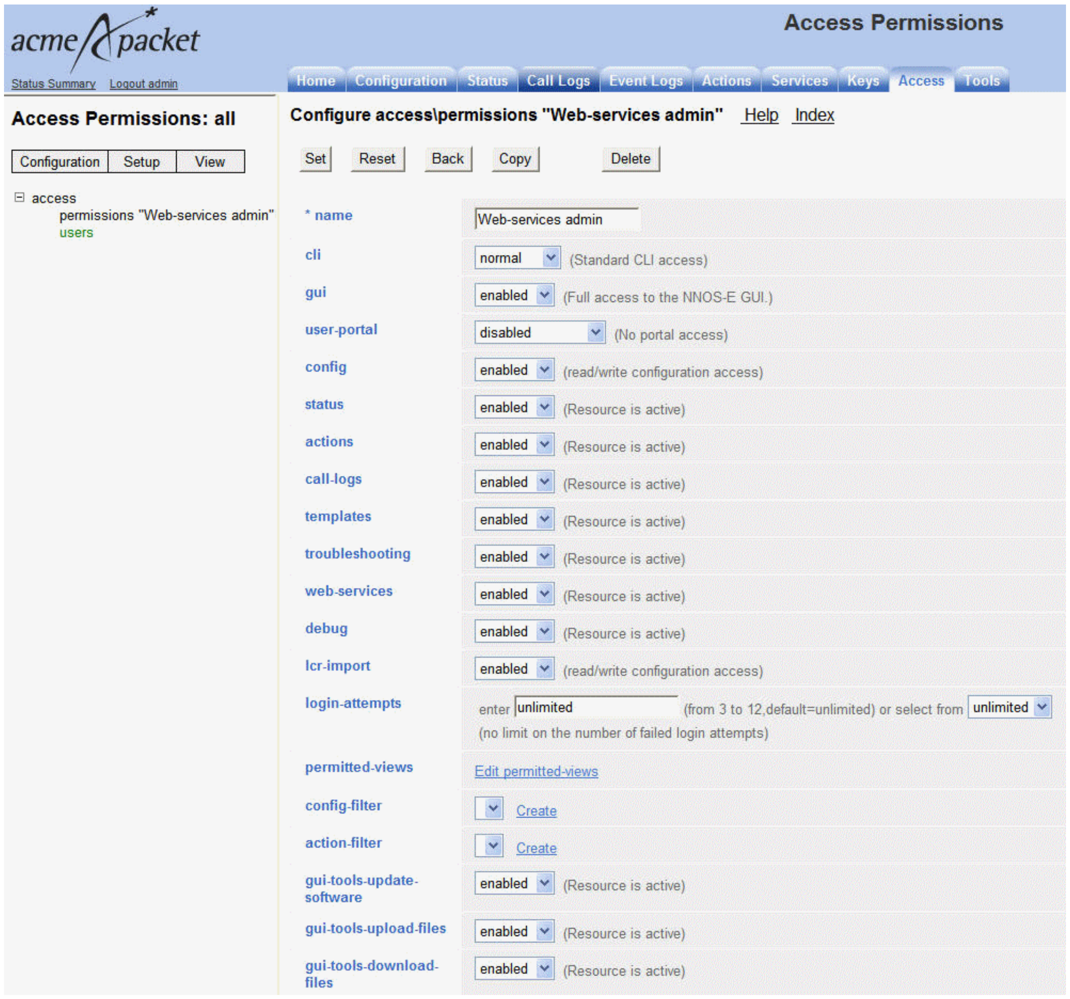
1. Click the **Access** tab and select **access**.

2. Click **Add permissions**.



permissions	Add permissions
directories	directories / admin: Edit Delete, users, enabled; Add enterprise; Add radius; Add users
permission-filters	Configure

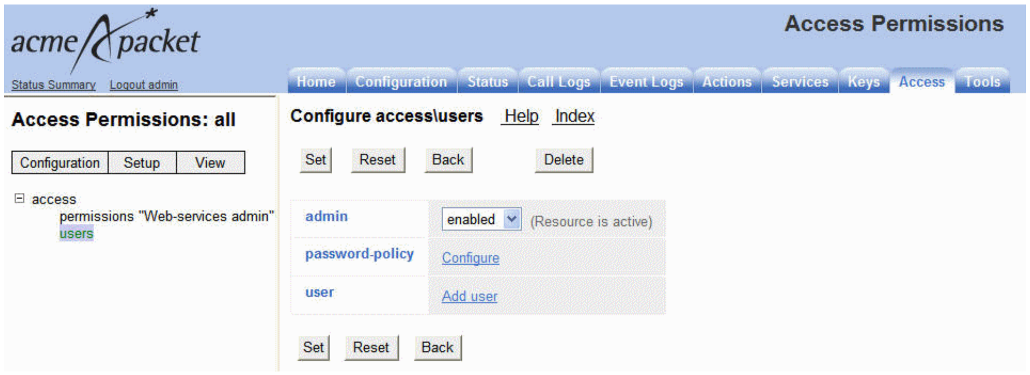
3. Name the permission set and click **Create**. The page listing all available permissions appears. This example shows a permission set named “Web-services admin.”



* name	Web-services admin
cli	normal (Standard CLI access)
gui	enabled (Full access to the NNOS-E GUI.)
user-portal	disabled (No portal access)
config	enabled (read/write configuration access)
status	enabled (Resource is active)
actions	enabled (Resource is active)
call-logs	enabled (Resource is active)
templates	enabled (Resource is active)
troubleshooting	enabled (Resource is active)
web-services	enabled (Resource is active)
debug	enabled (Resource is active)
lcr-import	enabled (read/write configuration access)
login-attempts	enter unlimited (from 3 to 12,default=unlimited) or select from unlimited (no limit on the number of failed login attempts)
permitted-views	Edit permitted-views
config-filter	Create
action-filter	Create
gui-tools-update-software	enabled (Resource is active)
gui-tools-upload-files	enabled (Resource is active)
gui-tools-download-files	enabled (Resource is active)

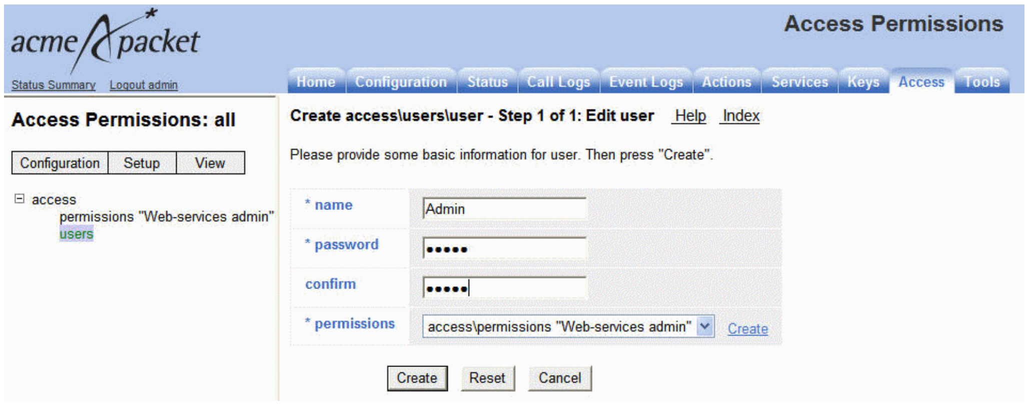
4. Enable **web-services** and click **Set**. The permission set is created.

5. Update and save the running configuration.
6. Click **users** and select **Add user**.



admin	enabled (Resource is active)
password-policy	Configure
user	Add user

7. Enter the user **name** and **password**.
8. Select the permission set just created with **web-services** enabled. This example shows a user named Admin.



Create access\users\user - Step 1 of 1: Edit user

Please provide some basic information for user. Then press "Create".

* name	Admin
* password	•••••
confirm	•••••
* permissions	access\permissions "Web-services admin"

9. Click **Create**.
10. Click **Set**. The user is created.
11. Save and update the configuration.

Legacy and New Schema

There are two types of schema the ASC supports, legacy and new. The schema is the WSDL’s .xsd file’s specification of all configuration, status, action, and event objects on the ASC. These schemas are equivalent and support the same functionality. The ASC supports the existing legacy format for backwards compatibility and in the cxc.wsdl file, generates verbose Java and C# code.

The new format is much more compact and concise than the legacy. The file name for the new format is AcmePacketASCManagement.wsdl.

NOTE: Oracle recommends you use the new schema, particularly if you are implementing a new ASC application. Existing ASC applications may continue to use the legacy format for backwards compatibility purposes only.

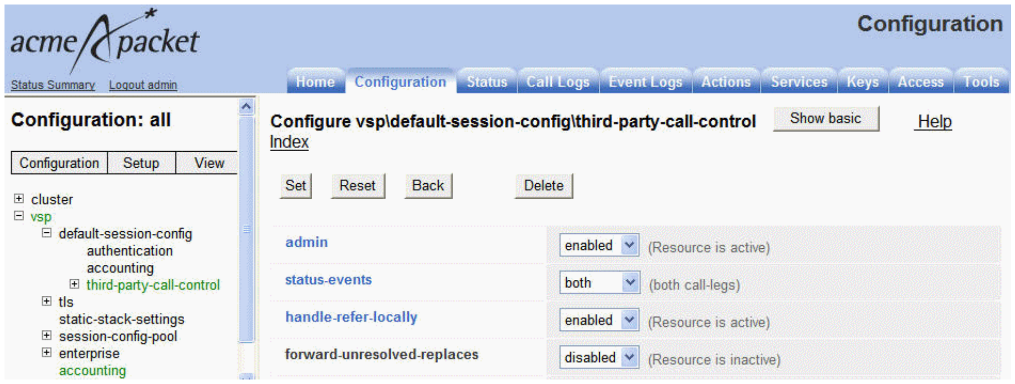
Legacy and Custom Event Messages

The ASC includes certain standard information in the event messages it sends. However, you can choose to include new information not included in the standard format. You can configure the ASC to include custom content in these event messages.

See Appendix B: Event Message Examples for examples of both legacy and new format and legacy and custom content event messages.

To include custom information in event messages:

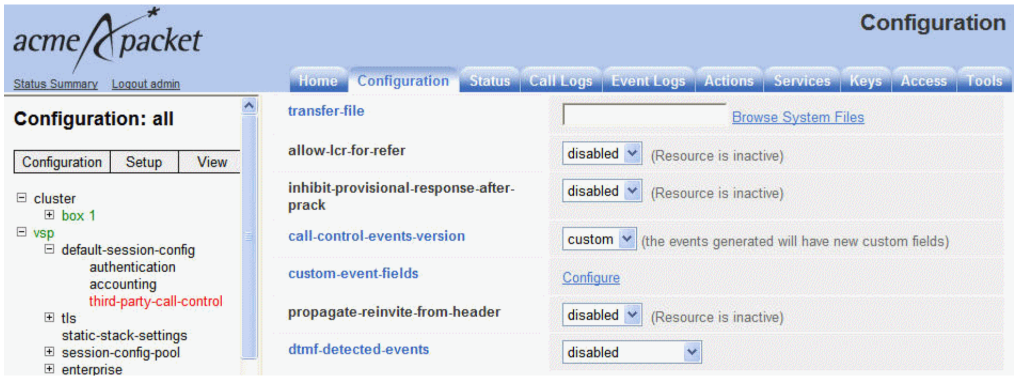
1. Click the **Configuration** tab and select either **default-session-config** or **session-config-pool > entry**.
2. Click on the **third-party-call-control** object.
3. Set **admin** to **enabled**.



Configure vsp\default-session-config\third-party-call-control

admin	enabled	(Resource is active)
status-events	both	(both call-legs)
handle-refer-locally	enabled	(Resource is active)
forward-unresolved-replaces	disabled	(Resource is inactive)

4. Select **custom** from the **call-control-events-version** drop-down box. The default is **legacy**.
5. Click **Configure** next to **custom-event-fields** to set the custom event fields to include in the event messages.



transfer-file		Browse System Files
allow-lcr-for-refer	disabled	(Resource is inactive)
inhibit-provisional-response-after-prack	disabled	(Resource is inactive)
call-control-events-version	custom	(the events generated will have new custom fields)
custom-event-fields	Configure	
propagate-reinvite-from-header	disabled	(Resource is inactive)
dtmf-detected-events	disabled	

For more information on configuring named variables and regular expressions, see Using Regular Expressions in Chapter 1: How to Use the ACLI of the Oracle Communications OS-E Objects and Properties Reference Guide.

6. Click **Set**.
7. Update and save the configuration.

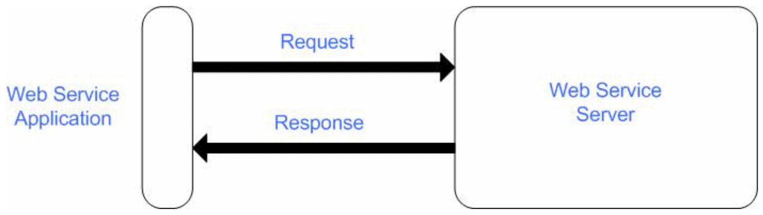
Web Services Requests

A web service request is a request made by a web services application sent via HTTP/HTTPS to the ASC web services server. When the server receives a request, it processes it and sends back a response.

The response that the ASC sends back contains a code number and a message. If the action was successful, the code is 0. If there is an error with the request, the code will be a value other than 0. The error message describes what error occurred.

When processed successfully, the response can contain:

- Information requested via the following top-level APIs
  - get configuration
  - get status
  - query status
- Status for an operation being performed via the following top-level APIs
  - set configuration
  - execute action



Get Configuration

The ASC "Get configuration" API is a request to the server to receive all or a portion of the configuration. Specify the configuration objects or properties you want returned. If you specify no parameters, the entire configuration is returned.

The internal names for the top level configuration objects are:

- cluster—Cluster
- services—Services
- master-services—MasterServices
- vsp—SCP
- external-services—ExternalServices
- preferences—Preferences
- access—CXCAccess
- features—Features
- box—Box

SOAP

The SOAP "Get configuration" request name is getConfig.

Response Content:

XML Format: The configuration. The schema is defined in cxc.xsd (legacy) or AcmePacketASCManagement.xsd (new).

REST

The REST “Get configuration” request resource path is

```
/cms/config
```

using the HTTP GET method.

If parameters are specified, include the path of the configuration under the top level object to be retrieved.

Response Content:

XML or JSON format (XML is the default if no format is specified): ExtPageList structure. This includes:

- objects—Configuration objects
- resultCode—0 if success; error code if error occurs
- resultStr—”Success” if success; error message if error occurs

Set Configuration

The ASC “Set configuration” API is a request to the server to change all or a portion of the configuration.

SOAP

The SOAP “Set configuration” name is setConfig. Specify the configuration parameters you want to set, then specify a mode. The valid modes are:

- merge—Merges the configuration in the request with the existing configuration on the ASC.
- replace-full—Replaces the entire existing configuration on the ASC with the configuration in the request.
- replace-partial—Replaces only top-level existing ASC configuration wiht top-level configuration objects in the request.

  Note: Setting the **mode** parameter to **replace-full** deletes the entire existing configuration and leaves only the request configuration in its place and therefore should be used with caution.

Response Content:

XML Format: setConfigResponse structure. This includes:

- Code—”Success” or “Error”
- Text—Error code if error occurs

REST

The REST “Set Configuration” API request resource path is

```
/cms/config
```

using the HTTP POST method.

The setConfig API syntax is:

```
setConfig <operation> [xmlconfig] [path] [add.property] [mode]
```

The valid parameters are:

- operation: Specifies the action you want the API to take on the configuration. Valid values are:
  - modify <*xmlconfig*> [*mode*]—Modifies the configuration with the configuration you specify in **xmlconfig**.
  - add <*xmlconfig*> <*path*> <*add.property*> [*mode*]—Adds an **xmlconfig** to the specific property (**add.property**) of a parent configuration object (**path**).
  - delete <*path*>—Deletes a configuration object specified by the **path**.
- xmlconfig: (Applicable to the **add** and **modify** operations only.) Specifies the XML format of the configuration to either be added or modified.

  Note: Oracle recommends using the simplified schema, specified at http://x.x.x.x:8080/mgmt?xsd=cxc_simplified.xsd.
- path: (Applicable to the **add** and **delete** operations only.) Specifies the path for the configuration being added or deleted. Use backslashes (\) to separate the list when there are multiple objects in a path (for example, **\cluster\box 1\interface eth0**). You can find an object's path either in the CLI or at the top of the web management configuration page.

  Note: The ASC returns an error if you attempt to delete an object required by another part of the configuration.
- add.property: (Applicable to the **add** operation only.) Specifies which property from the parent object to add to the configuration.
- mode: When modifying or adding, this property specifies how to apply the **xmlconfig** to the existing configuration.
  - merge—(default) Merges the configuration in the request with the existing configuration on the ASC.
  - replace-partial—Replaces only the parts of the existing configuration with those specified in the request configuration.
  - replace-full—Replaces the entire existing configuration on the ASC with the configuration in the request.

  Note: Setting the **mode** parameter to **replace-full** deletes the entire existing configuration and leaves only the request configuration in its place and therefore should be used with caution.

Response Content:

XML or JSON format (XML is the default if no format is specified): structure

- Result code—0 if success; non-zero if error occurs
- Result string—"Success" if success; error message if error occurs

Get Status

The ASC "Get status" API is a request to the server to receive all or a portion of the statuses on the ASC. When working with SOAP, you cannot specify a filter and must receive the entire status report. When working with REST, you can specify a filter to return a subset of the status report. If no filter is specified, the entire status report is returned.

SOAP

The SOAP "Get status" request name is getStatus.

Response Content:

XML format: getStatusResponse structure

REST

The REST “Get status” request resource path is

```
/cms/status/<status alias>
```

using the HTTP GET method.

Specify the pageSize. This is the number of entries returned per page. This is only sent on the first request.

Specify the page. This is the page number to retrieve. This value always starts with 1.

Response Content:

XML or JSON format (XML is the default if no format is specified).

- objects—A list of status objects being returned.
- totalPages—The number of pages of status objects.
- pageSize—The number of entries on each page.
- currentPage—The page number for the current page. This number always starts with 1.
- resultCode—The result code. This number is 0 if the request is successful and a non-zero if an error occurs.
- resultStr—The result string. This string is “Success” if the request is successful and an error message if an error occurs.

Query Status

The ASC “Query status” API is a request to the server to retrieve the status report from the server.

SOAP

The SOAP “Query status” request name is queryStatus.

Specify the status you want to retrieve in XML format. The following example returns the entire **show processes** status report:

```
<status><Processstatus/>
```

You can also specify a property value in the status object to filter the results further. To do this, include

<condition>*condition*</condition>

in the request where *condition* is the status filter you want to use.

Response Content:

XML format: queryStatusResponse structure

REST

The REST “Query status” request source path is

```
/cms/status/<status alias>
```

using the HTTP GET method.

Specify the pageSize. This is the number of entries returned per page. This is only sent on the first request.

Specify the page. This is the page number to retrieve. This value always starts with 1.

You can further narrow the status results by using the **search.x** parameter, where *x* is the property used for filtering status results.

Response Content:

XML or JSON format (XML is the default if no format is specified).

- objects—A list of status objects being returned.
- totalPages—The number of pages of status objects.
- pageSize—The number of entries on each page.
- currentPage—The page number for the current page. This number always starts with 1.
- resultCode—The result code. This number is 0 if the process is a success and a non-zero if an error occurs.
- resultStr—The result string. This string is "Success" if the process is a success and an error code if an error occurs.

Execute Action

The ASC "Execute action" API is a request to the server to perform an action. The ASC can return action data in one of two ways, unstructured or structured. The majority of ASC actions only support unstructured data.

The following actions return structured data:

- arp request
- call-control-attach
- call-control call
- call-control connect
- call-control-create-session
- call-control disconnect
- call-control fork
- call-control hold
- call-control join
- call-control-monitor-session
- call-control park
- call-control annotate
- call-control-redirect
- call-control retrieve
- call-control terminate
- call-control transfer
- call-control-intercept
- call-control-send-message
- config validate

- file-info
- file-play
- ping
- dynamic-event-service

For information on the structured information returned by each of these actions, access the Actions > Response Structures in the web services on-line REST documentation.

SOAP

The ASC supports two SOAP APIs for “Execute action”, doAction and doActionEx. The doAction API is used for returning unstructured data and the doActionEx API is used for actions that return structured data.

Specify the action you want performed in XML format, including all properties.

Response Content:

XML format: doActionResponse structure. This includes:

- Code—”Success” or “Failure”
- Text—Error message if error occurs
- Message—Informational text
- Structured Content if a structured response is being provided.

REST

The REST “Execute action” request resource path is

```
/cms/action/<action alias>
```

using the HTTP GET method.

The parameters you must specify vary depending on the action. To view this information see the web services on-line REST documentation. To do this:

1. Type **http://<*ip:port*>** into the browser.
2. Click on **REST** in the left panel of the screen.
3. Click on the **Actions** link on the REST documentation page.

Response Content:

XML or JSON format (XML is the default if no format is specified): structure. This includes:

- resultCode—0 if success; non-zero if error occurs
- resultString—”Success” if success; error message if error occurs
- Info—Informational text
- Structured Content if a structured response is being provided.

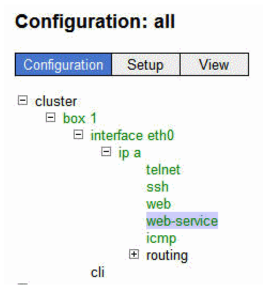
Configuring the ASC

This section describes how to configure the **web-service** object. This is necessary for the ASC to function properly.

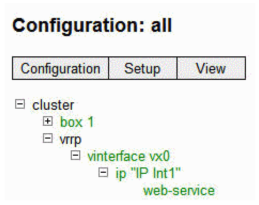
Instructions and Examples

To access web-service on the ASC:

1. Click on the **Configuration** tab and select **web-services**. This can be done via the **box** object using the following path.



   Or it can also be done via the **vrrp** object using the following path.



2. **admin**—Set this property to **enabled** to start the ASC web services process. This property is **enabled** by default.

3. **protocol**—Select the protocol you want to use. After selecting the protocol, select the web services listening port (or accept the default). This is the port the server listens on for HTTP(S) requests. If HTTPS is specified, specify the **vsp > tls** certificate to use with encryption.

   The default values for this property are **http 8080** or **https 8443**. The valid values are:

   - http [port]—Sets an insecure (unencrypted) protocol for use in web transmission. Optionally, you can configure a listening port different than the default.
   - https [port] <certificate> [alias]—Sets a secure transmission of data by using HTTP over SSL. Optionally, you can configure a listening port different than the default. Enter the vsp\tls certificate to use with encryption along with an optional alias value.

4. **max-threads**—Enter the number of threads available to process a request. This includes the number of simultaneous requests and users for your application. The default setting is **10**. The valid values are:

   - Minimum—1
   - Maximum—50

5. **min-spare-threads**—Leave this value at **1**, the default. This is the minimum number of idle threads for processing requests.

6. **max-spare-threads**—Leave this value at **5**, the default. This is the maximum number of idle threads for processing requests.

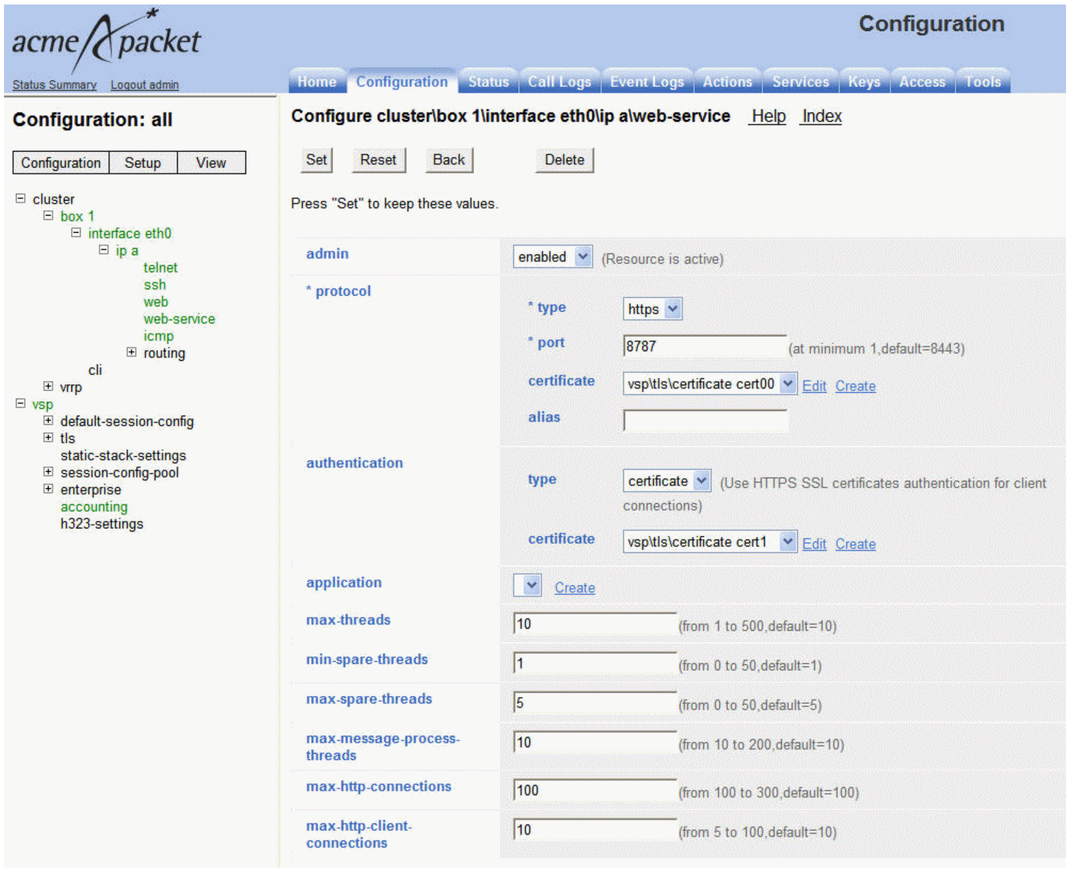
7. **max-message-process-threads**—Enter the maximum number of threads used by the web services process to receive messages from other ASC processes. The default setting is **10**. The valid values are:

   - Minimum—10

- Maximum—200

8. **max-http-connections**—Enter the maximum number of outbound connections for callbacks from the ASC to the web services application for external event notification and external policy processing. The default value is **100**.
   - Minimum—100
   - Maximum—300
9. **max-http-client-connections**—Enter the maximum number of outbound connections to any single host running web services application for callbacks such as external event notification and external policy processing. The default value is **10**.
   - Minimum—5
   - Maximum—100
10. **authentication**—Select the type of authentication you want to use for the ASC web service. The default setting for this property is **certificate**.
    - Basic—This requires the ASC to use HTTP basic authentication for client connections.
    - Certificate—Uses HTTPS SSL certificates authentication for client connections.

    NOTE: You must have at least one user configured under the **access** object with **access > permission > web-services** set to **enabled** in order for authentication to work. Users with the web-services permission enabled have access to the entire system (all configuration, statuses, and actions).



Configure cluster\box 1\interface eth0\ip a\web-service

Press "Set" to keep these values.

admin	enabled	(Resource is active)
* protocol	* type	https
	* port	8787 (at minimum 1,default=8443)
	certificate	vsp\tls\certificate cert00 Edit Create
	alias	
authentication	type	certificate (Use HTTPS SSL certificates authentication for client connections)
	certificate	vsp\tls\certificate cert1 Edit Create
application		Create
max-threads	10	(from 1 to 500,default=10)
min-spare-threads	1	(from 0 to 50,default=1)
max-spare-threads	5	(from 0 to 50,default=5)
max-message-process-threads	10	(from 10 to 200,default=10)
max-http-connections	100	(from 100 to 300,default=100)
max-http-client-connections	10	(from 5 to 100,default=10)

11. Update and save the running configuration.

Configuring the ASC as an Embedded Web Server

The OS-E supports an embedded Tomcat web server which allows users to host some simple Java-based web applications directly on the OS-E.

The OS-E is able to do this by means of virtual hosting. A virtual host is simply an alternate DNS name for an IP address. The OS-E uses a Tomcat server to process requests on a per-domain basis. Based on the configuration, you can select which virtual hosts the OS-E responds to.

> **NOTE:** Because HTTP DNS-based virtual hosting requires distinct names, two names are needed for the v.irtual host to work properly. You need one name to map to the OS-E web services server and the other to map to the applications virtual host.

For example, “ose.example.com” and “oseapps.example.com both point to the same IP. Applications are available on the virtual host oseapps.example.com name, while the applications are configured to use OS-E web services available on the asc.example.com name.

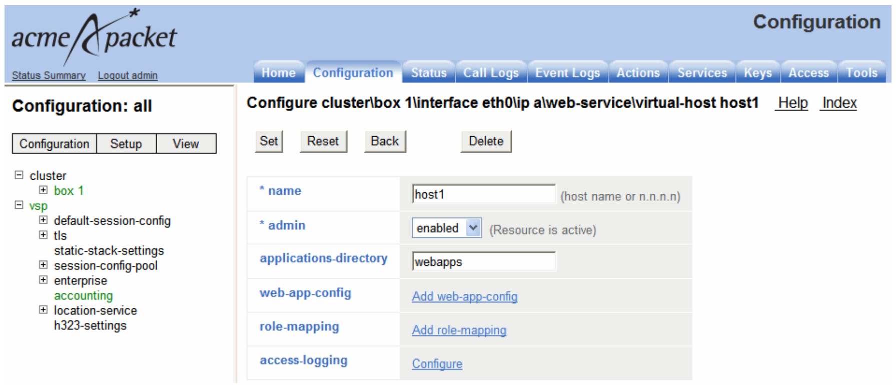
For the embedded web server to work, you must specify a directory in the virtual host configuration on which to copy Web Application Resource (WAR) files you want to deploy.

> **NOTE:** The directory onto which you copy WAR files is always under the cxc_common directory and cannot be changed.

To configure an embedded web server on the web services server, you must configure a **virtual-host** with **web-app-config**, **role-mapping**, and **access-logging** configuration objects.

To configure a virtual host:

1. Select the **Configuration** tab and click the **cluster > box > interface > ip > web-service** object on which you are configuring the virtual host.
2. Click **Add virtual-host** next to the **virtual-host** object.
3. Enter a host name or IP address as the **name** of this **virtual-host**.
4. Set **admin** to **enabled**.
5. Click **Create**. The **virtual-host** object and properties appear.



Configure cluster\box 1\interface eth0\ip a\web-service\virtual-host host1

* name	host1 (host name or n.n.n.n)
* admin	enabled (Resource is active)
applications-directory	webapps
web-app-config	Add web-app-config
role-mapping	Add role-mapping
access-logging	Configure

6. Enter the name of the **applications-directory** on which you are placing the WAR file for this virtual host. By default, this is **webapps**.

To configure a web-app-config object:

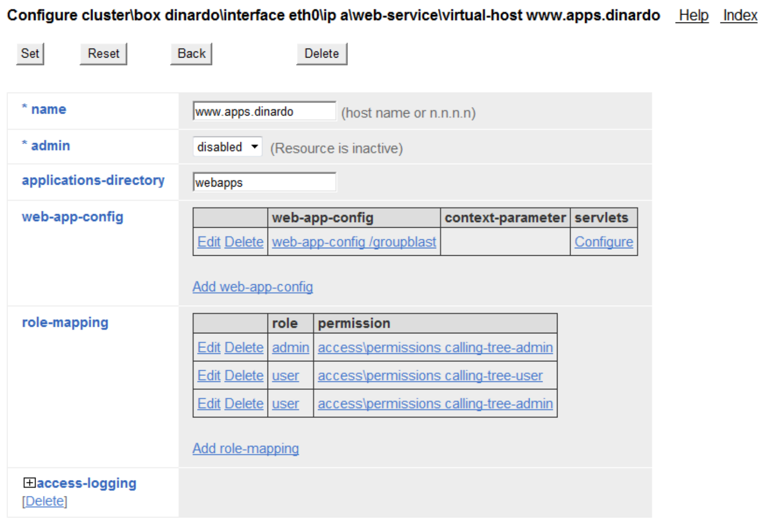
1. Click **Add web-app-config**. It is here that you configure the web application running on this virtual host.
2. Enter the **path** to where this application is being deployed.
3. Click **Create**.
4. Click **Add context-parameter**. A context-parameter is an application-level configuration property.
5. Enter the **name** of the **context-parameter**.
6. Enter the **value** of the **context-parameter**.
7. Click **Add servlets**. For information about servlets, see http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html.
8. Enter a **name** for this servlet.
9. Click **Add init-parameter**.
10. Enter a **name** for this **init-parameter**.
11. Enter a **value** for this **init-parameter**.
12. Click **Create**. Do this for as many **init-parameters** you are creating for this servlet.
13. Click **Set** when you are done adding **init-parameters**.
14. Click **Create**. Click **Set**. You are returned to the **virtual-host** object.

To configure a role-mapping object:

1. Click **Add role-mapping**. This object assigns application roles configured for security to permissions specified in the **access > permissions** configuration.

   For information on configuring access permissions on the OS-E, see Configuring Access in the *Oracle Communications* OS-E Management Tools Guide.
2. Enter a name for the **role** to assign this permission. For information about roles, see http://docs.oracle.com/javaee/6/tutorial/doc/gijrp.html.
3. Select a pre-configured **permission** from the drop-down box to assign to the role. Click Create if no permissions have been created. Click Edit if you want to edit an existing permission.
4. Click **Create**. You are returned to the **virtual-host** object.

   The following is an example of a properly configured **role-mapping** object.



Configure cluster\box dinardo\interface eth0\ip a\web-service\virtual-host www.apps.dinardo Help Index

Set Reset Back Delete

* name	www.apps.dinardo (host name or n.n.n.n)
* admin	disabled (Resource is inactive)
applications-directory	webapps
web-app-config	see table below; Add web-app-config
role-mapping	see table below; Add role-mapping
⊞access-logging [Delete]	

	web-app-config	context-parameter	servlets
Edit Delete	web-app-config /groupblast		Configure

	role	permission
Edit Delete	admin	access\permissions calling-tree-admin
Edit Delete	user	access\permissions calling-tree-user
Edit Delete	user	access\permissions calling-tree-admin

To configure access-logging:

1. Click **Configure** next to **access-logging**. This object configures a log file to record all requests and responses to and from this virtual host.
2. Set **admin** to **enabled**.
3. Enter the **directory** on which to write the log files.
4. Enter the **prefix** of the file name for the logging files.
5. Select the **pattern**, or format, to use to log the requests. This can either be:

- **common**—A basic common format.
- **combined**—A combined format that extends the common format.

  See http://tomcat.apache.ort/tomcat-5.5-doc/config/valve.html#Access_Log_Valve for more information on patterns.

6. Set **buffered** to **true** if you want log messages to be buffered before writing. Leave as **false** (the default) if you do not.
7. Enter the **maximum-file-age**. This is the maximum number of days to keep the log in the directory before deleting. The default is **7**. The minimum is **0** and the maximum is **4294967296**.
8. Click **Set**. Update and save the configuration.

Five status show actions have been created to provide information regarding virtual hosts.

The **show web-services-virtual-hosts** action provides information about all virtual hosts configured on the OS-E.

```
NNOS-E>show web-services-virtual-hosts

name                    state                   applications-directory
----                    -----                   ----------------------
host1                     STARTED                    webapps
```

Field	Description
name	The name of the virtual host.
state	The state of the virtual host.
applications-directory	The directory where this virtual host's WAR files are located.

The **show web-services-virtual-host-application-parameters** action provides information about application context parameters.

```
NNOS-E>show web-services-virtual-host-application-parameters

name            path   context-parameter-name context-parameter-value
----             ----  ---------------------- -----------------------
m5apps.com  /citi  codec               pcmu
m5apps.com  /citi  flashVersion        1100
m5apps.com  /sim   contextConfigLocation /WEB-
INF/applicationContext.xml
m5apps.com  /sim    default.operator.sipUrl
sip:*17813284400@foo;postd=0
m5apps.com  /sim    default.timeOut.response 45
```

```
m5apps.com  /sim    default.voiceMail.sipUrl sip:*17813284444@foo
m5apps.com  /sim    Extract.sipUrl.replacement sip:*$0@foo
m5apps.com  /sim    Extract.user.group  2
```

Field	Description
name	The name of the virtual host.
path	The path where this web application is deployed.
context-parameter-name	The name of the context-parameter for this servlet.
context-parameter-value	The value of the context-parameter for this servlet.

The **show web-services-virtual-host-applications** action provides information about all web applications configured on each virtual host on the OS-E.

```
NNOS-E>show web-services-virtual-host-applications

name          path     display-name  state    applications-directory
available
----         ----     ------------  -----     ----------------       -
-----
davisapps.com                    STARTED  /cxc_common/webapps    true
davisapps.com /acme    Acme Packet   STARTED  /cxc_common/webapps
true
davisapps.com /axa     Acme Packet   STARTED  /cxc_common/webapps
true
davisapps.com /group             STARTED  /cxc_common/webapps    true
davisapps.com /rtp     Acme Packet   STARTED  /cxc_common/webapps
true
davisapps.com /web     Acme Packet   STARTED  /cxc_common/webapps
true
```

Field	Description
name	The name of the virtual host.
path	The path where this web application is deployed.
display-name	The display name used within the web application.
state	The state of the web application.
applications-directory	The directory where this virtual host's WAR files are located.
available	The availability of the web application.

The **show web-services-virtual-host-application-servlets** action provides information on servlets configured for all web applications on each virtual hosts on the OS-E.

```
NNOS-E>show web-services-virtual-host-application-servlets

name           path           servlet        state          available
----           ----           -------        -----          ---------
davisapps.acmepacket.com /acme-packet    ConfigServlet   STARTED
true
davisapps.acmepacket.com /acme-packet    default        STARTED
true
davisapps.acmepacket.com /acme-packet    jsp           STARTED
true
davisapps.acmepacket.com /axa-tech      ConfigServlet   STARTED
true
```

```
davisapps.acmepacket.com /axa-tech      default       STARTED
true
davisapps.acmepacket.com /rtpstats   Acme Packet RTP Stats STARTED
/cxc_common/webapps true
davisapps.acmepacket.com /webphone   Acme Packet Web Phone STARTED
/cxc_common/webapps true
```

Field	Description
name	The name of the virtual host.
path	The path where the web application is deployed.
servlet	The name of the servlet.
state	The state of the servlet.
available	The availability of the servlet.

The **show web-services-virtual-host-application-servlets-parameters** action provides information on servlet parameter settings for each web applications on each virtual host on the OS-E.

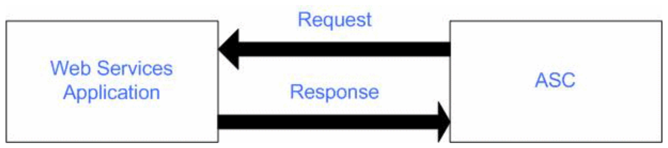
```
NNOS-E>show web-services-virtual-host-application-servlet-parameters

name            path            servlet         init-parameter-name
init-parameter-value
----            ----            -------         ------------------- -
-------------------
davisapps.acmepacket.com /axa-tech       ConfigServlet   ascBaseUrl-
disabled https://davis:8443
davisapps.acmepacket.com /axa-tech      ConfigServlet   rtmpBaseUrl
rtmp://davis.acmepacket.com/live
davisapps.acmepacket.com /axa-tech      ConfigServlet   uriFormat
sip:{0}@davis.acmepacket.com
davisapps.acmepacket.com /groupblast   IGroupBlastService adminRole
admin
davisapps.acmepacket.com /webphone       ConfigServlet   ascBaseUrl-
disabled https://davis:8443
davisapps.acmepacket.com /webphone      ConfigServlet   rtmpBaseUrl
rtmp://davis.acmepacket.com/live
davisapps.acmepacket.com /webphone      ConfigServlet   uriFormat
sip:{0}@davis.acmepacket.com
```

Field	Description
name	The name of the virtual host.
path	The path where the web application is deployed.
servlet	The name of the servlet.
init-parameter-name	The servlet’s init-parameter name.
init-parameter-value	The servlet’s init-parameter value.

2 Using ASC Callouts

Web Service Callouts

The Net-Net ASC supports web service callouts. A callout is when the ASC initiates contact with the web service client. Web service callouts are only supported in WSDL.



The ASC API supports two uses of callouts.

- External policy service—Sends policies when the ASC processes SIP messages
- External event service—Sends event notifications

External Policy Service

The external policy service sends a request to the web services application whenever the ASC is processing a SIP message. The web services application examines information about the SIP message and based on that information, returns the policy that it wants applied to the SIP message.

The WSDL request name is getAuthSessionPolicy.

Policies are configured and applied on the ASC in a specific order. The following is the hierarchy of session-config and normalization application:

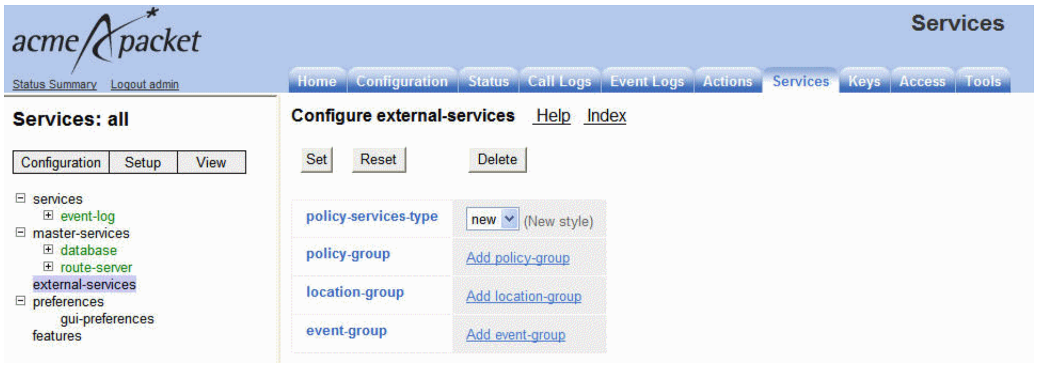
- default-session-config
- policy
- server inbound session-config
- server inbound normalization
- dial-plan/registration-plan > normalization
- dial-plan/registration-plan > arbiter > session-config
- dial-plan/registration-plan > route normalization
- dial-plan/registration-plan > route > session-config
- Policy sent from the web services application to the ASC via the getAuthSessionPolicy request
- server outbound session-config
- server outbound normalization
- server outbound normalization session-config

Configuring External Policy Service

To configure the ASC so that the external policy service works properly, you must configure a **policy-group** with a **policy-service**. Then, you must configure an **authorization** policy.

To configure policy-group and policy-service objects:

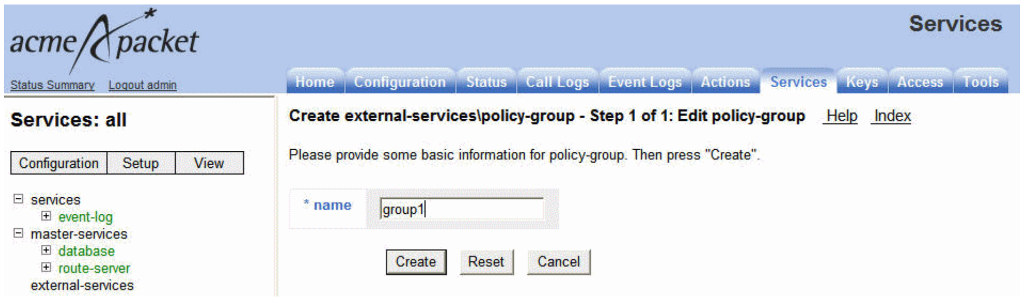
1. Click the **Services** tab and select **external-services**.
2. Select **new** from the **policy-services-type** drop-down box.



Configure external-services

policy-services-type	new (New style)
policy-group	Add policy-group
location-group	Add location-group
event-group	Add event-group

3. Click **Set**.
4. Click **Add policy-group**.
5. Enter a **name** for the policy-group you are creating.



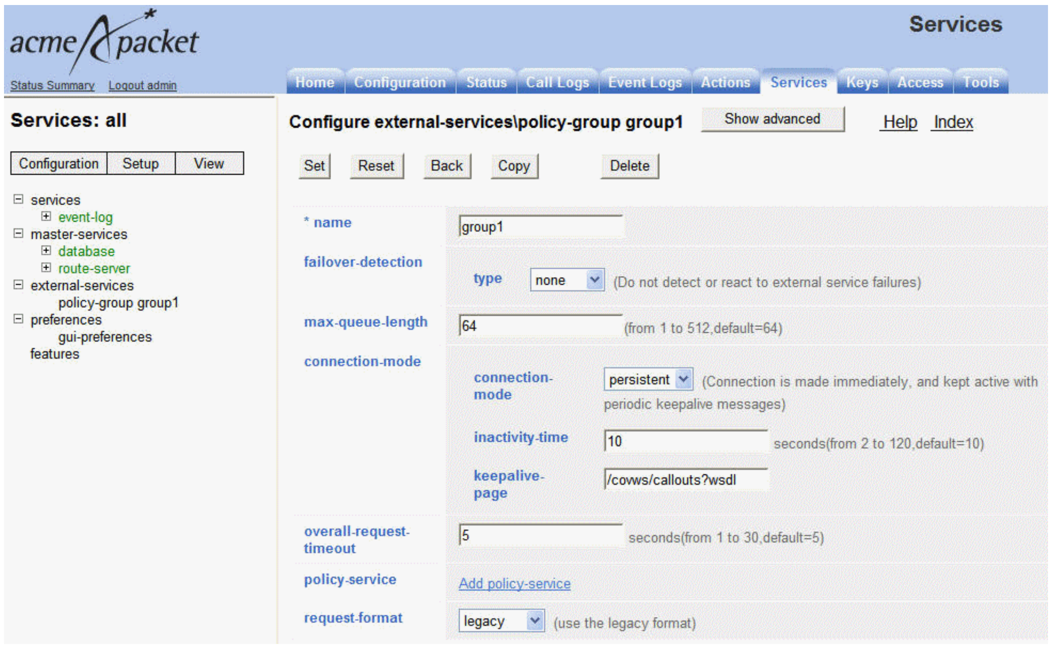
Create external-services\policy-group - Step 1 of 1: Edit policy-group

Please provide some basic information for policy-group. Then press "Create".

* name: group1

6. Click **Create**.
7. **failover-detection**—Leave this value at **none**, the default. The ASC performs no failover detection. If a request is not serviced, the system continues to send requests until a configured timeout value is reached or the request is manually withdrawn.
8. **max-queue-length**—Leave this value at **64**, the default. This is the maximum number of WSDL requests that can be queued for a policy group (awaiting assignment to a server). If the queue grows to this number, subsequent requests are rejected, with the result "queue-clipped," until the queue drops below this level.
9. **connection-mode**—Specify the manner in which connections between the ASC and WSDL client are established and maintained. The default value is persistent **10 /covws,callouts?wsdl**. The valid values are:
   - persistent [*seconds*][*page*]—Connections are initiated at boot time, and maintained using periodic keepalives. Specify an inactivity timeout, between 2 and 120 seconds, and a keepalive page.
   - lingering—Connections are made on demand, then linger until broken by the remote server.
   - transient—Connections are made on demand, then broken when a response is received.

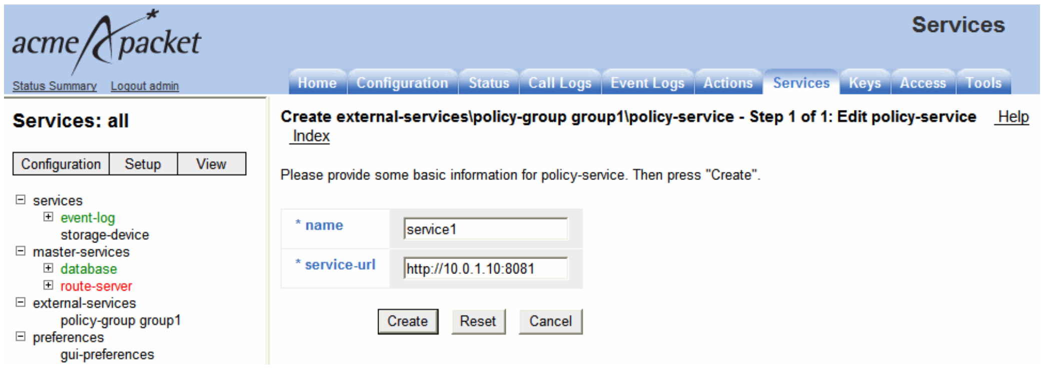
10. **overall-request-timeout**—Leave this value at **5**, the default. This specifies the number of seconds a request can remain in the queue for a policy server before it is timed out by the ASC.
11. Click **Set**.



Configure external-services\policy-group group1

* name	group1	
failover-detection	type: none	(Do not detect or react to external service failures)
max-queue-length	64	(from 1 to 512,default=64)
connection-mode	connection-mode: persistent	(Connection is made immediately, and kept active with periodic keepalive messages)
	inactivity-time: 10	seconds(from 2 to 120,default=10)
	keepalive-page: /cows/callouts?wsdl	
overall-request-timeout	5	seconds(from 1 to 30,default=5)
policy-service	Add policy-service	
request-format	legacy	(use the legacy format)

12. Click **Add policy-service**.
13. Enter a **name** for the policy-service.
14. Enter the **service-url**. This is the web service client's endpoint URL.



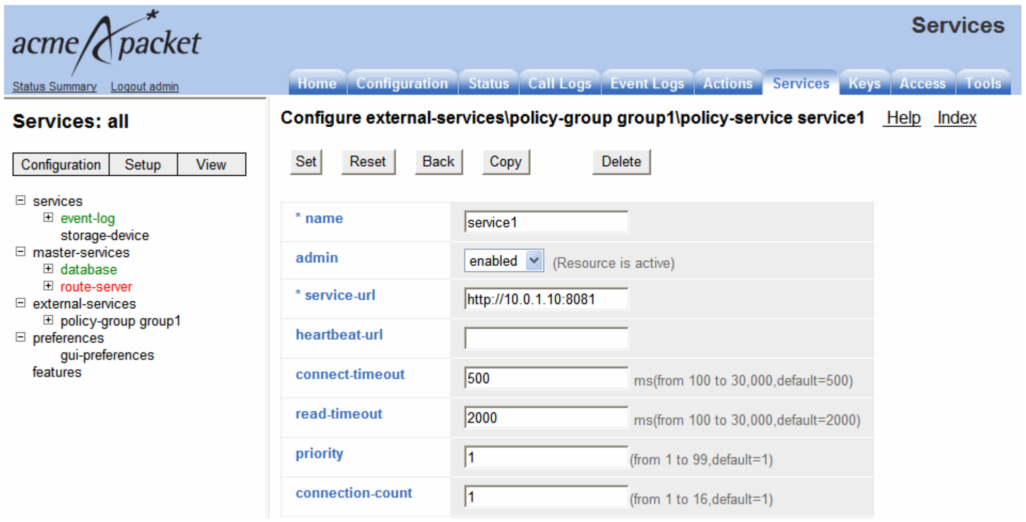
Create external-services\policy-group group1\policy-service - Step 1 of 1: Edit policy-service

Please provide some basic information for policy-service. Then press "Create".

* name	service1
* service-url	http://10.0.1.10:8081

15. Click **Create**.
16. **admin**—Leave this **enabled**, the default. This enables this policy service for use.
17. **connect-timeout**—Leave this value at **500**, the default. This specifies the length of time, in milliseconds, that the ASC allows to complete a connection to the external policy service before cancelling the request.
18. **read-timeout**—Leave this value at **2000**, the default. This specifies the length of time, in milliseconds, that the ASC waits for a response from the external policy service before cancelling the request.
19. **priority**—Leave this value at **1**, the default. This specifies the priority of this server within the policy group. The lower the number, the higher the priority.

20. **connection-count**—Leave this value at **1**, the default. This specifies the number of simultaneous connections allowed to this server.



Configure external-services\policy-group group1\policy-service service1

* name	service1
admin	enabled (Resource is active)
* service-url	http://10.0.1.10:8081
heartbeat-url	
connect-timeout	500 ms(from 100 to 30,000,default=500)
read-timeout	2000 ms(from 100 to 30,000,default=2000)
priority	1 (from 1 to 99,default=1)
connection-count	1 (from 1 to 16,default=1)

21. Click **Set**. Update and save the configuration.

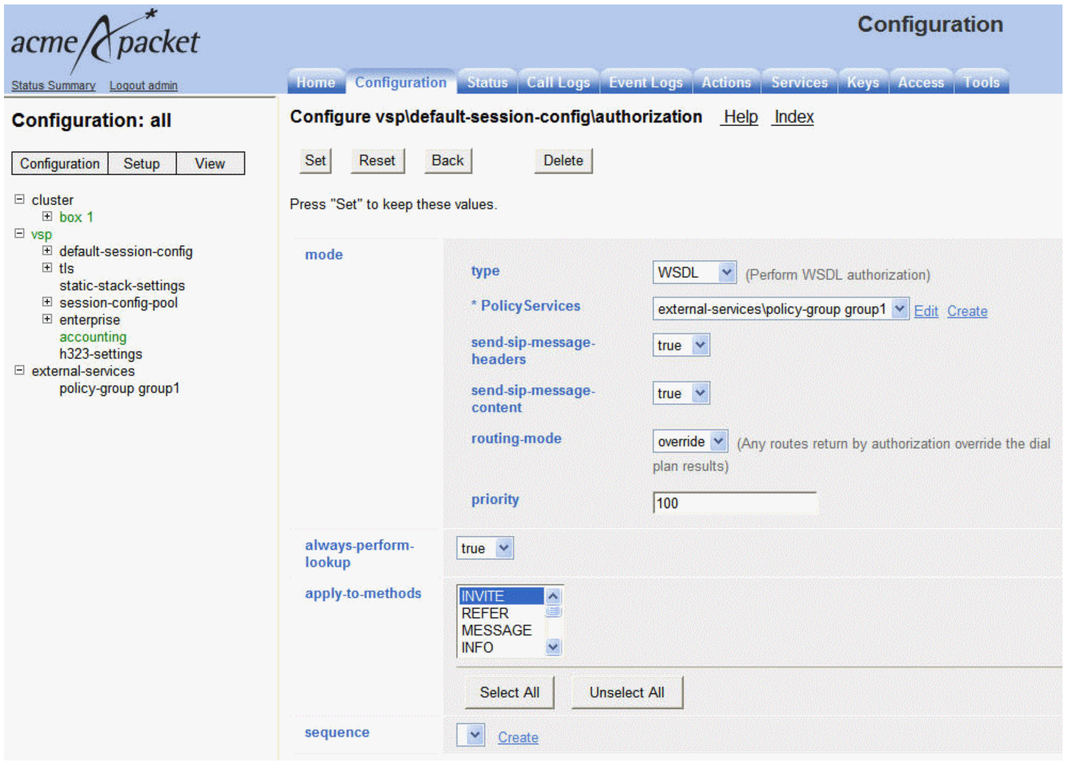
To configure the authorization policy object:

1. Click the **Configuration** tab and select **vsp**.
2. Select either **default-session-config** or **session-config-pool > entry**. (If you configure **entry**, you must reference it.)
3. Click **Configure** beside the **authorization** property.
4. **mode**—Select **WSDL** from the drop-down box. The ASC sends the request for authorization data retrieval to the external services policy server specified in the policy-group object. The default is **None**.

   When you select WSDL, the following properties appear.

   - **PolicyServices**—Select the previously configured **policy-group** object from the drop-down box. If it is not there, you can create it by clicking **Create** and entering the path to the policy group.
   - **send-sip-message-headers**—Select **true**. This allows SIP message headers to be sent to the web services client.
   - **send-sip-message-content**—Select **true**. This allows SIP message content to be sent to the web services client.
   - **routing-mode**—Leave this set to **override**, the default. This means any routes returned by authorization override the dial plan results.
   - **Priority**—Leave this set to **100**, the default.
5. **always-perform-lookup**—Leave this set to **true**, the default. This means the ASC retrieves authorization data regardless of other configuration settings.

6. **apply-to-methods**—Select the SIP messages to which the ASC applies authorization processing. The default is INVITE.



Configure vsp\default-session-config\authorization

Press "Set" to keep these values.

mode	type	WSDL	(Perform WSDL authorization)
	* PolicyServices	external-services\policy-group group1	
	send-sip-message-headers	true	
	send-sip-message-content	true	
	routing-mode	override	(Any routes return by authorization override the dial plan results)
	priority	100	
always-perform-lookup		true	
apply-to-methods		INVITE, REFER, MESSAGE, INFO	
sequence			

7. Click Set. Save and activate the configuration.

External Event Service

The external event service sends, or “pushes,” notifications of all events generated by the ASC to a web services application. These events are all available as SNMP traps, however, this service allows you to receive events without having to use SNMP.

The WSDL request name for this service is processEvent.

Using Cometd 2.0, the OS-E supports channels, a dynamic, path-like hierarchy describing the topic of an event. Third-party applications can subscribe to events on specific channels and, thus, narrow the scope of events to process.

In releases prior to 36.0m5, users could subscribe only to specific, hard-coded, request-ID based channels. By default, the OS-E still emits the legacy channels, however, you can disable them if they are no longer used. To stop the OS-E from using the legacy channels, set the **eventpush-service > legacy-events** property to **disabled**.

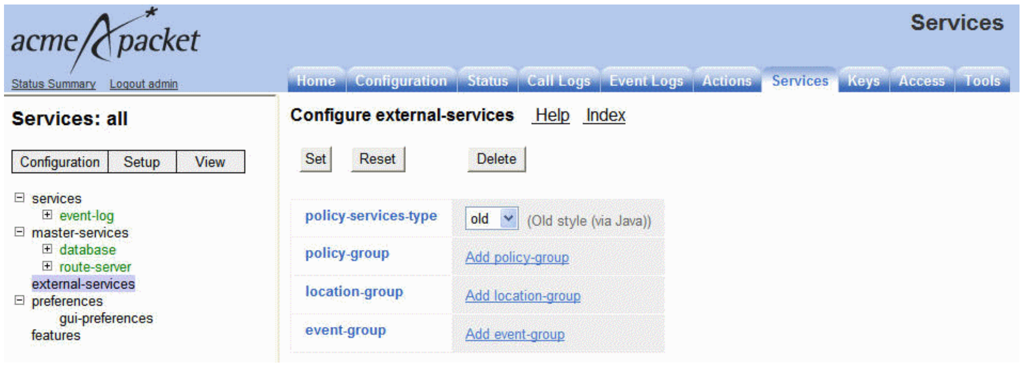
There are two ways to enable web services event processing, configuring **external-event-groups** or via the **dynamic-event-service** action.

Configuring External Event Service

To configure the ASC so that the external event service works properly, you must configure an **event-group** with an **event-service**. Then reference the **event-group** in the **vsp > external-event-group** object. You must also set the **third-party-call-control > status-events** property to both.

To configure event-group and event-service objects:

1. Click the **Services** tab and select **external-services**.
2. Click **Add event-group**.

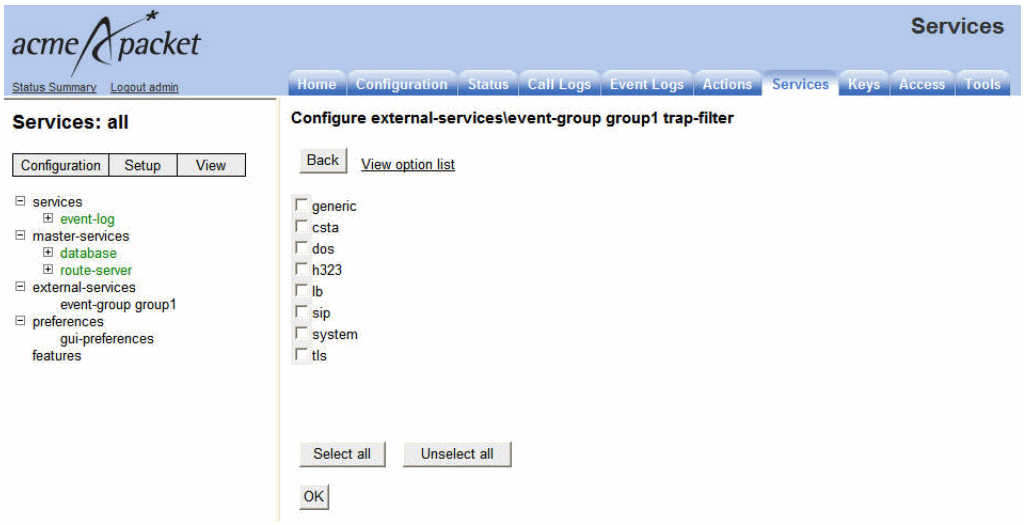


Configure external-services

policy-services-type	old (Old style (via Java))
policy-group	Add policy-group
location-group	Add location-group
event-group	Add event-group

3. Enter a **name** for the event-group and click **Create**.
4. Click **Edit trap-filter**. A list of categories appears. If you don't select any categories, all events are sent.

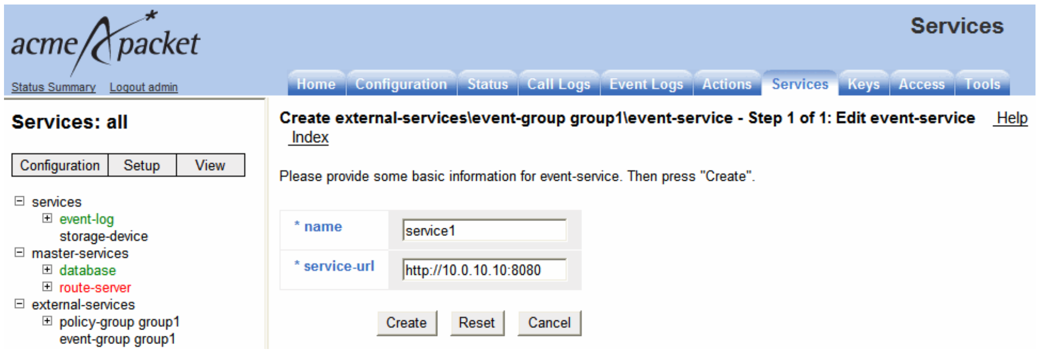
   To receive events only pertaining to calls, set **trap-filter** to **csta**.



Configure external-services\event-group group1 trap-filter

- [ ] generic
- [ ] csta
- [ ] dos
- [ ] h323
- [ ] lb
- [ ] sip
- [ ] system
- [ ] tls

5. Click **OK**.
6. Click **Add event-service**.
7. Enter a **name** for the event-service.
8. Enter a **service-url** for this event-service. This is the web services client endpoint.



Create external-services\event-group group1\event-service - Step 1 of 1: Edit event-service

Please provide some basic information for event-service. Then press "Create".

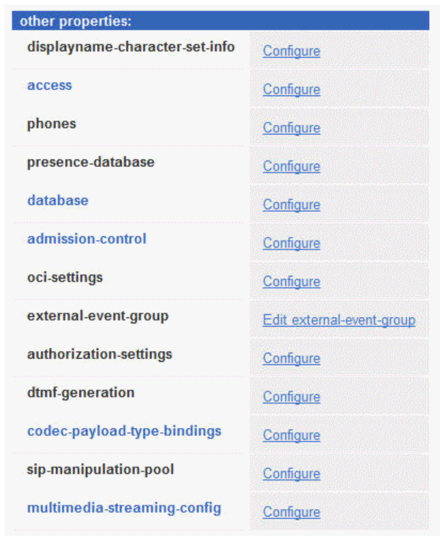
* name	service1
* service-url	http://10.0.10.10:8080

9. Click **Create**. Update and save the configuration.

To reference the event-group to the vsp > external-policy-group:

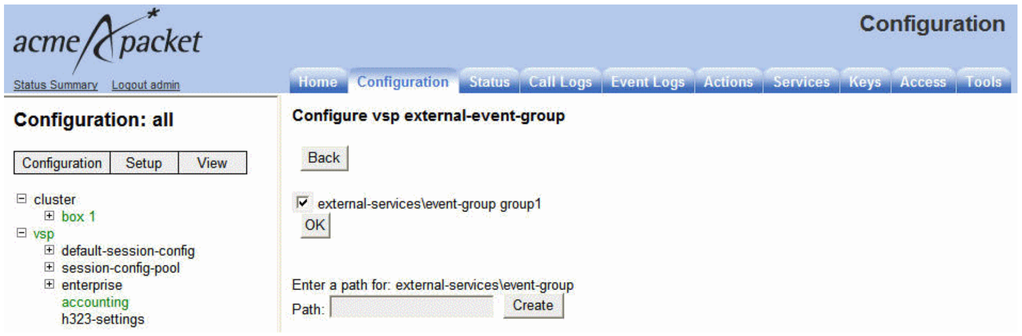
1. Click the **Configuration** tab and select **vsp**.
2. Click **Edit external-event-group** next to the **external-event-group** property.

   Note: This is an Advanced property. You must click the **Show advanced** button at the top of the page to see this property.



other properties:	
displayname-character-set-info	Configure
access	Configure
phones	Configure
presence-database	Configure
database	Configure
admission-control	Configure
oci-settings	Configure
external-event-group	Edit external-event-group
authorization-settings	Configure
dtmf-generation	Configure
codec-payload-type-bindings	Configure
sip-manipulation-pool	Configure
multimedia-streaming-config	Configure

3. Select the previously created **event-group** you are referencing. A list of all event-groups configured on the box appear. If no event-groups have been created you can create one.



Configure vsp external-event-group

☑ external-services\event-group group1

Enter a path for: external-services\event-group

Path:

4. Click **OK**. Update and save the configuration.

To receive call-control events:

1. Click the **Configuration** tab and select **vsp**.
2. Select either the **default-session-config** or the **session-config-pool > entry** property.
3. Click **Configure** next to **third-party-call-control**. The **third-party-call-control** object appears.
4. Select **both** from the **status-events** drop-down box.
5. Click **OK**. Update and save the configuration.

Executing dynamic-event-service

A web application can register itself by using the web service REST and SOAP clients to call the **dynamic-event-service register** action. Using the **dynamic-event-service keepalive** action you can keep current registrations alive, and via the **dynamic-event-service unregister** action, the web application can unregister itself. The action syntax is:

```
dynamic-event-service register <endpoint> [channels] [xml-format]
[time-to-live] [connect-timeout] [read-timeout] [character-set]
[request-style] [include-channels-in-events]

dynamic-event-service keepalive <registration-id>

dynamic-event-service unregister <registration-id>
```

Valid arguments for the **dynamic-event-service register** action are:

- <*endpoint*>—The application endpoint that receives events.
- [*channels*]—The channels for which the endpoint is getting events.
- [*xml-format*]—The XML format used by this server. This can be either **simplified** (the default) or **legacy**.
- [*time-to-live*]—The time to live, in minutes, for the keepalive on this registration. The default is untilRestart, meaning the registration stays alive until the system is restarted.
- [*connect-timeout*]—The connect timeout, in milliseconds, for the endpoint. The default is **1000**.
- [*read-timeout*]—The read timeout, in milliseconds, for the endpoint. The default is **1000**.
- [*character-set*]—The character set to use when forming requests to this endpoint. This can be **utf-8** (the default) or **iso-8859-1**.
- [*request-style*]—The style to use when sending events to this listener. This can be **SOAP** (the default), **XML**, or **JSON**.
- [*include-channels-in-events*]—Whether channels are included in events. This is **enabled** by default.

Once an application has registered itself to receive events, you can view information about the registration via the **show dynamic-event-services** status provider.

```
NNOS-E>show dynamic-event-services

       endpoint: 10.0.0.10
registration-id: d710c03c-70b3-454d-9ee2-c1b6f60dd5b7
        created: 12:10:20.857000 Thu 2012-03-01
   time-to-live: untilRestart seconds
 last-keepalive: 12:10:20.857000 Thu 2012-03-01
       channels:
connect-timeout: 1000 ms
   read-timeout: 1000 ms
  character-set: utf-8
  request-style: soap
       requests: 0
       failures: 0
```

Field	Description
endpoint	The application endpoint being called out.
registration-id	The registration identifier.
created	The date and time this registration was created.

Field	Description
time-to-live	The configured time to live, in minutes, on this registration.
last-keepalive	The date and time that the last keep alive was received.
channels	The channels for which the endpoint is getting events.
connect-timeout	The configured connect timeout, in milliseconds, for the endpoint.
read-timeout	The configured read timeout, in milliseconds, for the endpoint.
character-set	The character set used when forming requests to this endpoint. This can be either utf-8 or iso-8859-1.
request-style	The style used when sending events to this listener. This could be either XML, JSON, or SOAP.
requests	The number of requests that have been made to the endpoint.
failures	The number of requests that have failed to reach the endpoint.

The **session-config > event-settings** object configures events and user-specified event channels on the OS-E.

The **event-settings > channel** property configures user-specified channels on the OS-E. Each time the OS-E needs to emit an event for a session, the event configuration component dynamically regenerates all of the appropriate channels specified by the user based on the this property.

This property consists of an array of strings used to compose channel paths. These strings can contain named-variables that are replaced with a value extracted from the current state of the session. Named-variables must start and end with percent (%) characters.

Named variables can be added to sessions on the OS-E 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 Oracle Communications OS-E Objects and Properties Reference Guide.

Named-variables can also be added via the **named-variable-add** action. For information on this action, see the Named Variable Actions section of this guide.

Under the **event-settings** object you can insert named-variables into events. This is done via the **named-variable-entry** property.

NOTE: In order for named-variables to work in either the event-settings > channel or named-variable-entry properties, named-variables must be configured elsewhere on the OS-E, either within the session-config > named-variables object or via the named-variables-add action.

The following example shows adding one variable called **my-variable** with a value of **my-value** to the **default-session-config > named-variable** object.

```
NNOS-E>config vsp
config vsp>config default-session-config
config default-session-config>config named-variables
config named-variables>config named-variable my-variable
Creating 'named-variable my-variable'
config named-variable my-variable>set value my-value
config named-variable my-variable>return
config named-variables>return
```

This next example shows the **event-settings** object configured with a **channel** and **named-variable-entry** that correspond with the **session-config** > **named-variables** configuration in the above example.

Specific-channel-name is a static channel name and the OS-E does not attempt to look up the value of this string. Because it is enclosed in percentage signs, the **/%my-variable%** value signifies a named-variable channel name. The **named-variable-entry** property's **my-variable my-variable-name** value represents the inclusion of the named-variable configured in the first example in the contents of the events. **My-variable-name** is the name that is shown inside the events for this variable.

```
NNOS-E>config vsp
config vsp>config default-session-config
config default-session-config>config event-settings
config event-settings>set channel /specific-channel-name
config event-settings>set channel /%my-variable%
config event-settings>set named-variable-entry my-variable my-
variable-name
config event-settings>return
```

Here is an example of an event for a session that has the above configuration. Note the two channels: specific-channel-name and my-value. There is also an <nvpData> entry (which stands for named-value-pair) for my-variable-name and my-value.

```
<Event box="1" process="SIP" timestamp="16:41:26.000001 Wed 2012-03-
21" channel="">
<object>
  <CallCreatedEvent>
   <callEvent>
    <CallEvent>
     <requestID/>
     <handle>15217493</handle>
     <sessionID>343475565090092753</sessionID>
     <callID>1-11664@10.33.5.10</callID>
     <to>sip:service@10.33.80.65:5060</to>
     <from>sip:sipp@10.33.5.10:6021</from>
     <nvpData>
      <name>my-variable-name</name>
      <value>my-value</value>
     </nvpData>
    </CallEvent>
   </callEvent>
  </CallCreatedEvent>
</object>
<channels>/specific-channel-name</channels>
<channels>/my-value</channels>
<userData>0x00000000</userData>
</Event>
```

The same named variables can be used to configure both the **channel** and **named-variable-entry** properties.

NOTE: Named variables used in the **channel** property must start and end with percentage (%) characters to work properly.

These variables can be broken down into three types: event, session, and call, in-leg, and out-leg.

Event named variables are derived from the current event being published. The object of these variables can be any of the events the OS-E can generate. To view the full list of OS-E events, see Events in the web services home page's REST documentation.

You can retrieve a property in the event object by specifying **$event.<*property*>** where <*property*> is the name or alias of a property in the event object being generated.

For example, for a call control event with a requestID of 123456, specifying **/req/%$event.requestID%** results in the channel **/req/123456** being created.

Specifying **/event-name/%$event._alias%** results in the channel **/event-name/call-terminated** being created for call-terminated events.

Available variables for the event class are:

- **$event**—Event-based named variables.
- **$event._alias**—Alias for a generated event.

Session named variables are derived from the current session for the events being published. Available variables for this class are:

- **$session-session-id**—Session ID for this session.
- **$session.request-id**—Request ID for this session.
- **$session.caller-id**—Caller ID for this session.
- **$session.diversion-header**—Diversion-header for this session.
- **$session.pcharging-vector**—P-charging-vector for this session.
- **$session.digest-realm**—Digest realm for this session.
- **$session.source-lnp**—Source-lnp for this session.
- **$session.destination-lnp**—Destination-lnp for this session.

Call, in-leg, and out-leg named variables are derived from the call legs of the current session for events being published. Call events are generated on a specific leg. Therefore the call variables provide access to the leg on which the event is being generated.

Each call session has one or two legs, deemed the in-leg and out-leg based on call direction. In-leg variables use the in-leg for the session that generated this event and out-leg variables use the out-leg for the session that generated this event.

Available variables for these classes are:

- **$call.request-id**—Request ID for this call.
- **$call.to**—To: URI for this call.
- **$call.to.user**—User portion of the To: URI for this call.
- **$call.to.host**—Host portion of the To: URI for this call.
- **$call.from**—From: URI for this call.
- **$call.from.user**—User portion of the From: URI for this call.
- **$call.from.host**—Host portion of the From: URI for this call.
- **$call.request**—Request: URI for this call.

- **$call.request.user**—User portion of the Request: URI for this call.
- **$call.request.host**—Host portion of the Request: URI for this call.
- **$call.call-id**—Call-id for this call.
- **$call.to-contact**—Local endpoint for this call.
- **$call.to-contact.user**—User portion of the local endpoint for this call.
- **$call.to-contact.host**—Host portion of the local endpoint for this call.
- **$call.from-contact**—Remote endpoint for this call.
- **$call.from-contact.user**—User portion of the remote endpoint for this call.
- **$call.from-contact.host**—Host portion of the remote endpoint for this call.
- **$call.p-assert**—P-asserted-identity header for this call.
- **$call.p-assert-user**—User portion of the p-asserted-identity header for this call.
- **$call.p-assert-host**—P-asserted-identity header for this call.
- **$in-leg.request-id**—Request-id for the in-leg.
- **$in-leg.to**—To: URI for the in-leg.
- **$in-leg.to.user**—User portion of the To: URI for the in-leg.
- **$in-leg.to.host**—Host portion of the To: URI for the in-leg.
- **$in-leg.from**—From: URI for the in-leg.
- **$in-leg.from.user**—User portion of the From: URI for the in-leg.
- **$in-leg.from.host**—Host portion of the From: URI for the in-leg.
- **$in-leg.request**—Request: URI for the in-leg.
- **$in-leg.request.user**—User portion of the Request: URI for the in-leg.
- **$in-leg.request.host**—Host portion of the Request: URI for the in-leg.
- **$in-leg.call-id**—Call-id for the in-leg.
- **$in-leg.to-contact**—Local endpoint for the in-leg.
- **$in-leg.to-contact.user**—User portion of the local endpoint for the in-leg.
- **$in-leg.to-contact.host**—Host portion of the local endpoint for the in-leg.
- **$in-leg.from-contact**—Remote endpoint for the in-leg.
- **$in-leg.from-contact.user**—User portion of the remote endpoint for the in-leg.
- **$in-leg.from-contact.host**—Host portion of the remote endpoint for the in-leg.
- **$in-leg.p-assert**—P-asserted-identity header for the in-leg.
- $in-leg.p-assert.user—User portion of the p-asserted-identity header for the in-leg.
- **$in-leg.p-assert.host**—Host portion of the p-asserted-identity header for the in-leg.
- **$out-leg.request-id**—Request ID for the out-leg.
- **$out-leg.to**—To: URI for the out-leg.
- **$out-leg.to.user**—User portion of the To: URI for the out-leg.
- **$out-leg.to.from**—Host portion of the To: URI for the out-leg.
- **$out-leg.from**—From: URI for the out-leg.
- **$out-leg.from.user**—User portion of the From: URI for the out-leg.

- **$out-leg.from.host**—Host portion of the From: URI for the out-leg.
- **$out-leg.request**—Request: URI for the out-leg.
- **$out-leg.request.user—**User portion of the Request: URI for the out-leg.
- **$out-leg.request.host**—Host portion of the Request: URI for the out-leg.
- **$out-leg.call-id**—Call-id for the out-leg.
- **$out-leg.to-contact**—Local endpoint for the out-leg.
- **$out-leg.to-contact.user—**User portion of the local endpoint for the out-leg.
- **$out-leg.to-contact.host—**Host portion of the local endpoint for the out-leg.
- **$out-leg.from-contact**—Remote endpoint for the out-leg.
- **$out-leg.from-contact.user**—User portion of the remote endpoint for the out-leg.
- **$out-leg.from-contact.host**—Host portion of the remote endpoint for the out-leg.
- **$out-leg.p-assert**—P-asserted-identity header for the out-leg.
- **$out-leg.p-assert.user**—User portion of the p-asserted-identity header for the out-leg.
- **$out-leg.p-assert.host—**Host portion of the p-asserted-identity header for the out-leg.

To configure channels on the OS-E:

1. Select the **Configuration** tab and click the **vsp > default-session-config** or **vsp > session-config-pool > entry** object.
2. Click the **event-settings** object.
3. Click **Edit channel**.
4. Enter the string to use to generate events for this session. Click **Add**. Click **OK**.
5. Click **Set**. Update and save the configuration.

To configure named-variable-entries on the OS-E.

1. Select the **Configuration** tab and click the **vsp > default-session-config** or **vsp > session-config-pool > entry** object.
2. Click the **event-settings** object.
3. Click **Add named-variable-entry**.
4. Enter a **variable** or select one from the drop-down list.
5. Click **Create**. You are returned to the **event-settings** object.
6. To give the variable a display-name, click **Edit** next to the variable name.
7. Enter the **display-name**. This is the name that will be displayed within the event instead of the actual named-variable name.
8. Click **Set**. Update and save the configuration.

Generating Event Messages

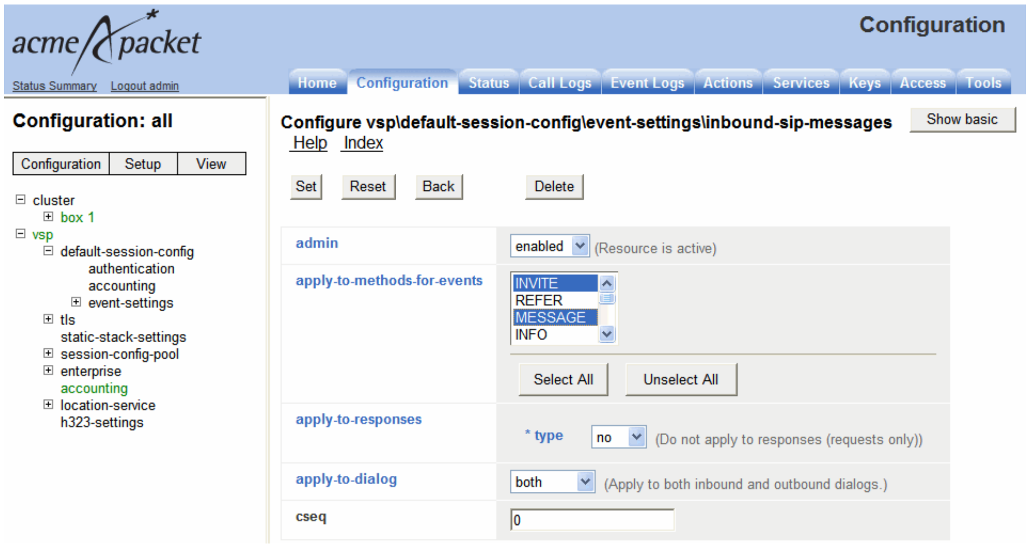
Two of the most common types of event messages that the ASC can generate are SIP event messages and call-control event messages. To enable the ASC to generate SIP event messages, see the following section. To work with call-control event messages, see Chapter 3, Configuring Events.

Sending SIP Event Messages

You can configure the ASC to send SIP message events when the ASC receives and transmits SIP messages. The **event-settings** > **inbound-sip-messages** and **outbound-sip-messages** objects configure the ASC to send SIP message events for incoming and outgoing SIP messages.

To configure the ASC to send SIP event messages:

1. Select the **Configuration** tab and click the **vsp** > **default- session-config** or **vsp** > **session-config-pool** > **entry** object.
2. Click the **event-settings** object.
3. Click **Configure** next to **inbound-sip-messages** to enable events for incoming SIP messages. Click **Configure** next to **outbound-sip-messages** to enable events for outgoing SIP messages.



Configure vsp\default-session-config\event-settings\inbound-sip-messages

admin	enabled (Resource is active)
apply-to-methods-for-events	INVITE, REFER, MESSAGE, INFO
apply-to-responses	* type: no (Do not apply to responses (requests only))
apply-to-dialog	both (Apply to both inbound and outbound dialogs.)
cseq	0

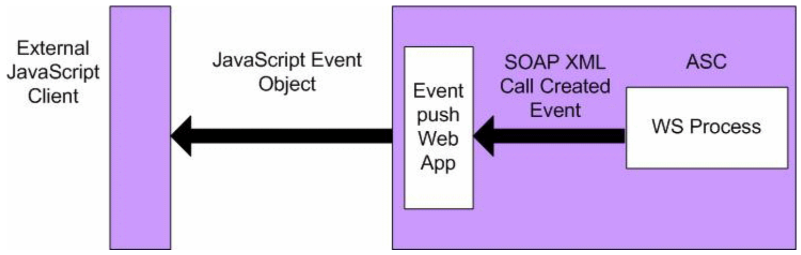
NOTE: **Inbound-sip-messages** and **outbound-sip-messages** are advanced properties. To see advanced properties, click the **Show advanced** button at the top of the window.

4. **admin**—Set to **enabled**.
5. **apply-to-methods-for-events**—Select the SIP methods you want the OS-E to create events for.
6. Click **Set**. Update and save the configuration.

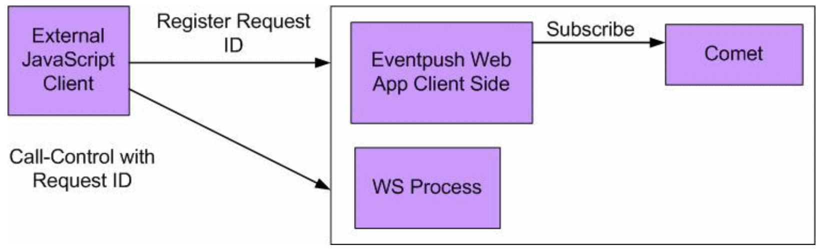
Eventpush Service

The ASC supports a web services application called eventpush service. Eventpush service is a solution which allows you to forward event information from the ASC to clients on external web applications which are unable to implement a SOAP/WSDL endpoint.

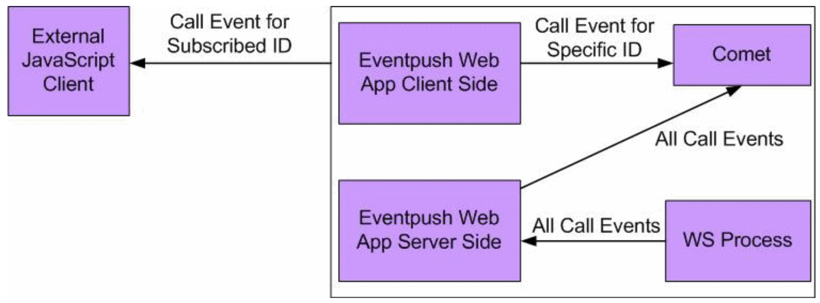
Eventpush service is configured as its own process within the ASC under the **eventpush-service** object.



Eventpush service supports a publish/subscribe interface using Cometd. There is a JavaScript API that wraps the Cometd technology. The customer application *subscribes* by indicating that it only wants to receive call events for calls with a specific requestID.


External JavaScript Client
Register Request ID
Call-Control with Request ID
Eventpush Web App Client Side
Subscribe
Comet
WS Process

The eventpush web application then *publishes*, or sends, only the events with that subscribed requestID.


External JavaScript Client
Call Event for Subscribed ID
Eventpush Web App Client Side
Call Event for Specific ID
Comet
All Call Events
Eventpush Web App Server Side
All Call Events
WS Process

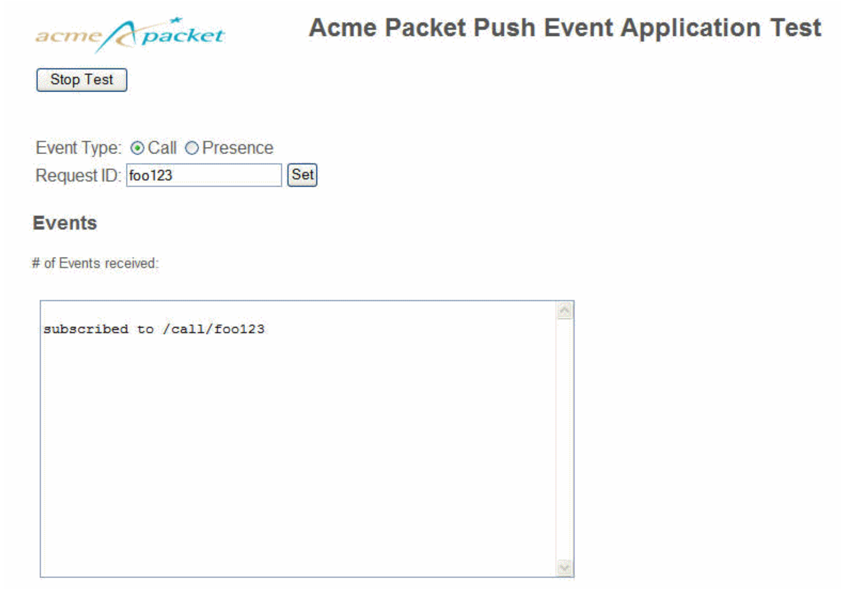
To enable cross-domain communication between the eventpush application and the customer web service application, the ASC's eventpush service DNS suffix must be the same as the customer web service application's.

To test the publish/subscribe interface, access the ASC eventpush service page. The URI for this page is:

http(s)://*ip*:*port*/cometapp/comet_test.html

Enter either **http** or **https**, the IP and port you have configured under the **eventpush-service** object.

Specify the requestID to which you are subscribing. This tells the ASC to publish only call events with that requestID.



Event Type: [✓ Call] Presence
Request ID: foo123

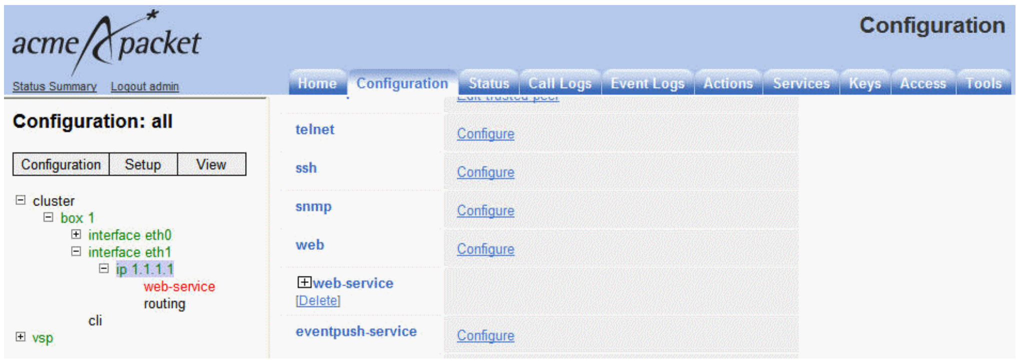
```
subscribed to /call/foo123
```

For more information on publish/subscribe technology, see http://en.wikipedia.org/wiki/Publish/subscribe.

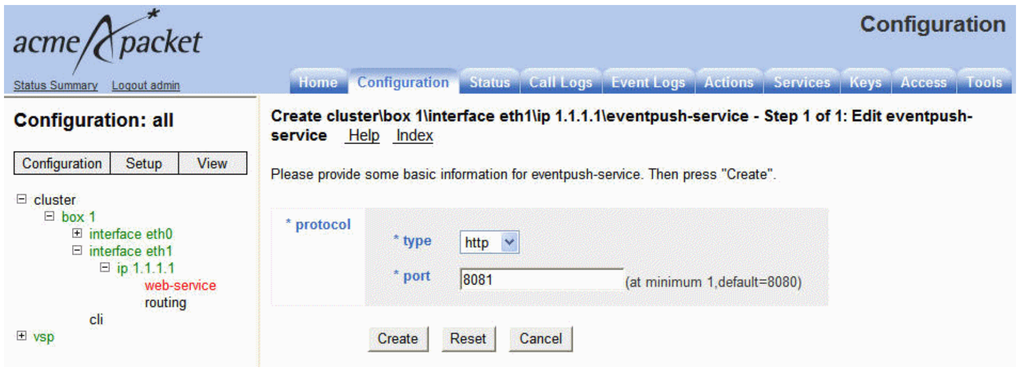
For more information on cometd technology, see http://cometdproject.dojotoolkit.org.

To configure eventpush-service:

1. Click the **Configuration** tab and select **Cluster**.
2. Select the **box**, **interface**, and **ip address** on which you want to configure the **eventpush-service**.
3. Click **Configure** next to **eventpush-service**.



4. Set the protocol **type** and **port** and click **Create**.

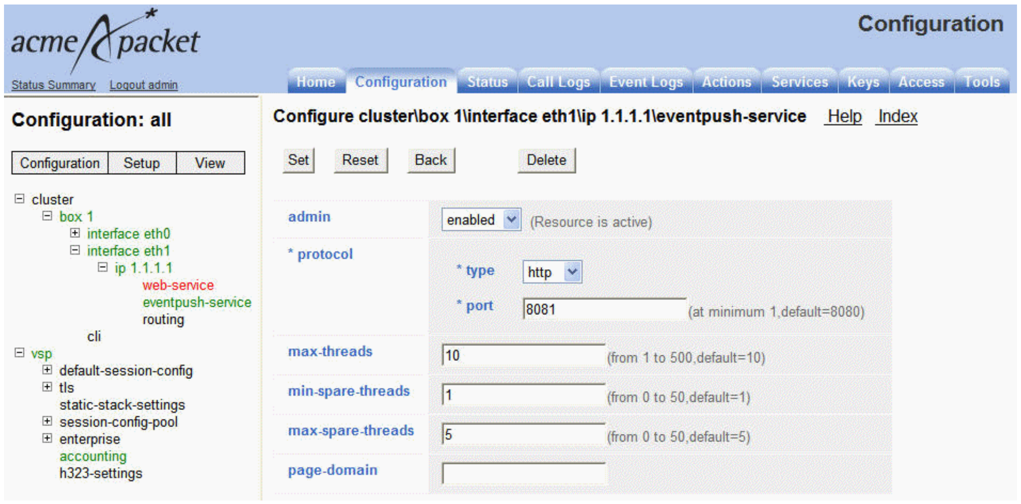


Create cluster\box 1\interface eth1\ip 1.1.1.1\eventpush-service - Step 1 of 1: Edit eventpush-service

Please provide some basic information for eventpush-service. Then press "Create".

* protocol
* type: http
* port: 8081 (at minimum 1,default=8080)

5. Set the **page-domain** to the domain name of the ASC.



Configure cluster\box 1\interface eth1\ip 1.1.1.1\eventpush-service

admin	enabled (Resource is active)
* protocol	* type: http * port: 8081 (at minimum 1,default=8080)
max-threads	10 (from 1 to 500,default=10)
min-spare-threads	1 (from 0 to 50,default=1)
max-spare-threads	5 (from 0 to 50,default=5)
page-domain	

6. Click **Set**. Update and save the configuration.

Two status providers provide information on the current set of active cometd channels.

The **show cometd-channel-summary** action provides a summary of channel information for the cometd server.

```
NNOS-E>show cometd-channel-summary

name                        subscriber-count
----                        ----------------
/**                         1
/call                       0
/call/to                    0
/call/to/019785551212       1
/cometd                     0
/cometd/meta                1
/meta                       0
/meta/connect               0
/meta/disconnect            0
/meta/handshake             0
/meta/subscribe             0
/meta/unsubscribe           0
```

Field	Description
name	The name of the channel.
subscriber-count	The number of subscribers on this channel.

The **show cometd-channel-detail** action provides more detailed channel information, specifically, on the subscribers to each of the channels.

Note that if a channel appears in the summary but not in the details, it means that the channel exists without any active cometd client subscriptions.

```
NNOS-E>show cometd-channel-details

name            remote-address  remote-port id             user-agent
----            --------------  ----------- --             ----------
/**             10.1.21.57      49804       372tj5ikmvga8ant2b6m2wcjs
Mozilla/5                                  .0 (Windows NT 6.1; WOW64)
AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.96
3.79 Safari/535.11
/call/to/019785551212 10.1.21.57      49728
21sxpszu2lkikc1pnadt0mdfzvg Mo                               zilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/
17.0.963.79 Safari/535.11
/cometd/meta     10.1.21.57      49804       372tj5ikmvga8ant2b6m2wcjs
Mozilla/5                                  .0 (Windows NT 6.1; WOW64)
AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.96
3.79 Safari/535.11
```

Field	Description
name	The name of the channel.
remote-address	The remote address for this subscriber.
remote-port	The remote port for this subscriber.
id	The identifier assigned internally by the OS-E for this publisher.
user-agent	The user agent the subscriber used to establish the session.

Two status providers have been added to provide information on the current set of active cometd subscribers.

The **show cometd-subscriber-summary** action provides high-level information about the subscribers.

```
NNOS-E>show cometd-subscriber-summary

remote-address  remote-port id          channel-count message-count
user-agent
--------------  ----------- --          ------------- ------------- -
---------
10.1.21.57    49728     21sxpszu2lkikc1pnadt0mdfzvg 1         0
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like
Gecko) Chrome/17.0.963.79 Safari/535.11
10.1.21.57    49804     372tj5ikmvga8ant2b6m2wcjs 2         0
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like
Gecko) Chrome/17.0.963.79 Safari/535.11
```

Field	Description
remote-address	The remote address for the subscriber.
remote-port	The remote port for the subscriber.
id	The identifier assigned internally by the OS-E for this publisher.
channel-count	The number of channels to which the subscriber is currently subscribed.
message-count	The number of messages a subscriber has currently been sent.
user-agent	The user agent the subscriber used to establish the session.

The **show cometd-subscriber-details** action provides more detailed information, specifically on the channels subscribed to by each subscriber.

Note that if a subscriber appears in the summary but not the details, it means that the subscriber exists without any active cometd channel subscriptions.

```
NNOS-E>show cometd-subscriber-details

remote-address  remote-port channel
--------------  ----------- -------
10.1.21.57      49728       /call/to/019785551212
10.1.21.57      49804       /**
10.1.21.57      49804       /cometd/meta
```

Field	Description
remote-address	The remote address for the subscriber.
remote-port	The remote port for the subscriber.
channel	The name of the channel.

3 ASC Call Control Action

Web Service Call Control

Many of the applications you can create via the Net-Net ASC will use the **call-control** action. This chapter describes how to use **call-control**, its parameters, as well as the results and event messages that are subsequently generated.

Identifying Calls and Sessions

When the Net-Net ASC creates calls, it uses several elements to identify specific calls and portions of calls. These unique markers are request IDs, session IDs, call leg handles, and SIP call-IDs.

For more information on which elements appear in what event messages and which are parameters for **call-control** actions, see

Request IDs

When creating new calls, an application identifies the endpoints involved using their SIP URIs. An application may also supply a request ID to the ASC. If it does supply a request ID, the ASC labels the resulting session with that request ID. This ID is returned in the subsequent responses to the request and any events pertaining to that session. In actions which add new call legs mid-call, like **call-control fork** and **conference**, each new leg creates a new session between it and the originating leg. These new sessions inherit the original request ID.

The request ID is an obscure string as far as the ASC is concerned. Any interpretation of its contents is solely a matter for the application writer.

Session IDs

Each session in the ASC is given a session ID, internally represented as a 64-bit number, which functions as a globally unique ID (GUID). This means session IDs are not repeated even after the ASC reboots and are unique between multiple ASCs. The session ID is returned in response to all call creation, disconnection and manipulation actions, and in all events pertaining to the session.

Call Leg Handles

Each leg of a call is identified by a handle, internally represented as a 32-bit number. You must reference a call leg handle in all actions performed on calls after they have been created.

SIP Call-IDs

Within SIP, calls are identified by Call-IDs, which functions as a GUID. Every call leg has a unique call ID, and these are reported in the CallCreated, CallConnected, and CallTerminated events. The call-ID should be used when you need to correlate calls with other systems. If this is not sufficient, you can populate call events with custom parameters that can be obtained from arbitrary SIP headers.

Configuring To and From URIs

When you use the **call-control call** action, you need to include **to** and **from** properties. You can configure the ASC so that you don't have to include the SIP scheme and domain parts every time you place a call. By configuring a condition list and header normalization, then adding them to a policy rule, the ASC looks for the absence of a host portion in the To URI in a **call-control** action, and adds the necessary components to the To and From URIs.

The following example displays a configuration where the ASC applies the condition list to the **call-control** action. It creates four **header-normalization** rules which prepend **sip:** to the **call-control to** and **from** properties and append **@acmepacket.com** to these properties.

```
config rule check-for-host
     config condition-list
      set to-uri-condition host match ^$
      set action-condition call-control
     return
     config session-config
      config header-settings
       config header-normalization 1
        set destination To
        set value prepend sip:
       return
       config header-normalization 2
        set destination From
        set value prepend sip:
       return
       config header-normalization 3
        set destination To
        set value append @acmepacket.com
       return
       config header-normalization 4
        set destination From
        set value append @acmepacket.com
       return
      return
     return
    return
```

For more information about configuring condition lists and normalization, see the Oracle Communications OS-E Object and Properties Reference Guide.

Action Results

When the **call-control** action is executed, you receive an XML result containing information about whether the action was successful or not.

The following is an example of an XML result generated from a successful **call-control** action:

```
<ExtActionResponse>
  <resultCode>0</resultCode>
  <resultStr>Success</resultStr>
  <info>343196502737231705
```

```
14490500:14490499</info>
  <structure>
    <CallControlCallResult>
      <requestId>foo123</requestId>
      <sessionId>343196502737231705</sessionId>
      <inCallLegHandle>14490500</inCallLegHandle>
      <outCallLegHandle>14490499</outCallLegHandle>
    </CallControlCallResult>
  </structure>
</ExtActionResponse>
```

A <resultCode> of zero indicates the action was successful. Any other value indicates a failure, which is described by the <resultStr> object.

The <info> element provides supplementary information about the executed **call-control** action. In the case of a successful call the first line is the session ID. The second line consists of the two call-leg handles, separated by a colon.

Structured information equivalent to the content of the <info> element is also returned for some of the **call-control** actions, making the extraction of the required fields easier. If it was provided in the original request, the requestId is returned in the structured information.

NOTE: Not all **call-control** actions return structured data. This only happens when the <info> element contains useful information that needs parsing.

When using a RESTful API, you can request the result in a simplified XML format by adding **&_format=simplified** to the URL. The following is an example of a simplified XML result.

```
<object xsi:type="ExtActionResponseType">
  <resultCode>0</resultCode>
  <resultStr>Success</resultStr>
  <info>343196530540399894
14490520:14490519</info>
  <structure xsi:type="CallControlCallResultType">
    <request-id>foo123</request-id>
    <session-id>343196530540399894</session-id>
    <in-call-leg-handle>14490520</in-call-leg-handle>
    <out-call-leg-handle>14490519</out-call-leg-handle>
  </structure>
</object>
```

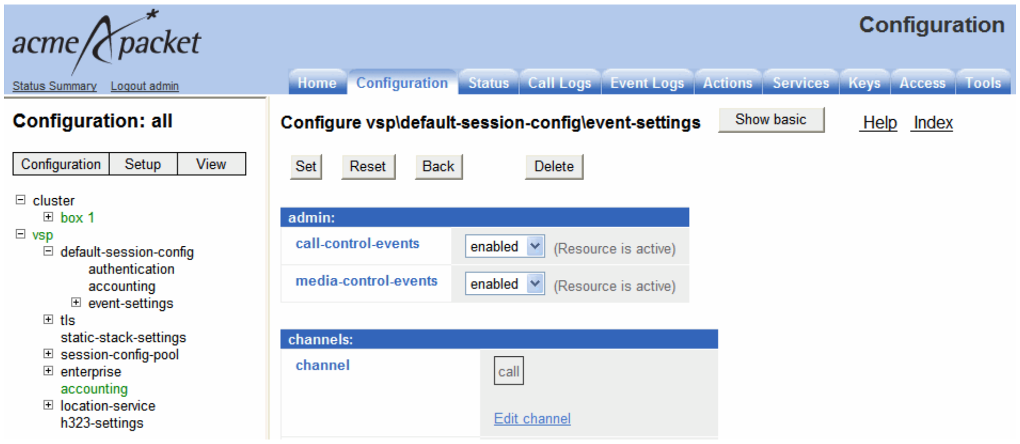
Configuring Call Events

When enabled to do so, the ASC can generate event messages, two of the most common types being call-control event messages and SIP event messages. To enable the ASC to generate call-control event messages, see the following section. To work with SIP event messages, see Chapter 2, Sending SIP Event Messages.

To generate call-control event messages:

1. Select the **Configuration** tab and click the **vsp > default- session-config** or **vsp > session-config-pool > entry** object.

2. Click the **event-settings** object.



Configure vsp\default-session-config\event-settings

admin:		
call-control-events	enabled	(Resource is active)
media-control-events	enabled	(Resource is active)

channels:	
channel	call

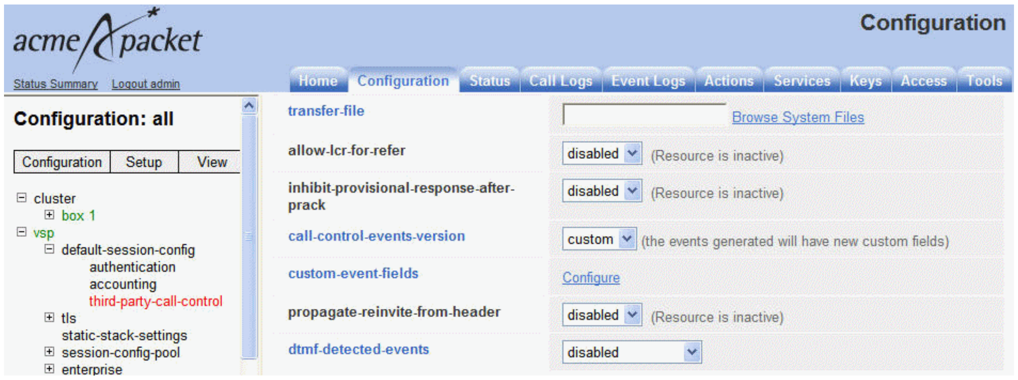
3. **call-control-events**—Set to **enabled** for the OS-E to send call-control events.
4. Click **Set.** Update and save the configuration.

The ASC includes certain standard information in the event messages it sends. However, if you want to include information not included in the standard format, you can configure the ASC to include custom content in the CallCreated, CallConnected, and CallTerminated event messages.

See Appendix B: Event Message Examples for examples of both legacy and new format and legacy and custom content event messages.

To include custom information in event messages:

1. Click the **Configuration** tab and select **third-party-call-control**.
2. Select **custom** from the **call-control-events-version** drop-down box. The default is **legacy**.
3. Click **Configure** next to **custom-event-fields** to set the custom event fields to include in the event messages.



transfer-file		
allow-lcr-for-refer	disabled	(Resource is inactive)
inhibit-provisional-response-after-prack	disabled	(Resource is inactive)
call-control-events-version	custom	(the events generated will have new custom fields)
custom-event-fields	Configure	
propagate-reinvite-from-header	disabled	(Resource is inactive)
dtmf-detected-events	disabled	

For more information on configuring named variables and regular expressions, see Using Regular Expressions in Chapter 1: How to Use the ACLI of the Oracle Communications OS-E Objects and Properties Reference Guide.

4. Click **Set**.
5. Update and save the configuration.

Common Call Events

The **call-control** actions create call events. The following table lists and describes common call events.

Event Name	Description	Parameters
CallCreated	Generated every time a call leg is created.	• [requestId] • handle • sessionID • callID • to • from • sessConfig (legacy schema only) • dtmfCapability (legacy schema only)
CallCreatedEventCustom	Generated every time a call leg is created and the ASC is configured to include custom event fields in event messages.	• [requestId] • handle • sessionID • callID • to • from • sessConfig (legacy schema only) • dtmfCapability (legacy schema only) • customField
CallConnected	Generated every time a call leg is connected.	• [requestId] • handle • sessionID • callID • to • from • content
CallConnectedEventCustom	Generated every time a call leg is connected and the ASC is configured to include custom event fields in event messages.	• [requestId • handle • sessionID • callID • to • from • content • customField
CallTerminated	Generated when a party hangs up and every time a call leg is terminated.	• [requestId] • handle • callDuration • reason • sessionID • callID
CallTerminatedEventCustom	Generated when a party hangs up and every time a call leg is terminated.	• [requestId] • handle • callDuration • reason • sessionID • callID • customField

Event Name	Description	Parameters
CallHeld	Generated every time a call leg is placed on hold.	• [requestID] • handle • heldByRemote—can be true or false
CallRetrieved	Generated every time a call leg is retrieved from being on hold.	• [requestID] • handle
PlayInitiated		• [requestID] • handle • scanTime
PlayComplete	Generated whenever an audio file has finished playing or when it has been stopped.	• [requestID] • handle • fileTime • playedTime
PlayPaused	Generated every time an audio message is paused.	• [requestID] • handle • fileTime • playedTime
PlayResumed	Generated every time you resume playing an audio message.	• [requestID] • handle • fileTime • playedTime
PlayStopped	Generated every time you stop playing an audio message.	• [requestID] • handle • fileTime • playedTime
PlayFailed		• [requestID] • handle • reason • scanTime
RecordComplete	Generated every time the recording of an audio message is finished.	• [requestID] • handle • fileName
FileInformation	Generated every time you request file information.	• [requestID] • fileTime
MessageSend	Generated every time you manually send a message.	• [requestID] • sessionID • responseCode—the SIP response code from the message recipient • responseString—the corresponding string • callID • to • from • ContentType • body

Event Name	Description	Parameters
MessageReceived	Generated every time SIP MESSAGE messages are received.	• [requestID] • sessionID • callID • to • from • contentType—normally has the value of text/plain • body—content of the message
IncomingDtmfDigitStart	Generated when the start of a DTMF digit is received on a call leg. Every digits receives its own event. You must set **session-config > in-dtmf-preferences** to detect DTMF methods of choice. For parked calls, you must set **nnos-call-policy > apply-policy-to-nnos-calls** to **enabled**.	• [requestID] • handle • method—identifies the method used to receive DTMF • digit • volume • duration—the initial duration in milliseconds; reflects how many milliseconds were are received in the first packet if received as an RFC 2833 event in the media stream
IncomingDtmfDigitUpdate	Generated when the end of a DTMF digit is detected on a call leg.	• [requestID] • handle • method • digit • volume • duration—reflects the duration of the entire DTMF tone.
OutgoingDtmfDigitStart	Generated when the start of a DTMF digit is sent on a call leg. You must set **session-config > out-dtmf-preferences** to detect DTMF methods of choice. The actual method used depends on the capabilities of the endpoint. Note that you cannot send DTMF digits to a parked endpoint.	• [requestID] • handle • method—identifies the method used to receive DTMF • digit • volume • duration—the initial duration in milliseconds; reflects how many milliseconds were are received in the first packet if received as an RFC 2833 event in the media stream
OutgoingDtmfDigitUpdate	Generated when the end of a DTMF digit is sent on a call leg.	• [requestID] • handle • method • digit • volume • duration—reflects the duration of the entire DTMF tone.

Event Name	Description	Parameters
CallRedirected	Generated when a party redirects a call leg.	• requestId • handle • sessionID • callID • to • from
AttachedEvent	Generated when a call leg is attached to a session.	• handle • sessionID • callID • to • from • requestID
DetachedEvent	Generated when a call leg is detached from a session.	• handle • sessionID • callID • to • from • requestID
MediaStartedEvent	Generated when a media event is started, such as playing a file.	• handle • sessionID • callID • to • from • requestID • capabilities • media-file-status
MediaCompleteEvent	Generated when a media event is complete.	• handle • sessionID • callID • to • from • requestID • media-file-status
MediaStoppedEvent	Generated when a media event is stopped.	• handle • sessionID • callID • to • from • requestID • media-file-status
MediaPausedEvent	Generated when media playback is paused.	• handle • sessionID • callID • to • from • requestID • media-file-status
MediaResumedEvent	Generated when a media playback is resumed.	• handle • sessionID • callID • to • from • requestID • media-file-status

Event Name	Description	Parameters
MediaSeekEvent	Generated when the location in a media source is changed.	• handle • sessionID • callID • to • from • requestID • media-file-status
RecordCompleteEvent	Generated when a recording event has completed.	• handle • sessionID • callID • to • from • requestID • filename
RecordingStartedEvent	Generated when on demand recording is started.	• handle • sessionID • callID • to • from • requestID • filename
RecordingStoppedEvent	Generated when on demand recording is stopped.	• handle • sessionID • callID • to • from • requestID • filename

The following is a list of elements commonly found in event messages:

- requestID—The ID provided by the call-control caller. This element only appears if it was originally provided.
- handle—The call leg handle, expressed as a decimal number.
- sessionID/session-id—The internally applied session ID, expressed as a decimal number.
- callID/call-id—The call-ID field from the SIP message. Each call leg should have distinct Call-IDs.
- to—The To URI.
- from—The From URI.
- sessConfig/session-config—The session configuration that was applied to the call.
- callDuration—The length of a call, expressed as an *ISO 8601-format* time duration. This may either look like P*n*DT*n*H*n*M*n*S (legacy format) or P*n*Y*n*M*n*DT*n*H*n*M*n*.*n*S (simplified format), where *n* represents the integer.
- reason—The reason a call was terminated, based on the SIP response message (200 for normal termination, 404 for not found, 500 for internal error, etc.)
- fileTime—The length of an audio file, in milliseconds.
- playTime—The number of milliseconds of an audio file that was played.
- fileName—The name of the file that was recorded.

Call-Control Actions

This section describes all of the **call-control** actions, their parameters, structure of their result XML, and events generated.

Parameters surrounded by brackets ([ ]) are optional.

call-control-accept

Accepts an incoming call from an offering endpoint.

> **Note:** You must specify content-type as application/sdp and body as the SDP for the call.

Syntax

```
call-control-accept <handle> [content-type] [body]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- [*content-type*]—Specifies the Content-Type: for the indication.
- [*body*]—Specifies the body for the indication.

call-control-annotate

Annotates the text you specify to a call leg.

Syntax

```
call-control-annotate <handle> <text>
```

Arguments

- <*handle*>—Identifies the handle of the call to which you want to add annotated information.
- <*text*>—The annotated text you are providing to the call leg.

call-control-attach

Attaches a call leg to an existing SIP session.

Syntax

```
call-control-attach <handle> <session-id>
```

Arguments

- <*handle*>—The handle of the endpoint to be attached.
- <*session-id*>—The session to which the endpoint is being attached.

call-control-call

Initiates a call using To and From SIP URIs you provide.

You can set the ASC to add post-dial digits to a **call-control call** action. Append the string **postd=*digits*** to the user portion of the **to** parameter. The following example shows the ASC adding post-dial digits **12345@acmepacket.com** to a call.

Syntax

```
call-control-call <to> <from> [requestId] [originatorFirst] [async] [
transport] [config]
```

Arguments

- <*to*>—The destination SIP URI of the call.

- <*from*>—The originating SIP URI of the call.
- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.
- [*originatorFirst*]—When **enabled** (the default), the originating party is connected first. When **disabled**, the called party is connected first.
- [*async*]—When **enabled**, causes the ASC to return a response immediately without waiting for the action to complete. When **disabled** (the default), the ASC waits for the action to complete before returning a response.
- [*transport*]—The transport method to use for the call. This can be set to **any**, **TCP**, **UDP**, or **TLS**.
- [*config*]—The **session-config** on the ASC to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

  Enclose the value in quotation marks when using the CLI.

call-control-call-to-session

Initiates a call to an existing session.

Syntax

```
call-control-call-to-session <to> <from> <session-id> [requestId]
[originatorFirst] [async] [transport] [config] [content-type] [body]
```

Arguments

- <*to*>—The destination SIP URI of the session.
- <*from*>—The originating SIP URI of the session.
- <*session-id*>—The optional session ID for the session.
- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.
- [*originatorFirst*]—When **enabled** (the default), the originating party is connected first. When **disabled**, the called party is connected first.
- [*async*]—When **enabled**, causes the OS-E to return a response immediately without waiting for the action to complete. When **disabled**, (the default) the OS-E waits for the action to complete before returning a response.
- [*transport*]—The transport method to use for the call. This can be set to **any**, **TCP**, **UDP**, or **TLS**.
- [*config*]—The **session-config** on the OS-E to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```
- [*content-type*]—Specifies the Content-Type: for the indication.
- [*body*]—Specifies the body for the indication.

call-control-create-session

Creates a rendezvous session to which you can then add call-legs, add named-variables, or destroy the session. The OS-E automatically assigns the session a unique 64-bit session ID.

Syntax

```
call-control-create-session [requestId] [to] [from]
```

Arguments

- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.
- [*to*]—The To URI for the rendezvous session.
- [*from*]—The From URI for the rendezvous session.

call-control-connect

Connects an existing parked call leg to a given endpoint. If the called party ends the call, the original call reverts back to a parked state.

Syntax

```
call-control-connect <handle> <endpoint> [async] [requestId] [park]
[config]
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*endpoint*>—The URI of the call's destination.
- [*async*]—When **enabled**, causes the OS-E to return a response immediately without waiting for the action to complete. When **disabled**, (the default) the OS-E waits for the action to complete before returning a response.
- [*requestId*]— A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.
- [*park*]—When enabled, the outgoing call leg persists and reverts to a parked state when its peer is terminated.
- [*config*]—The **session-config** on the OS-E to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

call-control-custom

Creates and controls established calls and overrides specific session configuration settings.

Syntax

```
call-control-custom <call> <to> <from> [requestId] [originatorFirst]
[async] [transport] [config] [session-id]
```

Arguments

- <*call*>—Initiates a call using provided To and From SIP URIs.
- <*to*>—The To URI for the session.
- <*from*>—The From URI from the session.
- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.

- [*originatorFirst*]—When **enabled** (the default), the originating party is connected first. When **disabled**, the called party is connected first.
- [*async*] —When **enabled**, causes the OS-E to return a response immediately without waiting for the action to complete. When **disabled**, (the default) the OS-E waits for the action to complete before returning a response.
- [*transport*]—The transport method to use for the call. This can be set to **any**, **TCP**, **UDP**, or **TLS**.
- [*config*]—The **session-config** on the OS-E to use to process a call. Use the full path to the **session-config**. For example:

```
vsp\session-config-pool\entry MyConfig
```

Enclose the value in quotation marks when using the CLI.

- [*session-id*] —The optional session ID for the session.

call-control-destroy-session

Destroys a rendezvous session.

Syntax

```
call-control-destroy-session <session-id>
```

Arguments

- <*session-id*>—The session-id for the rendezvous session you are destroying. This is the unique 64 bit session ID given to the session by the OS-E when it was created.

call-control-detach

Detaches a call leg from an existing SIP session. If you do not specify a session ID, the OS-E creates a new parked session with that call leg. If you specify a session ID, the OS-E parks the call leg to that existing session.

Syntax

```
call-control-detach <handle> [session-id]
```

Arguments

- <*handle*>—The handle of the endpoint to be detached.
- <*session-id*>—The optional session ID to which you are parking this call leg. If you do not provide a session ID, the ASC creates a new session.

call-control-detach-to-session

Detaches a call leg and parks it to an existing specified session.

Syntax

```
call-control-detach-to-session <handle> <session-id>
```

Arguments

- <*handle*>—The handle of the endpoint from which you are detaching.
- <*session-id*>—The session ID to which you are parking this call leg.

call-control-disconnect

Disconnects all legs of a call. The **handle** parameter can be the handle of either call leg.

Syntax

```
call-control-disconnect <handle>
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-drop-file

Plays the specified audio file to the party connected to the call leg. When finished, the ASC terminates the call leg.

Syntax

```
call-control-drop-file <handle> <filename> [async]
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*filename*>—The name of the audio file where a message is recorded or from where a message is played. Audio files must be .wav files in 44.1 kHz, 16-bit mono PCM format. If you give an invalid filename, it is placed in or taken from the /cxc directory.
- [*async*]—When **enabled**, causes the OS-E to return a response immediately without waiting for the action to complete. When **disabled** (the default), the OS-E waits for the action to complete before returning a response.

call-control-fork

Adds a new endpoint's SIP URI to the parked call. The endpoint can receive media but cannot send it. Multiple endpoints can be added using this action.

Syntax

```
call-control-fork <handle> <endpoint> [async] [requestId] [config]
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*endpoint*>—The URI of the call's destination.
- [*async*]—When **enabled**, causes the ASC to return a response immediately without waiting for the action to complete. When **disabled** (the default), the ASC waits for the action to complete before returning a response.
- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.
- [*config*]—The **session-config** on the ASC to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

  Enclose the value in quotation marks when using the CLI.

call-control-get-annotation

Retrieves the annotated text given to the call leg.

Syntax

```
call-control-get-annotation <handle>
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-hold

Places the specified call leg on hold. This puts the media of that call leg into send-only mode. The media of the other call leg, if present, is put into receive-only mode.

Syntax

```
call-control-hold <handle>
```

Parameters

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-info-request

Sends an INFO on an existing call.

Syntax

```
call-control-info-request <handle> [info-package] [content-type]
[body]
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- [*info-package*]—The INFO message to send to the existing call.
- [*content-type*]—Specifies the Content-Type: for the indication.
- [*body*]—Specifies the body for the indication.

call-control-insert-dtmf

Inserts DTMF digits into the call leg. DTMF is inserted only into the call leg specified; the other party does not hear it.

Note also that DTMF insertion is currently only supported for two-legged calls, not parked calls.

Syntax

```
call-control-insert-dtmf <handle> <digits> [volume] [duration]
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*digits*>—Specifies the digits inserted into the call leg.
- [*volume*]—The volume of the DTMF digits, in decimals from -36 to 0. The value **1** is the default.
- [*duration*]—The duration of each digit in milliseconds, from 100 to 10000. The value **0** is the default.

call-control-intercept

Connects an incoming call to an existing parked call.

Syntax

```
call-control-intercept <handle> <target>
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*target*>—The handle of the parked call.

call-control-join

Connects the parties of two separate calls together. The original call legs, identified by handle1 and handle2, are disconnected.

Syntax

```
call-control-join <handle1> <handle2>
```

Arguments

- <*handle1*>—Identifies the leg of the first call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*handle2*>—Identifies the leg of the second call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-media-pause

Pauses the playing of an audio file on an active call leg.

Syntax

```
call-control-media-pause <handle>
```

Parameters

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-media-resume

Resumes the playing of an audio file on an active call leg.

Syntax

```
call-control-media-resume <handle>
```

Parameters

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-media-scanner-start

Attaches a media scanner to a call-leg (in-leg or out-leg) and begins analyzing the signal strength of the received audio. The media scanner reports events when the signal strengths detected cross from the low-threshold property settings to the high-threshold property settings, or vice-versa, and based on the configuration for the media-scanner settings. The media-scanner configuration is retrieved from the session-config associated with the target call. You can specify a named session config, which overrides the session config. The media-scanner settings configuration applied is based on the following precedence:

- **in-media-scanner-settings**—media scanner settings per in-leg call
- **out-media-scanner-settings**—out media scanner settings per out-leg call
- **session-config-media-scanner-settings**—media scanner settings per session-config

- **default-media-scanner-settings**—default property settings

The media scanner will report one of the following events when a transition has occurred:

- **Short-pause**—When a transition (for example, from stable tone to quiet) takes less time than the low-long-duration property setting, such as less than 200 milliseconds
- **Long-pause**—When a transition (for example, from stable tone to quiet) takes more time than the low-long-duration property setting, such as more than 200 milliseconds
- **Short-talk**—When the media-scanner detects talk less than the high-long-duration property setting, such as less than 900 milliseconds
- **Long-talk**—When the media-scanner detects talk longer than the high-long-duration property setting, such as longer than 900 milliseconds
- **Stable-tone**—When the media-scanner detects a constant signal over a sample interval as determined by the averaging window.

Syntax

```
call-control-media-scanner-start <handle> [config]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- [*config*]—The **session-config** on the OS-E to use to process a call. Use the full path to the **session-config**. For example:

```
vsp\session-config-pool\entry MyConfig
```

call-control-media-scanner-stop

Detaches a media scanner from a call leg.

Syntax

```
call-control-media-scanner-stop <handle>
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-media-seek

Seeks a specific point in a monitored recording file.

Syntax

```
call-control-media-seek <handle> <seek-offset> [position]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*seek-offset*>—The offset, in milliseconds, to begin seeking. A negative value seeks backwards. Seeking starts at the spot specified by the **position** parameter.
- [*position*]—Indicates the position to begin seeking:
  - start—Seek from the start of the file. This is the default behavior.
  - current—Seek from the current position of the file.
  - end—Seek from the end of the file.

call-control-media-stop

Stops the playing of an audio file on an active call leg.

Syntax

```
call-control-media-stop <handle>
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-memo-begin

Records a message from the parked party, identified by a call leg handle, and stores it in a file you specify.

> **Note:** When **cluster** is **enabled**, **master-service > file-mirror** must be enabled for it to work properly.

Syntax

```
call-control-memo-begin <handle> <filename> [greeting] [cluster]
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*filename*>—The name of the audio file where a message is recorded or from where a message is played. Audio files must be .wav files in 44.1 kHz, 16-bit mono PCM format. If you give an invalid filename, it is placed in or taken from the /cxc directory.
- [*greeting*]—A greeting file that may be applied first as a prompt.
- [*cluster*]—When **enabled**, the file is available to all ASCs in the cluster. When **disabled** (the default), the file is only available on the local ASC.

call-control-memo-end

Ends a recording on the specified call leg.

Syntax

```
call-control-memo-end <handle>
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-message

Connects to a given endpoint, plays the file you specify, then disconnects the call. If you specify a From URI, that appears in the From header as the calling party; if no URI is specified, the To URI is used as the From header.

Syntax

```
call-control-message <filename> <endpoint> [from] [requestId] [async]
[config]
```

Arguments

- <*filename*>—The name of the audio file where a message is recorded or from where a message is played. Audio files must be .wav files in 44.1 kHz, 16-bit mono PCM format. If you give an invalid filename, it is placed in or taken from the /cxc directory.

- <*endpoint*>—The URI of the call's destination.
- [*from*]—The originating SIP URI of the call.
- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.
- [*async*]—When **enabled**, causes the ASC to return a response immediately without waiting for the action to complete. When **disabled** (the default), the ASC waits for the action to complete before returning a response.
- [*config*]—The **session-config** on the ASC to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

  Enclose the value in quotation marks when using the CLI.

call-control-message-request

Sends a MESSAGE on an existing call.

Syntax

```
call-control-message-request <handle> [content-type] [body]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- [*content-type*]—Specifies the Content-Type: for the indication.
- [*body*]—Specifies the body for the indication.

call-control-modify

Sends a re-INVITE on an existing call leg.

Syntax

```
call-control-modify <handle> [content-type] [body]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- [*content-type*]—Specifies the Content-Type: for the indication.
- [*body*]—Specifies the body for the indication.

call-control-monitor-file

Attaches a monitor session to a recording file. A recording file can be a live session currently being recorded, an old session that was recorded, an on-demand recording of a session, or a memo actively being recorded.

Syntax

```
call-control-monitor-file <handle> <session-id> <monitor-target>
[seek-offset] [position]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

- <*session-id*>—The optional session ID for the session.
- <*monitor-target*>—The filename of the file to be played. This can be:
    - session—A session recording file is going to be monitored.
    - memo—A memo actively being recorded is going to be monitored.
    - name—The on-demand filename specified in the call-control-record-start action is being monitored.
- [*seek-offset*]—The offset, in milliseconds, to begin seeking. A negative value seeks backwards. Seeking starts at the spot specified by the **position** parameter.
- [*position*]—Indicates the position to begin seeking

call-control-monitor-session

Attaches a monitor session to a live target session. The monitor session must join the target session in-progress as it has no ability to seek forward or backward during a live recording.

Syntax

```
call-control-monitor-session <handle> <session-id>
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*session-id*>—The optional session ID for the session.

call-control-mute-off

Turns off the mute functionality for a call leg.

Syntax

```
call-control-mute-off <handle>
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-mute-on

Turns on the mute functionality for a call leg.

Syntax

```
call-control-mute-on <handle>
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-notify

Causes a SIP NOTIFY message to be sent to the party you specify in the **handle** parameter, with the value of the Event header set by the **event** parameter.

Syntax

```
call-control-notify <handle> <event>
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*event*>—The content of the Event header.

call-control-notify-request

Sends a NOTIFY on an existing call.

Syntax

```
call-control-notify-request <handle> <event> [async] [content-type]
[body]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*event*>—The content of the event header.
- [*async*]—When **enabled**, causes the OS-E to return a response immediately without waiting for the action to complete. When **disabled**, (the default) the OS-E waits for the action to complete before returning a response.
- [*content-type*]—Specifies the Content-Type: for the indication.
- [*body*]—Specifies the body for the indication.

call-control-options-request

Sends an OPTIONS on an existing call.

Syntax

```
call-control-options-request <handle> [content-type] [body]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- [*content-type*]—Specifies the Content-Type: for the indication.
- [*body*]—Specifies the body for the indication.

call-control-park

Creates a call to an endpoint from a given SIP URI. If you specify a From URI, it is used as the From URI in the SIP message; if you specify no From URI, the From URI is that of the given endpoint.

Syntax

```
call-control-park <endpoint> [from] [requestId] [async] [sessionID]
[persist] [config]
```

Arguments

- <*endpoint*>—The URI of the call's destination.
- [*from*]—The originating SIP URI of the call.
- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.

- [*async*]—When **enabled**, causes the ASC to return a response immediately without waiting for the action to complete. When **disabled** (the default), the ASC waits for the action to complete before returning a response.
- [*sessionId*]—The optional session ID for a rendezvous session.
- [*persist*]—When **enabled**, a connected session remains parked even when the remote endpoint disconnects.
- [*config*]—The **session-config** on the ASC to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

  Enclose the value in quotation marks when using the CLI.

call-control-park-to-session

Parks a call to an existing session.

Syntax

```
call-control-park-to-session <endpoint> <sessionID> [from] [requestId]
[async] [persist] [config]
```

Arguments

- <*endpoint*>—The handle of call leg on the existing session.
- <*session-id*>—The optional session ID for the session.
- [*from*]—The From URL of the parked call.
- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.
- [*async*]—When **enabled**, causes the OS-E to return a response immediately without waiting for the action to complete. When **disabled**, (the default) the OS-E waits for the action to complete before returning a response.
- [*persist*]—When **enabled**, a connected session remains parked even when the remote endpoint disconnects.
- [*config*]—The **session-config** on the OS-E to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

call-control-persistence

Makes a call-leg persist in a parked state even when its peer is terminated.

Syntax

```
call-control-persistence <handle> <persist>
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*persist*>—When **enabled**, a connected session remains parked even when the remote endpoint disconnects.

call-control-play

Plays a given audio file to the specified call leg. If two call legs are connected, the file is played to both parties.

If the **session-config > media-scanner-settings** is configured, the ASC waits until the recipient (or an answering machine) has finished speaking before delivering the message. If the media scanner times out waiting for the recipient to finish speaking, the file is not played.

Syntax

```
call-control-play <handle> <filename> [startTime] [async]
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*filename*>—The name of the audio file where a message is recorded or from where a message is played. Audio files must be .wav files in 44.1 kHz, 16-bit mono PCM format. If you give an invalid filename, it is placed in or taken from the /cxc directory.
- [*startTime*]—The number of milliseconds the ASC waits before playing the file.
- [*async*]—When **enabled**, causes the ASC to return a response immediately without waiting for the action to complete. When **disabled** (the default), the ASC waits for the action to complete before returning a response.

call-control-record-start

Starts the on-demand recording or a target session to a specific <*filename*> file. This recording can then be monitored via the **call-control-monitor-file** action. You can execute this command one or more times for a given target session, provided you give it a different <*filename*> each time. If a <*filename*> already exists for a given target session, the existing <*filename*> is preserved and the action fails.

Syntax

```
call-control-record-start <session-id> <filename>
```

Arguments

- <*session-id*>—The optional session ID for the session.
- <*filename*>—The name of the recording for this particular target session.

call-control-record-stop

Stops the on-demand recording or a target session to a specific <*filename*>.

Syntax

```
call-control-record-stop <session-id> <filename>
```

Arguments

- <*session-id*>—The optional session ID for the session.
- <*filename*>—The name of the recording for this particular target session.

call-control-redirect

Redirects an initiated call to a new endpoint, prior to the call being answered. This creates a new call leg and cancels the original one.

Syntax

```
call-control-redirect <handle> <endpoint> [config]
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*endpoint*>—The URI of the call's destination.
- [*config*]—The **session-config** on the ASC to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

  Enclose the value in quotation marks when using the CLI.

call-control-reject

Rejects an incoming call from an offering endpoint.

Syntax

```
call-control-reject <handle> [response-code] [responseText]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- [*response-code*]—The SIP response code to return in response.
- [*responseText*]—Text text to return in the response.

call-control-retrieve

Retrieves the held call leg you specify by call handle. This reconnects the call's media for that call leg and, if present, the other call leg.

Syntax

```
call-control-retrieve <handle>
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-ringing

Redirects an initiated call to a new endpoint, prior to the call being answered. This creates a new call leg and cancels the original one.

Syntax

```
call-control-redirect <handle> <endpoint> [config]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*endpoint*>—The URI of the call's destination.
- [*config*]—The **session-config** on the OS-E to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

Enclose the value in quotation marks when using the CLI.

call-control-send-message

Sends a message to the endpoint specified by the To URI. If you specify a From URI, it is used for the From URI. If a From URI is not specified, the From URI is the same as the To URI.

Syntax

```
call-control-send-message <to> <from> [requestId] [content-type]
[body] [config]
```

Arguments

- <*to*>—The destination SIP URI of the call.
- <*from*>—The originating SIP URI of the call.
- [*requestId*]—A unique identifier provided by an external application. This value can be used to identify the call in subsequent events and actions. If a requestId is specified, there is a corresponding XML element in the event messages generated for the session.
- [*content-type*]—Should be set to **text/plain.**
- [*body*]—The content of the message.
- [*config*]—The **session-config** on the ASC to use to process a call. Use the full path to the **session-config**. For example:

  ```
  vsp\session-config-pool\entry MyConfig
  ```

  Enclose the value in quotation marks when using the CLI.

call-control-subscribe-request

Sends a SUBSCRIBE on an existing call.

Syntax

```
call-control-subscribe-request <handle> [pkg] [expires] [content-type]
[body]
```

Arguments

- <*handle*>—Identifies the leg of a session. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- [*pkg*]—Specifies the package for the SUBSCRIBE.
- [*expires*]—The expiration value for the SUBSCRIBE.
- [*content-type*]—Specifies the Content-Type: for the indication.
- [*body*]—Specifies the body for the indication.

call-control-terminate

Terminates the call leg indicated by the handle you specify. This parameter is only available for calls with a parked status.

Syntax

```
call-control-terminate <handle>
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.

call-control-transfer

Transfers the specified call leg to the specified To SIP URI. The original call leg, referred to by its handle, is disconnected. Handle can be thought of as belonging to the party doing the transfer, even though the transfer is done via a third-party action.

Syntax

```
call-control-transfer <handle> <to>
```

Arguments

- <*handle*>—Identifies the leg of a call. Handles are returned as part of the <info> element of **call-control** results and can be used to manipulate each leg of a call independently.
- <*to*>—The destination SIP URI of the call.

Media Forking

The OS-E now supports audio and video media forking, meaning a source endpoint can fork media to one or more target endpoints. The source endpoint is a one-legged call which initiates a call to the OS-E. The OS-E then initiates a call to each forked target. In this type of media forking, the media flows in one direction only, from the source endpoint, through the OS-E, to each of the targets.

Media forking is initiated via the **call-control-fork** action. This action establishes a call from the source endpoint and replicates the media to the newly established target sessions. The action syntax is:

```
call-control-fork <handle> <endpoint> [async] [requestID] [config]
```

Valid arguments for this action are:

- <*handle*>—The call-leg handle of the source endpoint.
- <*endpoint*>—The URL of the target endpoint.
- [*async*]—When enabled, this action returns immediately as opposed to waiting for the action to complete the call.
- [*requestID*]—This call's request identifier. If included, this value is returned in all of this action's events.
- [*config*]—The **session-config** to use when calling the endpoint.

To end a media forking session, use the **call-control disconnect** action. If you disconnect a target endpoint, the call from the source and remaining targets is still active. If you disconnect the source endpoint, all call-legs to the target endpoints are disconnected. The action syntax is:

```
call-control disconnect <handle>
```

Valid arguments for this action are:

- <*handle*>—The handle of the call-leg to disconnect.

Attended Voice Insertion

This feature allows a caller to play a pre-recorded message that both the caller and callee can hear. The caller can start playing the message at any point, pause, resume, or stop playing the message.

The OS-E allows the caller to begin playing a file with the option of seeking to a specified point via the **call-control play** action. The action syntax is:

```
call-control play <handle> <filename> [startTime] [async]
```

Valid arguments for this action are:

- <*handle*>—The call-leg handle on which the file is played.
- <*filename*>—The .wav file being played.
- [*startTime*]—The optional start time in milliseconds. This is used if the caller does not want to begin playing the file right at the beginning. The default value is **0**.
- [async]—When enabled, this action completes immediately as opposed to waiting for the action to complete the call.

The OS-E stops the playing of a file via the **call-control media-stop** action. The action syntax is:

```
call-control-media-stop <handle>
```

Valid arguments for this action are:

- <*handle*>—The call-leg handle where the file is stopped.

The OS-E pauses the playing of a file via the **call-control media-pause** action. The action syntax is:

```
call-control-media-pause <handle>
```

Valid arguments for this action are:

- <*handle*>—The call-leg handle where the file is paused.

The OS-E resumes the playing of a file via the **call-control media-resume** action. The action syntax is:

```
call-control-media-resume <handle>
```

Valid arguments for this action are:

- <*handle*>—The call-leg handle where the file is resumed.

You can configure the OS-E to send events regarding the status of the file being played by the **call-control play** action. For more information on call-control events, see Chapter 3 of this guide.

When configured, the OS-E sends the following events:

- PlayInitiated—The file has begun to play.
- PlayPaused—The file has been paused.
- PlayResumed—The file has resumed playing.
- PlayStopped—The file has stopped playing.
- PlayCompleted—The file has completed playing.

On-Demand Call Monitoring and Recording

The OS-E now supports on-demand call monitoring, meaning an endpoint, known as the monitor session, has the ability to attach itself to either a live target session or recording file, for the purpose of listening.

When monitoring a live target session, you have the ability to start and stop monitoring. Any time a monitor session starts listening, it joins the session in-progress.

You can configure one or more locations to which the OS-E writes files for on-demand recording files via the **services > data-locations > rtp-on-demand-recorded**

<*directory*> [*directory*] property. By default the OS-E writes on-demand recording files to the /cxc_common/rtp_on_demand_recorded directory.

Once you have the **rtp-on-demand-recorded** property configured, you can set a rotation scheme for writing on-demand recorded files to a directory using the **services > data-locations > rtp-on-demand-recorded-rotation** property. This property can be set to either **first-available** or **round-robin**. **First-available** means the OS-E writes to the first directory that has enough space to hold the recording listed under the **rtp-on-demand-recorded** property and continues to write to that directory until the disk is full and then moves onto the next directory on the list. **Round-robin** means the OS-E rotates through all configured directories in a round-robin manner. This allows for an increase in the volume of simultaneous on-demand recorded calls by spreading the load across multiple disks.

There are four types of monitoring you can perform when working with a recording file: a live target session currently being recorded, a previously recorded session, an on-demand recording session, and a memo actively being recorded. When monitoring a recording file, the monitor session does have the ability to pause, resume, and seek forward or backward to a particular point in the file.

The OS-E attaches a monitor session to a live target session via the **call-control-monitor-session** action. The monitor session must join the target session in-progress as it has no ability to seek forward or backward during a live recording.

> NOTE: The session-config > nnos-call-policy > apply-policy-to-nnos-calls property must be enabled for this feature to work.

The **call-control-monitor-file** action attaches a monitor session to a recording file. A recording file can be a live session currently being recorded, an old session that was recorded, an on-demand recording of a session, or a memo actively being recorded.

> NOTE: The **session-config > nnos-call-policy > apply-policy-to-nnos-calls** property must be **enabled** for this feature to work.

To stop monitoring a target session or a recording file, use the **call-control media-stop** file. The action syntax is:

```
call-control-media-stop <handle>
```

Valid arguments for this action are:

- <*handle*>—The monitor session handle to stop listening.

The **call-control media-pause** action pauses the monitor of a recording file. The action syntax is:

```
call-control-media-pause <handle>
```

Valid arguments for this action are:

- <*handle*>—The monitor session handle to pause listening.

To resume monitoring a stopped or paused recording file, use the **call-control media-resume** action. The monitoring resumes from the point at which the monitoring was stopped or paused. The action syntax is:

```
call-control-media-resume <handle>
```

Valid arguments for this action are:

- <*handle*>—The monitor session handle to resume listening.

To seek to a specific point in a monitored recording file, use the **call-control media-seek** action. This action can also be used to seek to a certain point of a file when the **call-control play** action is used to play a file.

The **call-control-record-start** action starts the on-demand recording of a target session to a specific file. This recording can then be monitored via the **call-control-monitor-file** action. You can execute this command one or more times for a given target session, provided you give it a different <*filename*> each time. If a <*filename*> already exists for a given target session, the existing <*filename*> is preserved and the action fails.

The **call-control-record-stop** action stops the on-demand recording of a target session to a specific <*filename*>.

The **media-on-demand-delete** command deletes on-demand recording files by specifying a session-id and filename. The action syntax is:

```
media-on-demand-delete <session-id> <filename>
```

Valid arguments for this action are:

- <*session-id*>—The session-id of the on-demand recording file to delete.
- <*filename*>—The on-demand recording filename to delete.

The **media-on-demand-delete-old** action deletes all on-demand recording files that are older than the specified time. The time units can be specified in days or seconds. The default value in which to purg3e old on-demand recording files is 7 days. The action syntax is:

```
media-on-demand-delete-old <age> [units]
```

Valid arguments for this action are:

- <*age*>—The age at which to delete on-demand recordings. The default is 7 days.
- [*units*]—This optional parameter allows you to specify the units in which the age is measured. This can be either **days** or **seconds**. If you do not specify, the default is days.

You can archive on-demand recordings using the existing archiving support when the **session-config > media > recording-policy** object is configured. This existing archiver has been extended to support the archiving of one or more on-demand recordings per session. Note that multiple on-demand recordings can be created for the same session. The archiver also supports mixing the ras media files to a .wav file and archiving that file.

The **on-demand-mixed-media** command can be configured under either the **vsp > accounting > archive-local > path <*name*>** object or **vsp > accounting > archive-external > url <*url*>** object. It has been created to control whether the on-demand recordings associated with a session are mixed to a .wav file and included in the archive for a call. It also determines whether the raw on-demand recordings are included in the archive if the mixing of the on-demand recording fails.

The **on-demand-mixed-media** syntax is:

```
on-demand-mixed-media <include> <include-raw-media-on-mix-fail>
```

Valid arguments for this property are:

- <*include*>—Can be set to **true** or **false** and determines whether on-demand mixed media is included in the archive.
- <include-raw-media-on-mix-fail>—Can be set to **true** or **false** and determines whether on-demand raw media is included in the archive if the mixing fails.

To always include raw media in the archive use the **include-on-demand-raw-media** property configured under either the **vsp > accounting > archive-local > path <*name*>** object or **vsp > accounting > archive-external > url <*url*>** object. This property can be set to either **true** or **false**.

The **mix-session-threaded** action has been extended to support the mixing of on-demand recorded files. A new ***<recorded-filename>*** argument has been added to this action to indicate the on-demand recording filename that is being mixed. For more information on the **mix-session-threaded** action see the Oracle Communications OS-E Objects and Properties Reference Guide.

> **Note:** When using the **mix-session** and **mix-session-threaded** actions after executing the **call-control-record-start** and **call-control-record-stop** commands in the context of on-demand recording, you must include the *<recorded-filename>* argument.

Two status show commands have been created to allow you to view on-demand call monitoring information.

The **show media-on-demand-recordings** status displays the on-demand recording files for a given session. This information displayed with this status provider can be used with the **call-control monitor-file** command to listen to these on-demand recording files.

```
NNOS-E>show media-on-demand-recordings

session-id          filename  start-time
----------          --------  ----------
0x4c42b6e0e5a6577    r9        15:57:30.798092 Tue 2011-12-06
0x4c42be1a934be68    r10       12:12:17.890681 Thu 2011-12-08
```

Field	Description
session-id	The session-id of the session that is recorded.
filename	The on-demand recording filename.
start-time	The date and time the on-demand recording was started.

The **show media-memo-recordings** status provider displays the sessions that are actively recording memos. The information displayed with this status provider can be used with the **call-control monitor-file** command to listen to these memos as they are being recorded.

```
NNOS-E>show media-memo-recordings

session-id          filename   start-time
----------          --------   ----------
0x4c43a6bb77329a3    frank.wav  15:00:04.295810 Mon 2012-02-06
```

Field	Description
session-id	The session-id of the session that is recording a memo.
filename	The filename of the memo recording
start-time	The date and time the memo recording file was started.

Rendezvous Session Support

The OS-E now supports rendezvous sessions. Rendezvous sessions are useful for accumulating information in named variables before attaching call legs. They have unique 64 bit session IDs as with other OS-E sessions but do not have any call-legs attached. Once a rendezvous session is created, you can add call-legs, remove call-legs, destroy the session, or add named-variables.

Using the **call-control-create-session** action, you can create a rendezvous session to which you can then add call-legs, add named-variables, or destroy the session. The OS-E automatically assigns the session a unique 64 bit session ID. The action syntax is:

```
call-control-create-session
```

To destroy a rendezvous session manually, use the **call-control-destroy-session** action. .

The OS-E also destroys a rendezvous session if you have the **session-config > sip-settings > session-duration-max** property set. This property specifies how many seconds the OS-E maintains a session after the session has been successfully established. It puts a timer on the session and forces it to close upon expiration. If set to **0** (the default), the session remains open until it is complete and does not timeout. This property applies to all sessions on the OS-E, including rendezvous sessions.

To add call-legs to a rendezvous session dynamically, use the **call-control park** and **call-control call** actions. These have been enhanced to include an optional [***session-id]*** argument. Once a rendezvous session has a call-leg attached, it is "promoted" to a connected session. All subsequent interactions can be accomplished using the call control handles as you would with a normal session.

Manually Attaching and Detaching From an Endpoint

The OS-E supports functionality which provides control over managing session endpoints. The **call-control-attach** and **call-control-detach** actions allow you to attach and detach from rendezvous sessions and endpoints manually.

Rendezvous sessions can be created by one of two ways. Via the **call-control-create-session** action or by detaching an endpoint from a single endpoint session. For more information on rendezvous sessions, see the Rendezvous Session Support section in this guide.

Endpoints can be created in a few different ways. You can create an outbound call-leg via the **call-control park** action, enable the **third-party-call-control > park-incoming-calls** property, or use the **call-control-detach** command during a rendezvous session.

You can manually attach an endpoints to either rendezvous sessions or sessions resulting from a SIP DIALOG. When attached to a rendezvous session, an endpoint remains in a PARKED state. When attached to a single endpoint session, the OS-E joins the two endpoints and two-way communication can take place. When the call is terminated, a previously PARKED endpoint reverts back to PARKED and the session remains active.

When you attach an endpoint to a session already containing two endpoints, a three-way conference call is created and three-way communication can take place. When one endpoint terminates the call, the remaining two endpoints remain joined and two-way communication commences.

To attach an endpoint to an existing session, use the **call-control-attach** action. The action syntax is:

```
call-control-attach <handle> <session-id>
```

Valid arguments for this action are:

- <*handle*>—The handle of the endpoint to be attached.

- <*session-id*>—The session to which the endpoint is being attached.

Just as you can manually attach endpoints, you can also manually detach endpoints. If you detach a PARKED endpoint from a session that is not a rendezvous session, the endpoint is terminated. If you detach a CONNECTED endpoint, both endpoints from the two-way session are placed in a PARKED state. If you detach a CONFERENCED endpoint, the detached endpoint is placed in a PARKED state and the remaining two endpoints continue as a two-way call.

To detach an endpoint from a session, use the **call-control-detach** action.

Appendix A ASC API Examples

This appendix provides examples for the ASC top-level APIs. Included are both SOAP and REST web services requests and responses. REST actions are broken down to include both flat and hierarchical request examples.

ASC top-level APIs are:

- getConfig
- setConfig
- doAction
- getStatus
- queryStatus

getConfig

The ASC getConfig API uses the HTTP GET Method.

The following examples display a getConfig API request from the server for the **cluster** object. The responses received from the client include the **cluster** configuration, including all of its subobject configurations.

SOAP

Request

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:mgmt="http://www.acmepacket.com/asc/ws/mgmt">
   <soapenv:Header/>
   <soapenv:Body>
      <mgmt:getConfig>
         <!--Zero or more repetitions:-->
         <config homogeneous="false">
            <!--Zero or more repetitions:-->

         <object xsi:type="MasterServicesType"
xmlns="http://www.acmepacket.com/asc/ws/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <!--Optional:-->

        </object></config>
         <!--Zero or more repetitions:-->
      </mgmt:getConfig>
   </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<cov:getConfigResponse
xmlns:cov="http://www.acmepacket.com/asc/ws/mgmt">
  <config>
```

```
    <object xsi:type="data:MasterServicesType" revision="1"
xmlns:data="http://www.acmepacket.com/asc/ws/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <cluster-master>
        <admin>enabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
      </cluster-master>
      <directory>
        <admin>disabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
      </directory>
      <accounting>
        <admin>disabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
      </accounting>
      <database>
        <admin>enabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
        <maintenance>
          <time-of-day>
            <time>2012-01-30T03:00:00.000-05:00</time>
          </time-of-day>
        </maintenance>
        <database-threads-max>4</database-threads-max>
        <sip-cache-size>30000</sip-cache-size>
        <performance>call-details</performance>
        <dos-tcp-connect-multiplier>5</dos-tcp-connect-multiplier>
        <dos-tls-connect-multiplier>10</dos-tls-connect-multiplier>
        <sip-registers>enabled</sip-registers>
        <max-queue-depth>4000</max-queue-depth>
        <caching-threshold>3500</caching-threshold>
        <media>enabled</media>
        <write-mode>copy</write-mode>
      </database>
      <registration>
        <admin>enabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
        <mirror-all-entries>enabled</mirror-all-entries>
        <mirror-location-cache>enabled</mirror-location-cache>
        <force-regdb-lookup>disabled</force-regdb-lookup>
        <cache-poll-interval>86400</cache-poll-interval>
        <max-poll-duration>1000</max-poll-duration>
        <max-entries-per-poll>100</max-entries-per-poll>
      </registration>
      <route-server>
        <admin>enabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
        <max-routes>automatic</max-routes>
        <client-request-sender>only-master</client-request-sender>
        <simple-updates>enabled</simple-updates>
      </route-server>
      <sampling>
        <admin>enabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
        <SamplingTarget xsi:type="data:SamplingDatabaseType">
          <admin>enabled</admin>
          <duration>7</duration>
          <status>
            <cpu-usage>
              <admin>enabled</admin>
              <interval>P0Y0M0DT0H5M0.000S</interval>
            </cpu-usage>
          </status>
        </SamplingTarget>
      </sampling>
      <jtapi>
        <admin>disabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
      </jtapi>
      <advertisement-interval>60</advertisement-interval>
      <boot-interval>30</boot-interval>
    </object>
  </config>
</cov:getConfigResponse>
```

REST

Request

```
http://172.30.80.24:8080/cms/config?name=MasterServices
```

Response

```
<?xml version="1.0"?>
<object xsi:type="ExtPageListType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<version>E3.6.0.M5P0</version>
```

```
<resultCode>0</resultCode>
<resultStr>Success</resultStr>
<objects revision="1" xsi:type="MasterServicesType">
      <cluster-master>
        <admin>enabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
      </cluster-master>
      <directory>
        <admin>disabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
      </directory>
      <accounting>
        <admin>disabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
      </accounting>
      <database>
        <admin>enabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
        <maintenance>
          <time-of-day>
            <time>2012-01-30T03:00:00.000-05:00</time>
          </time-of-day>
        </maintenance>
        <database-threads-max>4</database-threads-max>
        <sip-cache-size>30000</sip-cache-size>
        <performance>call-details</performance>
        <dos-tcp-connect-multiplier>5</dos-tcp-connect-multiplier>
        <dos-tls-connect-multiplier>10</dos-tls-connect-multiplier>
        <sip-registers>enabled</sip-registers>
        <max-queue-depth>4000</max-queue-depth>
        <caching-threshold>3500</caching-threshold>
        <media>enabled</media>
        <write-mode>copy</write-mode>
      </database>
      <registration>
        <admin>enabled</admin>
        <host-box name="cluster\box 1"/>
        <group>0</group>
        <preempt>false</preempt>
        <takeover-timer-value>1000</takeover-timer-value>
        <mirror-all-entries>enabled</mirror-all-entries>
        <mirror-location-cache>enabled</mirror-location-cache>
```

```
          <force-regdb-lookup>disabled</force-regdb-lookup>
          <cache-poll-interval>86400</cache-poll-interval>
          <max-poll-duration>1000</max-poll-duration>
          <max-entries-per-poll>100</max-entries-per-poll>
        </registration>
        <route-server>
          <admin>enabled</admin>
          <host-box name="cluster\box 1"/>
          <group>0</group>
          <preempt>false</preempt>
          <takeover-timer-value>1000</takeover-timer-value>
          <max-routes>automatic</max-routes>
          <client-request-sender>only-master</client-request-sender>
          <simple-updates>enabled</simple-updates>
        </route-server>
        <sampling>
          <admin>enabled</admin>
          <host-box name="cluster\box 1"/>
          <group>0</group>
          <preempt>false</preempt>
          <takeover-timer-value>1000</takeover-timer-value>
          <SamplingTarget xsi:type="data:SamplingDatabaseType">
            <admin>enabled</admin>
            <duration>7</duration>
            <status>
              <cpu-usage>
                <admin>enabled</admin>
                <interval>P0Y0M0DT0H5M0.000S</interval>
              </cpu-usage>
            </status>
          </SamplingTarget>
        </sampling>
        <jtapi>
          <admin>disabled</admin>
          <host-box name="cluster\box 1"/>
          <group>0</group>
          <preempt>false</preempt>
          <takeover-timer-value>1000</takeover-timer-value>
        </jtapi>
        <advertisement-interval>60</advertisement-interval>
        <boot-interval>30</boot-interval>
  </objects>
  </object>
```

setConfig

This API uses the HTTP POST Method.

The following examples display a setConfig API request from the server, configuring a CLI banner via the **cli** object's **banner** property. The responses received from the client indicate the action was successful.

SOAP

Request

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:cal="http://www.covergence.com/ws/callouts">
   <soapenv:Header/>
   <soapenv:Body>
      <cal:setConfig mode="merge">
         <config>
            <Cluster>
               <box>
                  <Box number="1">
                     <cli>
                        <CLI>
                         <banner>The Acme Packet Application Session
Controller sure has Web Service

interfaces!</banner>
                        </CLI>
                     </cli>
                  </Box>
               </box>
            </Cluster>
         </config>
      </cal:setConfig>
   </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
     <setConfigResponse xmlns="http://www.covergence.com/ws/callouts">
         <Code>success</Code>
         <Text>Success</Text>
      </setConfigResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

REST

Request

```
POST
http://172.44.10.59:8080/cms/config?operation=modify&output=xml&mode=
merge&_format=legacy HTTP/1.1
```

```
Accept-Encoding: gzip,deflate
Content-Type: application/xml
User-Agent: Jakarta Commons-HttpClient/3.1
Host: 172.44.10.59:8080
Content-Length: 34

<SCP><admin>disabled</admin></SCP>

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID_WS=3C747DF0159B1E36714096B99FE2A7EA; Path=/;
HttpOnly
Cache-Control: no-cache
Content-Type: text/xml
Transfer-Encoding: chunked
Date: Thu, 13 Oct 2011 16:50:35 GMT
```

Response

```
<ExtActionResponse>
   <resultCode>0</resultCode>
   <resultStr>Success</resultStr>
</ExtActionResponse>
```

doAction

This API uses the HTTP GET Method.

Included are two examples for each SOAP and REST, the first example includes an unstructured response and the second example is a structured example. These examples display an API request from the server, performing the PING action. The ASC is pinging host 169.55.3.5. The responses received from the client indicate the action is a success.

SOAP

Request

Unstructured:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:cal="http://www.covergence.com/ws/callouts">
   <soapenv:Header/>
   <soapenv:Body>
      <cal:doAction>
         <action>
            <PingAction>
               <host>169.55.3.5</host>
            </PingAction>
         </action>
      </cal:doAction>
   </soapenv:Body>
</soapenv:Envelope>
```

Unstructured Response

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
```

```
   <soapenv:Body>
      <doActionResponse xmlns="http://www.covergence.com/ws/callouts">
         <Code>success</Code>
         <Text>Success</Text>
       <message>3 packets sent, 3 packets received, 0 packets lost (0%)
roundtrip minimum/average/maximum: 0.588/0.825/1.291 ms</message>
      </doActionResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Structured Response

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body>
      <cov:doActionExResponse xsi:type="data:ActionResultsType"
xmlns:cov="http://www.acmepacket.com/asc/ws/mgmt"
xmlns:data="http://www.acmepacket.com/asc/ws/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
         <result-code>0</result-code>
         <message>Success!</message>
         <info>28 bytes from 169.55.3.5: 0.134 milliseconds
28 bytes from 169.55.3.5: 0.107 milliseconds
28 bytes from 169.55.3.5: 0.102 milliseconds
3 packets sent, 3 packets received, 0 packets lost (0%)
Round trip minimum/average/maximum: 0.102/0.114/0.134
milliseconds</info>
         <structure xsi:type="data:ActionResultsPingType">
            <requests-sent>3</requests-sent>
            <replies-lost>0</replies-lost>
            <replies-received>3</replies-received>
            <round-trip-minimum>102</round-trip-minimum>
            <round-trip-average>114</round-trip-average>
            <round-trip-maximum>134</round-trip-maximum>
         </structure>
      </cov:doActionExResponse>
   </env:Body>
</env:Envelope>
```

REST

Flat Request

```
GET http://175.66.15.95:8080/cms?action=PingAction&Host=169.55.3.5
HTTP/1.1
```

Hierarchical Request

```
GET http://175.66.15.95:8080/cms/action/ping?host=169.55.3.5 HTTP/1.1
```

Unstructured Response

```
<ExtActionResponse>
   <Code>Success</Code>
   <Text>Success</Text>
   <message>3 packets sent, 3 packets received, 0 packets lost (0%)
roundtrip minimum/average/maximum: 0.588/0.825/1.291 ms</message>
</ExtActionResponse>
```

Structured Response

```
<?xml version="1.0"?>
```

```
<ExtActionResponse>
<resultCode>0</resultCode>
<resultStr>Success</resultStr>
<info>28 bytes from 169.55.3.5: 0.103 milliseconds 28 bytes from
169.55.3.5: 0.111 milliseconds 28 bytes from 169.55.3.5: 0.102
milliseconds 3 packets sent, 3 packets received, 0 packets lost (0%)
Round trip minimum/average/maximum: 0.102/0.105/0.111
milliseconds</info>
<structure>
<ActionResultsPing>
<RequestsSent>3</RequestsSent>
<RepliesLost>0</RepliesLost>
<RepliesReceived>3</RepliesReceived>
<RoundTripMinimum>102</RoundTripMinimum>
<RoundTripAverage>105</RoundTripAverage>
<RoundTripMaximum>111</RoundTripMaximum>
</ActionResultsPing>
</structure>
</ExtActionResponse>
```

getStatus

This ASC API uses the HTTP GET Method.

The following examples display a getStatus API request sent from the server, requesting the status of all current processes. The responses received from the client indicate the action was successful.

SOAP

Request

```
POST http://172.44.10.59:8080/ws HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: "getStatus"
User-Agent: Jakarta Commons-HttpClient/3.1
Host: 172.44.10.59:8080
Content-Length: 324

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:cal="http://www.covergence.com/ws/callouts">
   <soapenv:Header/>
   <soapenv:Body>
      <cal:getStatus>
         <status>
            <ClusterStatus />
         </status>
      </cal:getStatus>
   </soapenv:Body>
</soapenv:Envelope>

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
```

```
Set-Cookie: JSESSIONID_WS=35AC602AAB7ADC597C9BAFD27653FB5F; Path=/;
HttpOnly
Content-Type: text/xml
Transfer-Encoding: chunked
Date: Thu, 13 Oct 2011 17:47:12 GMT
```

Response

```
<?xml version='1.0' encoding='UTF-8'?><env:Envelope

xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"><env:Body><cov:
getStatusResponse
xmlns:cov="http://www.covergence.com/ws/callouts"><status><ClusterSta
tus
IPAddress="0.0.0.0"><boxID>1</boxID><bGetsConfig>false</bGetsConfig><
bGotConfig>false</bGotConfig></ClusterStatus></status></cov:getStatus
Response></env:Body></env:Envelope>
```

REST

Flat Request

```
GET http://175.66.15.95:8080/cms?status=ProcessStatus HTTP/1.1
```

Hierarchical Request

```
GET http://175.66.15.95:8080/cms/status/processes HTTP/1.1
```

Response

```
<ExtPageList><version>E3.6.0.M5P0</version><resultCode>0</resultCode>
<resultStr>Success</resultStr><objects><ClusterStatusIPAddress="0.0.0
.0"><boxID>1</boxID><bGetsConfig>false</bGetsConfig><bGotConfig>false
</bGotConfig></ClusterStatus></objects><totalPages>1</totalPages><cur
rentPage>1</currentPage><pageSize>1</pageSize></ExtPageList>
```

queryStatus

This ASC API uses the HTTP GET Method.

The following examples display a queryStatus API request sent from the server for the status of all running processes. The responses from the client server indicate the action was successful.

SOAP

Request

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:mgmt="http://www.acmepacket.com/asc/ws/mgmt">
   <soapenv:Header/>
   <soapenv:Body>
      <mgmt:queryStatus>
         <!--1 or more repetitions:-->
         <status homogeneous="false">
            <!--Zero or more repetitions:-->

         <object xsi:type="ns574:ProcessStatusType"
xmlns:ns574="http://www.acmepacket.com/asc/ws/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" process="ws">
   <!--Optional:-->
   <!--Optional:-->

   <!--Optional:-->
</object></status>
      </mgmt:queryStatus>
   </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<cov:queryStatusResponse
xmlns:cov="http://www.acmepacket.com/asc/ws/mgmt">
  <status>
    <object xsi:type="data:ProcessStatusType" process="WS"
xmlns:data="http://www.acmepacket.com/asc/ws/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <id>14817</id>
      <condition>running</condition>
      <run-level>7</run-level>
      <state>sleeping</state>
      <starts>1</starts>
      <uptime>P0Y0M4DT18H5M32.000S</uptime>
     <fds>198</fds>
    </object>
  </status>
</cov:queryStatusResponse>
```

REST

Flat Request

```
http://172.30.80.24:8080/cms?status=ProcessStatus&_format=simplified&
search.process=WS
```

Hierarchical Request

```
http://172.30.80.24:8080/cms/status/processes?search.process=WS&_form
at=simplified
```

Response

```
<?xml version="1.0"?>
<object xsi:type="ExtPageListType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<version>E3.6.0.M5P0</version>
<resultCode>0</resultCode>
<resultStr>Success</resultStr>
<objects xsi:type="ProcessStatusType" process="WS">
   <id>14817</id>
  <condition>running</condition>
  <run-level>7</run-level>
  <state>sleeping</state>
  <starts>1</starts>
  <uptime>P0Y0M4DT18H9M7.008S</uptime>
  <fds>195</fds>
  </objects>
<totalPages>1</totalPages>
<current-page>1</current-page>
<page-size>1</page-size>
</object>
```

Appendix B Event Message Examples

This appendix provides examples of the different types of event messages that can be sent by the ASC. The following examples are given:

- New Schema / Legacy Content
- New Schema / Custom Content

For more information on the different types of event message formatting and content, see the Legacy and New Schemas section of Chapter 1.

New Schema / Legacy Content

The following example shows a CallConnected event message sent from an ASC that is using the new schema and is configured to include the legacy content.

With the new simplified format, some of the names of the event attributes are hyphenated, rather than using “camelCase”. The SOAP message use a different namespace, and the event name is an attribute of the <object> element.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <cov:processEvent
xmlns:cov="http://www.acmepacket.com/asc/ws/mgmt">
      <cov:event>
        <object xmlns:data="http://www.acmepacket.com/asc/ws/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="data:CallConnectedType">
          <handle>14287669</handle>
          <session-id>343194204702856025</session-id>
          <call-id>ZDk2MTQwOGFkNTY0ZmMyMTViYmUyNGJmN2EzNmVkNTY.</call-
id>
          <to>sip:1001@acmepacket.com</to>
          <from>sip:2001@acmepacket.com</from>
          <content>v=0
o=3cxVCE 49342965 311118690 IN IP4 192.168.220.1
s=3cxVCE Audio Call
c=IN IP4 192.168.220.1
t=0 0
m=audio 40030 RTP/AVP 0 8 101
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

          </content>
        </object>
      </cov:event>
    </cov:processEvent>
  </env:Body>
</env:Envelope>
```

New Schema / Custom Content

The following example shows a CallConnectedEventCustom event message sent from an ASC that is using the new schema and is configured to include custom content.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <cov:processEvent
xmlns:cov="http://www.acmepacket.com/asc/ws/mgmt">
      <cov:event>
        <object xmlns:data="http://www.acmepacket.com/asc/ws/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="data:CallConnectedEventCustomType">
          <handle>14287667</handle>
          <session-id>343194196653337756</session-id>
          <call-id>CXC-103-4b6001b8-8d14010a-13c4-4eaeffe4-c6764eb-
53ce4bcc</call-id>
          <cookie>3389006614</cookie>
          <to>sip:2001@acmepacket.com</to>
          <from>sip:1001@acmepacket.com</from>
          <customField>user-agent=X-Lite 4 release 4.1 stamp
63214;</customField>
          <content>v=0
o=- 12964565220154654 1 IN IP4 192.168.220.1
s=CounterPath X-Lite 4.1
c=IN IP4 192.168.220.1
t=0 0
m=audio 51518 RTP/AVP 107 0 8 101
a=rtpmap:107 BV32/16000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv

          </content>
        </object>
      </cov:event>
    </cov:processEvent>
  </env:Body>
</env:Envelope>
```

Appendix C ASC Web Services Samples

Introduction

This chapter provides a list and brief description of sample applications created using the Net-Net ASC. Access these samples by clicking on the **Samples** link of the ASC homepage.

Note: As of release 3.7.0M4, the ASC’s samples are no longer included in the installation file or supported.

Acme Voice Message Manager Web Application

Broadcasts or drops voice messages to your contacts.

Platform JavaScript

ASC Status Checker

Polls a configured ASC for utilization statistics and allows the user to issue ICMP and SIP pings from the ASC using REST.

Platform Google Android (Java)

ASCServer Sample WAR

Using the ASC SDK JAR, this sample implements a basic service handling processEvent requests, getAuthSessionPolicy, and hosts a JSP capable of executing Call Control requests. This sample is an ant-based project that can be deployed without modification to JBoss, GlassFish, or Tomcat.

Platform Java

ASC Voice Message Manager

Records, broadcasts, or drops a message to selected contacts using REST.

Platform Google Android (Java)

Banking Call Center

Simulates a call center application that manages call center agents. This web application works with the Banking Store application in the Adobe Flash section. When a user using the Banking Store application requires service, this application connects the user to the appropriate agent.

Platform IBM Websphere Community Edition (Java)

Banking Store

Provisions the user the ability to get connected with an appropriate call center agent who can service his banking service needs. This application works with the sample Banking Call Center web application in the Websphere Application Server Community Edition section and is an Adobe AIR application that runs on iOS devices.

Platform Adobe Flash

Call Monitor

Provisions the user the ability to silently monitor active calls, conference into active calls, record on-demand, and review active calls. Also provisions the user the ability to review previously recorded calls, whether recorded through session-policy or on-demand.

This sample is an Adobe AIR application that runs on iOS, Android, and Blackberry devices.

Platform Adobe Flash

Call Monitoring and Conferencing Web Application

Either conferences into a call or monitors calls and recordings.

Platform JavaScript

Call Transfer Web Application

Either transfers or conferences calls during the duration or a session.

Platform JavaScript

Click-to-Call Internet Explorer Toolbar

Enables call control on phone numbers in any web page displayed in Microsoft Internet Explorer.

Platform JavaScript with BestToolbars Toolbar Studio

Configuration Wizard Sample

Executes a configuration template using ASC web services and a Java Swing user interface.

Platform NetBeans (Java)

Download Servlet

Exposes a directory hierarchy for downloading. By default, this sample exposes the /cxc_common/export directory where recordings may have been stored.

Platform Java

Emergency Call Example

Records and sends messages to groups via a GWT/GXT web application. For each person in a group, multiple phone numbers are tried in succession.

Platform Google Web Tookit/GXT (Java)

Event Processor EJB

Demonstrates the use of EJB 3.1 to implement an event processor. The event processor uses the new dynamic event service registration to hook itself into the ASC event system and uses simple *plugin* jars to process the received events from the ASC. A sample *plugin* jar is included in the project. The project builds an Enterprise Application Resource (EAR) that can be deployed on a J2EE container that implements EJB 3.1. Servers that support this sample are Glassfish (3.x) and JBOSS (6 and 7).

Platform Java

KPML Plugin

An Ant-based project that uses the ASC SDK JAR to implement a basic sample service to handle processEvent requests and getAuthSessionPolicy and to host a JSP capable of executing Call Control requests.

This sample can be deployed without modification to JBoss, GlassFish, or Tomcat.

Platform Java

Media Scanner Web Application

Demonstrates ASC on-demand media scanning on a call leg to detect voice activity. This sample plays a media file when it detects voice inactivity on a call leg.

Platform	Java

Mobile Dialer

An embedded Adobe Flex-based dialer that allows inbound and outbound calls from the web application, as well as the ability to redirect calls from an external number to the Mobile Dialer app. This sample also features conference and transfer capabilities.

Platform	Adobe Flash

Outlook Click to Call Toolbar

This is an Outlook add-on that recognizes phone numbers from senders. You can then press a button to execute a click-to-call action to that selected phone number.

Platform	Microsoft .NET (Outlook 2007 or Outlook 2010)

Pet Store Post Dial

Demonstrates a retail web site communicating with an IVR. On a specific page, a customer can click a button to initiate a call with customer support. The web application appends post-dial digits dialed into the IVR after the call is established. The customer is immediately directed to the appropriate representative skipping several levels of IVR menus.

Platform	IBM WebSphere Application Server Community Edition (Java)

PIN Reset

The ASC plays pre-recorded files as prompts and sends events for DTMF tones to an application. This sample walks you through a script for resetting a PIN number for an account.

Platform	NetBeans (Java)

Pre-Call Authorization

Demonstrates the ASC’s pre-call authorization by requiring a pin to use the click to call action. Enter the from and to phone numbers in the form. The application calls the “from” number and asks for a PIN before proceeding to make the call.

Platform	Java

RCSE

Externally manages video streaming, forking, and chat roulette via a java-based set of web applications.

Platform Java

Request Proxy/Event Demultiplexing Client

Demonstrates sending web service requests to the WSDemux service while handling synchronous request responses and asynchronous events.

Platform Microsoft .NET

Request Proxy/Event Demultiplexing Service

Demonstrates the ASC handling web service requests and supporting the demultiplexing of processEvent notifications to the appropriate client endpoint.

Platform Microsoft .NET

RTP Status Example

A GWT/GXT web application that creates a simple RTP status monitoring tool. It uses a GWT CometD library to do cross-domain script-tag injection for gathering statistics.

Platform Google Web Tookit/GXT (Java)

Session Policy Web Application

Implements the ASC’s session-config policy that allows you to dynamically modify the session. Through configuration, this sample allows you to include a named variable that was retrieved from an LDAP server using a component of a SIP URI.. These named variables can later be used to do things such as modify the headers in the SIP message and modify routing.

Platform Java

Simple Flex Based Embedded Dialer

Demonstrates an embedded flex-based dialer that allows the user to place an audio call to any phone number via the ASC.

Platform Adobe Flash

Simple Flex Based Embedded Phone

Demonstrates an embedded flex-based phone that can call a pre-defined customer support number via the ASC. Supports bi-directional audio and incoming video directly in the browser.

Platform Adobe Flash

Voice Memo Manager

Allows you to record messages on the ASC and send them either in the middle of a call to a contact or broadcast it to multiple contacts simultaneously.

Platform Apple iOS (Objective C)

Voice Message Manager

Rcords a message on the ASC and sends that message to a single person or group in a contact list.

Platform RIM Blackberry (Java), Google Android (Java), JavaScript, Apple Objective C

WCF Auto-Mute Example

Utilizes ASC media scanning to detect when a party in a phone conversation is too loud and automatically mutes that caller’s phone.

Platform Microsoft.NET

WCF Interface Sample Client

Sends web services requests to an ASC and processes their responses using WCF classes.

Platform Microsoft .NET

Web Phone Example

Creates a fully web-based dialer via a GWT/GXT web application. Flash components are utilized to manage the browser-based media.

Platform Google GWT/GXT web application (Java)

Web Services Demo Suite

Consists of a Java server running on the Metro Web Server stack that features several call control features. This sample uses Java, the Metro Web Services Stac, the JavaMail API, JavaScript, JQuery, cross-domain XML HTTP requests using JSONP, SOAP over HTTP requests, and Comet long-polling. It also features a web-based user interface accessible from ASC-based user authentication.

Platform

Java

Appendix D ASC Call Reconnect SDK

Introduction

The ASC Call Reconnect Software Development Kit (SDK) is a Java library containing a Java API that supports processing calls that experience media loss due to network failures. This Java API calls the ASC Web Services API. You can use these Java classes to more quickly construct a Java call continuation application.

Note: The Call Reconnect SDK supports ASC versions 3.7.0M3P0 and newer.

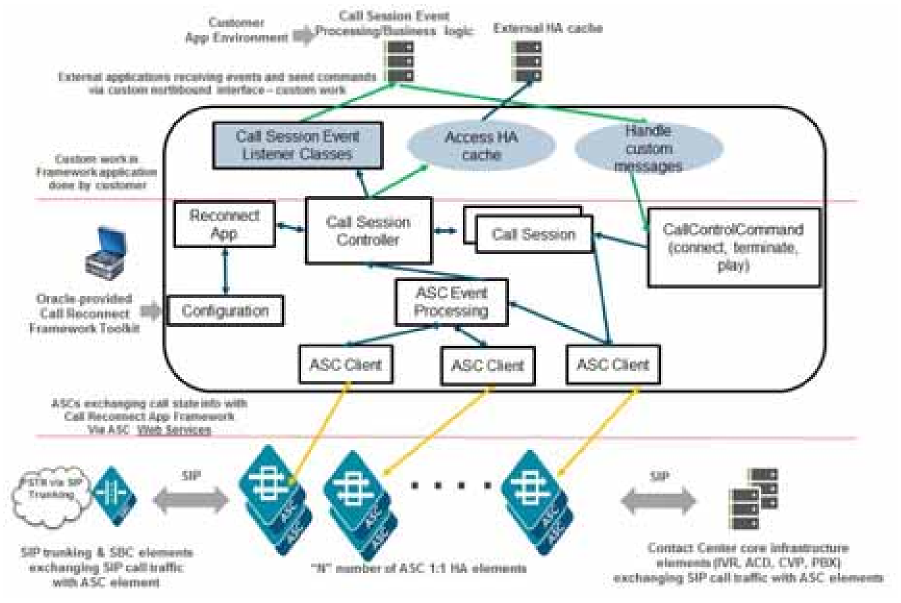
The Call Reconnect API:

- Acts as a service proxy for ASC API calls
- Subscribes to ASC events, including media loss and media resumption
- Supports storing and querying call continuation information and state pertaining to specific calls
- Supports playing .wav files to specific calls, reconnecting call legs to new SIP URIs, and terminating calls
- Extracts information from SIP headers and stores that information in the application's call information storage
- Preserves SIP header information into the redirected call
- Throttles the redirection of call legs to other phone numbers so that phone systems are not overwhelmed.

Additionally, by extending existing Call Reconnect classes or adding new classes to your application, you can implement storage of Call Reconnect data in a high availability (HA) enterprise cache of your choice, sending events to external applications, and integrating with external applications.

Note: The Call Reconnect SDK supports only calls with two parties such as a customer calling an agent and does not support conference calls.

The following diagram shows the Call Reconnect SDK's main classes. The Call Reconnect classes are displayed in white rectangles.


Customer App Environment
Call Session Event Processing/Business logic
External HA cache
External applications receiving events and send commands via custom northbound interface – custom work
Custom work in Framework application done by customer
Call Session Event Listener Classes
Access HA cache
Handle custom messages
Reconnect App
Call Session Controller
Call Session
CallControlCommand (connect, terminate, play)
Oracle-provided Call Reconnect Framework Toolkit
Configuration
ASC Event Processing
ASC Client
ASC Client
ASC Client
ASCs exchanging call state info with Call Reconnect App Framework Via ASC Web Services
PSTN via SIP Trunking
SIP
ASC
SIP
SIP trunking & SBC elements exchanging SIP call traffic with ASC element
"N" number of ASC 1:1 HA elements
Contact Center core infrastructure elements (IVR, ACD, CVP, PBX) exchanging SIP call traffic with ASC elements

The Call Reconnect SDK library includes Java classes to:

- Coordinate application-wide resources
- Read and apply application configuration information
- Manage ASC clusters
- Register for and receive ASC events

Call Reconnect Distribution

The Call Reconnect SDK library is distributed in a zip file and includes:

- Full configuration template for setting up a single ASC
- Merge configuration template for updating an already running ASC
- Full configuration template for setting up an ASC cluster

  Note: While you may use a single ASC for testing purposes, Oracle strongly recommends using an ASC cluster for production.
- call_reconnect.properties: Configuration for the Call Reconnect sample application (sample code that illustrates the use of the Call Reconnect SDK API)
- call_reconnect_binaries.zip: Includes the callreconnect.jar file for inclusion in your application; this jar file contains the binaries for the Call Reconnect application
- call_reconnect_sources.zip: Contains the Call Reconnect SDK sources and instructions on how to build the Call Reconnect SDK sources
- call_reconnect_javadoc.zip: Contains the HTML Javadoc files for the Call Reconnect API
- call_reconnect_sample_binaries.zip: Contains the .war file of the sample web application, instructions on how to deploy and configure the binary .war (properties file information), and information on securing web services; deployment and configuration instructions are in the *Call Reconnect Sample Deployment Guide*, CallReconnectSampleDeploymentGuide.docx

- call_reconnect_sample_sources.zip: Contains the source files for the sample and instructions on how to build it

Constructing Your Call Reconnect Application

The Call Reconnect SDK contains Java classes and interfaces to help you construct and manage your Call Reconnect application.

ReconnectApplication Class

The ReconnectApplication class reads configurations, initializes storage for call sessions, and manages ASC cluster web services endpoints.

For more information, see the Javadoc for ReconnectApplication in the call_reconnect_javadoc.zip file.

Configuration

The Configuration interface specifies an API for retrieving Call Reconnect configuration information. The PropertiesConfiguration class implements the Configuration interface and reads configuration settings from a Java properties file.

The ASC's Call Reconnect configuration interface also specifies an API for managing all ASC cluster web services endpoints.

Note: The Call Reconnect SDK supports alternate implementations of the Configuration interface that may use other configuration technologies such as Java Naming and Directory Interface.

For more information, see the Javadoc for the Configuration interface and the PropertiesConfiguration class in the call_reconnect_javadoc.zip file.

Dynamic Configuration Update

The ConfigurationController class updates configuration information so that you can update the Call Reconnect properties file without having to restart your application. You may implement a listener in your application to receive notifications when the Call Reconnect configuration has changed.

For more information, see the ConfigurationController and ConfigurationChangedListener classes in call_reconnect_javadoc.zip file.

Configured Default Commands

The Call Reconnect SDK supports configuring a set of commands to handle media loss in any configuration implementation (for example, PropertiesConfiguration).

The following is an example of default configured commands:

```
OutLegCmdOnMediaLost.1.name=Terminate
OutLegCmdOnMediaLost.1.order=1
OutLegCmdOnMediaLost.1.param.waitCompletion=false
OutLegCmdOnMediaLost.1.param.callLeg=OUT
```

For more information, see the Javadoc for the CommandConfiguration class in call_reconnect_javadoc.zip file.

Web Services Connection and Event Subscription

The ASCClient class allows you to access all ASC Web Services via SOAP and register for ASC events using primary and secondary event callback endpoints to support HA. The backup endpoint is implemented in another instance of your Call Reconnect application running on another server on a separate Java Virtual Machine (JVM).

ASCClient communicates with an ASC cluster that has Web Services and SIP support configured on a Virtual Routing Redundancy Protocol (VRRP) interface. The ASC VRRP transparently supports failover for Web Services and SIP calls.

For more information, see Javadoc for the ASCClient class in the call_reconnect_javadoc.zip file.

Event Handling

The ASCEventProcessingService class receives and handles ASC events, passing the received event to an instance of the CallSessionEventDispatcher class, which queues the event in the relevant CallSession instance's event queue to await processing.

When configured, the application receives media loss and media resumption events directly from the ASC. These events can be enabled by adding the following configuration to the appropriate **vsp > session-config-pool > entry**.

```
# Enable media loss events and set interval
   config out-media-loss-detection
    set admin enabled
# interval for checking media loss
    set interval "0 days 00:00:05"
   return
```

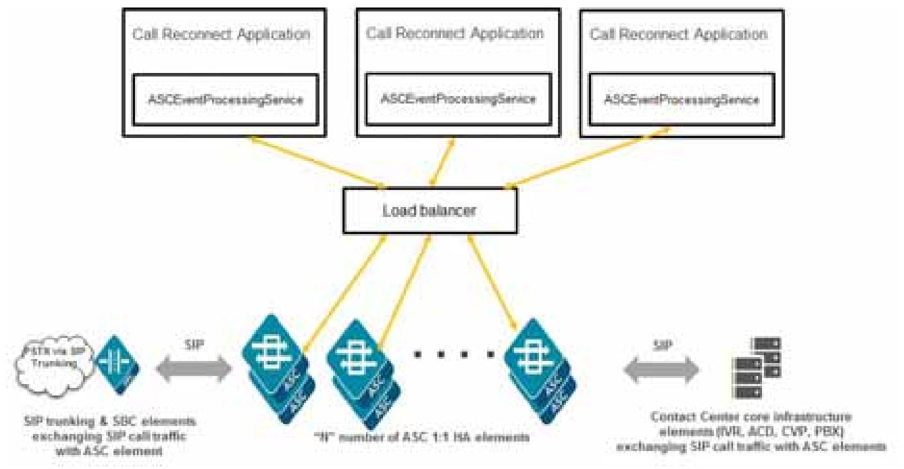
The rate at which the media sessions are checked is based on the configured interval. The above example shows that the interval is 5 seconds. Once the ASC detects a media loss that lasts for the media loss interval time you specify, it issues a media loss event.

For more information, see Javadoc for the ASCEventProcessingService class in the call_reconnect_javadoc.zip file.

Load Balancer Support

To balance the load of handling ASC events among Call Reconnect application nodes, you can use a load balancer between one or more ASC clusters and one or more externally running Call Reconnect application instances.

You can also disable event registration within the Call Reconnect application and use an alternative ASC event registration scheme that registers the load balancer as the direct recipient of ASC events.


Call Reconnect Application
ASCEventProcessingService
Call Reconnect Application
ASCEventProcessingService
Call Reconnect Application
ASCEventProcessingService
Load balancer
PSTN via SIP Trunking
SIP
ASC
ASC
SIP
SIP trunking & SBC elements exchanging SIP call traffic with ASC element
"N" number of ASC 1:1 HA elements
Contact Center core infrastructure elements (IVR, ACD, CVP, PBX) exchanging SIP call traffic with ASC elements

When it receives an event, the Call Reconnect application must decide which ASC the event came from. When the application receives an event, the ASCEventProcessingService class determines the ASC sender's IP address by examining the X-Forwarded-For (XFF) HTTP Header. If the XFF header is missing (because there is no proxy or load balancer intermediary), the sender's IP address is retrieved via the javax.servlet.http.HttpServletRequest getRemoteAddr() method.

Enabling and Disabling Event Registration

Via the isEventRegistrationEnable and setEventRegistrationEnable methods, you can disable event registration in the Call Reconnect application, either in the call_reconnect.properties file or in an implementation of the Configuration interface such as the PropertiesConfiguration class. You can also enable or disable event registration in the call_reconnect.properties file by setting the ADVANCED.eventRegistration.enable entry to true or false (by default this value is true).

If the application's event registration is disabled in a load balancer deployment, you must either configure static registration on the ASC clusters to point to the load balancer, or create an ASC application that dynamically registers the load balancer with all ASC clusters.

For more information, see the Javadoc for the AdvancedConfiguration methods in the call_reconnect_javadoc.zip file.

Call Session Stickiness

When deploying a call continuation application on more than one system for HA support, Oracle recommends directing all events for any specific call to the same application instance to reduce hits on the enterprise cache. This makes all of the processing for a call session "stick" to the same application instance.

To do this the load balancer must use one of the following methods:

- Examine the HTTP header and extract the sender's ASC IP address. Forward events from the same sender's ASC IP address to a specific Call Reconnect application instance. Assuming there is a SIP load balancer in front of the ASC clusters, this approach results in a balanced load until a Call Reconnect application instance fails.

- Examine the SOAP message and extract the session ID from the event content. Send events with the same session ID to the same Call Reconnect application instance, ensuring that any given SIP call session is processed by only one Call Reconnect application instance until that instance fails and another instance takes over processing.

Call Management

The call management classes in the Call Reconnect SDK manage each call being processed by any of the ASC clusters with which your application is communicating.

The Call Reconnect SDK contains several Java classes to:

- Store information about all the calls being processed by the ASC clusters with which your application is communicating
- Query information about these calls
- Process ASC call events

CallSession

The CallSession class maintains information about live calls on an ASC cluster including a hash map that stores attribute names and values from the SIP headers of these calls. The Call Reconnect application maintains the following information in the CallSession:

- ASC IP (The Web Services IP address for the ASC cluster handling the call, letting the application know which cluster is handling the call)
- Session ID
- Out-leg information (party receiving call)
- In-leg information (party initiating call)
- Reconnect leg information if applicable (the new party that replaces the leg experiencing media loss)
- In-leg or out-leg media loss if applicable

You may use custom code to call the API to store more information in the CallSession to add more attribute names and values to the hash map.

For more information, see the Javadoc for the CallSession class in the call_reconnect_javadoc.zip file.

CallLeg

The CallSession class contains CallLeg data representing the in-leg, out-leg, and reconnect-leg in a call session. Each CallLeg instance contains information not limited to the following:

- Call ID
- SIP URI for “from” and “to” endpoints
- Leg type: in-leg (the party that initiated the call; the “from” endpoint), out-leg (The party receiving the call; the “to” endpoint), or reconnect leg

For more information, see the Javadoc for the CallLeg class in the call_reconnect_javadoc.zip file.

CallSessionController

The CallSessionController class maintains a map of CallSession instances keyed by a combination of the ASC Web Services IP and session ID of the ASC handling the call. It calls registered listeners when any CallSession instance is added, removed, or updated.

For more information, see the Javadoc for the CallSessionController class in the call_reconnect_javadoc.zip file.

Call Information Event Processing

CallSessionListener is an interface that provides an API to receive notifications for the following events related to an ASC call:

- CallSession instance created
- CallSession instance removed
- Call leg created
- Call leg in the process of connecting to call session
- Call leg connected to call session
- Call leg terminated
- Call leg held
- Call leg attached
- Command invoked or completed on call leg
- Media lost on call leg
- Media resumed on call leg

Every CallSessionListener's method contains a parameter that is the entire ASC event object for further use by the CallSessionListener implementation. You can use the CallSessionListener's onCallLegOtherEvent method to examine ASC events that are not explicitly handled by the Call Reconnect application.

For more information, see the Javadoc for the CallSessionListener's method in the call_reconnect_javadoc.zip file.

DefaultMediaLostHandler

The DefaultMediaLostHandler class executes commands that are configured in your Call Reconnect application's configuration when the application receives media lost events. If your application is using the PropertiesConfiguration class, you can configure these commands for your Call Reconnect application in the call_reconnect.properties file.

For more information, see the Javadoc for the DefaultMediaLostHandler class in the call_reconnect_javadoc.zip file.

Configured Custom CallSessionListener

The Call Reconnect SDK allows you to optionally configure custom call session listeners, which are instantiated during the application initialization.

The custom call session listeners must meet the following requirements:

- Have a public no-argument constructor
- Implement the oracle.asc.reconnect.call.listener.CallSessionListener interface

When configured, the application instantiates the specified listener and adds the listener to the list of CallSessionListener maintained by the SessionController.

If your application is using the PropertiesConfiguration class, you may specify multiple classes in call_reconnect.properties in the format:

```
CallSessionListener.<n>.class=com.foo.bar.MyCallSessionListener
```

where <*n*> is a positive number.

For more information, see the CallSessionListener interface and PropertiesConfiguration class in the call_reconnect_javadoc.zip file.

Using Default Media Loss Handler With Custom Call Session Listener

The Call Reconnect SDK allows you to use both a default media loss handler and a custom call session listener. This is useful if there are commands you execute regularly to handle media loss and you must determine further processing programmatically based on other information.

Command Processing

The Call Reconnect SDK includes several Java classes to:

- Redirect a call leg to a new SIP URI
- Play a file to a call leg
- Terminate a call leg
- Stop playing a file to a call leg

To execute a command, create a list of one or more of the command class instances and invoke CallSession.startCommandInvocation on the CallSession instance for the SIP call session you are processing.

> **Note:** All Call Reconnect command classes have an option to wait for completion.When **true**, the Call Reconnecct application waits until it receives the events from the ASC indicating the command is complete before executing the next command in the command list. When **false**, the application executes the next command in the list immediately after receiving a success status for executing the command. If this option is not provided, the default behavior is false.

Redirecting a Call Leg to a New SIP URI

The CallControlConnectCommand redirects a call leg to a new SIP URI.

The CallControlConnectCommand class contains support to:

- Preserve SIP Header content in SIP messages for a new call, preserving custom SIP Header content
- Throttle redirect commands sent to the ASC, including parameters to specify a queue name. The application retrieves the number of calls to be sent at a time and the

amount of time to wait before sending the next batch of call redirects from the application configuration.

For more information, see the Javadoc for the CallControlConnectCommand class in the call_reconnect_javadoc.zip file.

Including SIP Header With Named Variable Support

The CallControlConnectCommand class has support to inject a new SIP header into the INVITE message in a SIP session with a specified SIP header name and value.

This class also supports using named variables specified in the properties file as the source of SIP header information to inject into the INVITE message. The application calls the CallControlConnectCommand methods to generate the new SIP header with the named variable name as the SIP header name and the named variable value as the SIP header value.

For more information, see the Javadoc for the CallControlConnectCommand and CommandConfiguration classes in the call_reconnect_javadoc.zip file.

Throttling

Throttling allows you to control the rate at which calls are redirected. Enable throttling by setting the maximum number of calls per second (Queue.default.maxCallsPerSec) for the default queue. The throttling system does not send the maximum calls per second all at once, but divides the calls into batches using the check-interval configuration property (Queue.default.checkInterval).

Default Queue

By default, all calls to be redirected are placed into a single queue. If not specified, the check-interval property (Queue.default.checkInterval) defaults to 100 milliseconds. The throttling system sends a portion of the calls on the queue, sleeps for the specified check-interval time, and then sends the next batch of calls. The batch size is the maximum number of calls (Queue.default.maxCallsPerSec) * (check interval/1000 milliseconds).

Additional Queues

If your Call Reconnect application redirects calls to more than one IVR system, you can specify additional numbered queues. For each additional queue, you must specify its redirect SIP URI. Calls to be redirected to those SIP URIs go to that queue.

Note: If you do not specify a maximum number of calls per second or batch interval, the application defaults to the default queue maxinum number of calls and check intervals, respectively.

You may specify additional queues in call_reconnect.properties in the following format:

```
Queue.<n>.phones
Queue.<n>.maxCallsPerSec
Queue.<n>.checkInterval
```

where <*n*> is a positive number.

For more information, see the Javadoc for the ThrottleQueueConfiguration, CallConnectThrottler, and CallConnectThrottlerQueue classes in the call_reconnect_javadoc.zip file.

Redirecting Call Legs Using Park and Attach

The CallControlParkAttachCommand class is an alternative to using the CallcontrolConnectCommand. CallControlParkAttachCommand processes the new agent leg first and then the customer leg by issuing the following ASC actions:

- The **call-control-custom** action parks the new agent SIP URI and sends the Cisco-Guid value in the INVITE to the new agent
- The **call-control-attach** action either parks the new agent call leg handle to the customer session or parks the customer leg to the new agent session.

CallControlParkAttachCommand supports injecting a SIP Header with the value of a named variable as well as throttling.

For more information, see the Javadoc for the CallControlParkAttachCommand class in the call_reconnect_javadoc.zip file.

Selecting Which Leg Attaches To a Session

The Call Reconnect SDK supports an advanced configuration option in the call_reconnect.properties file, called ADVANCED.attachAgentLegToInLeg. When **true** (the default), the application attaches the new agent leg to the customer session. When **false**, the application attaches the customer leg to the new agent session.

This allows you to affect the order in which the customer leg and the reconnected agent leg are negotiated.

For more information, see the Javadoc for the AdvancedConfiguration class in the call_reconnect_javadoc.zip file.

Playing a File To a Call Leg

CallControlPlayMediaCommand class plays a .WAV file stored on an ASC cluster to a call leg. You have the option to play the file in a loop until a StopMediaCommand is issued.

- **Note:** When this command’s “waitCompletion” and “loop” options are set to **true**, the application ensures that the file is played through completely once before aborting subsequent repeats due to reconnecting to a new SIP URI.

For more information, see the Javadoc for the CallControlPlayMediaCommand class in the call_reconnect_javadoc.zip file.

Terminating a Call Leg

CallControlTerminateCommand terminates a call leg, leaving the other call leg on the ASC in a parked state.

For more information, see the Javadoc for the CallControlTerminateCommand class in the call_reconnect_javadoc.zip file.

Stopping Playing a File To a Call Leg

CallControlStopMediaCommand aborts the playing of a .WAV file in progress to a call leg.

For more information, see the Javadoc for the CallControlStopMediaCommand class in the call_reconnect_javadoc.zip file.

Command Success and Failure Notifications

The CallSessionListener interface includes a method for notifying you of command success and failure. The onCommandStatusChanged method notifies the listener of one of the following:

- A command in the queue for a CallSession has been invoked and whether the command invocation was successful or returned an error
- A command was completed successfully or was aborted due to a specific error

SIP Header Content Extraction

The Call Reconnect SDK supports retrieving the content of any SIP Header and storing it in the CallSession attribute hash map for the related ASC session.

The following examples show the steps you must take on both the ASC and Call Reconnect application to extract SIP Header content.

ASC:

1. Configure the ASC to extract the SIP header value from the SIP 200 OK response header and add the name value pair (name=<*value*>) into the event.

   - Ensure you have the **vsp > default-session-config > third-party-call-control > admin** property set to **enabled**.
   - In the **vsp > default-session-config** object, configure the **inbound-header-setting** object and add a **named-variable-collector**. This configuration selects the entire <*SIP header name*> from the received 200 OK response and store it in the named variable called "*example-name*". For example:

   ```
   config inbound-header-settings
   config named-variable-collector 1
   set named-variable example-name
   set create Accept (.*) "\1" custom
   set apply-to-responses yes 200
   return
   return
   ```

   - In the **vsp > default-session-config** object, configure the **event-settings** object to insert the value of the "*example-name*" named variable into the events with the value of the specified SIP header. For example:

   ```
   config event-settings
   set named-variable-entry example-name <display-name>
   return
   ```

Call Reconnect Application:

1. Add the "*display-name*" named variable to the Call Reconnect property file. When Call Reconnect application restarts, the property file is read and the "*example-name*" named variable is stored internally.
2. When a call event arrives, the event contains the name value pair (*display-name*=<*value*>). The event processor extracts *display-name*=<*value*> from the event and stores the value in the CallSession instance with the attribute name "*display-name*".

Generating Call Session Reporting Files

The Call Reconnect SDK supports generating files containing information about call sessions with media loss. The SDK supports basic Comma Separated Value (CSV) file generation through the WriteCSVListener class as well as other file formats through custom inplementations of the AbstractWriterListener interface.

WriteCSVListener

The Call Reconnect SDK contains an implementation of AbstractWriterListener called WriteCSVListener. When this class is specified in the Call Reconnect WriterListener configuration, the application generates a CSV file in the specified directory on the system running the application. The application creates one CSV file for all calls experiencing media loss each day.

The CSV file contains the following information about a session:

- Session key
- From-leg call URI
- To-leg call URI
- Reconnect leg call URI
- Media lost time
- From-leg connected time
- To-leg connected time
- Reconnect leg connected time

You must remove the CSV files from the system for further processing or the files will continue to be generated, taking up disk space.

For more information, see the Javadoc for the WriteCSVListener class in the call_reconnect_javadoc.zip file.

AbstractWriterListener

You can use the AbstractWriterListener interface to implement a Java class that writes any kind of file on the system running the application. For example, it could be used to write XML files containing call sessions with media loss information. The Call Reconnect configuration allows you to configure a list of classes.

For more information, see the Javadoc for the getWriterListenerConfigurations method, PropertiesConfiguration class, and AbstractWriterListener interface in the call_reconnect_javadoc.zip file.

Enterprise Cache Support

The Call Reconnect SDK supports storing call session and connect throttle queue information in a cache that can be used to connect several application instances for HA and scalability. By coding your own class that implements the EntityStore API, you can integrate your choice of an enterprise cache into your Call Reconnect application.

Any time the application creates, updates, or deletes CallSession or CallConnectThrottleQueue instances, the application calls the appropriate EntityStore method and updates the corresponding item.

Cache Interface Classes

The cache interface class is oracle.asc.reconnect.store.EntityStore, which contains the following methods for cache access.

- Add or update item to the cache
- Retrieve item from the cache
- Remove item from the cache
- See if item is contained in the cache
- Lock item in the cache so that calling thread has exclusive access
- Unlock item in cache so any thread in any Call Reconnect application node can access item

The lock and unlock methods support a scenario where there are two Call Reconnect application nodes actively handling calls, throttling is enabled and there is only the default throttling queue. Call Reconnect nodes 1 and 2 each place a call on the queue. The application calls the EntityStore lock method so that the Call Reconnect node processing the call has exclusive access to it and then calls the EntityStore.unlock method when the processing is done. The cache needs to take care of locking because the Call Reconnect nodes are in separate JVMs.

For more information, see the Javadoc for the EntityStore interface in the call_reconnect_javadoc.zip file.

XML Serialization Classes

The Call Reconnect SDK calls EntityStore to be stored in the cache. If you want to store these objects in XML form, call the CallSessionXMLSerializer and ConnectQueueXMLSerializer classes in your EntityStore interface implementation.

CallSession XML Serializer

The Call Reconnect SDK includes a class called CallSessionXMLSerializer that supports de-marshaling a CallSession instance to an XML string or marshaling a correctly structured XML string into a CallSession instance.

For more information, see the Javadoc for the CallSessionXMLSerializer class in the call_reconnect_javadoc.zip file.

CallConnectThrottlerQueue XML Serializer

The Call Reconnect SDK includes a class called ConnectQueueXMLSerializer that supports converting a CallConnectThrottlerQueue instance into an XML format and from XML into a CallConnectThrottlerQueue instance.

For more information, see the Javadoc for the ConnectQueueXMLSerializer in the call_reconnect_javadoc.zip file.

Specifying Cache Store Implementation

You can specify your cache store implementation in the Call Reconnect properties file or in any implementation of the oracle.asc.reconnect.config.Configuration interface (if you are not using the properties file).

Properties File

In the call_reconnect.properties file, specify the following:

```
EntityStore.class=com.<mycompany.MyEntityStore>
```

where <*mycompany.MyEntityStore*> is the class name of your EntityStore implementation.

Configuration Interface

The oracle.asc.config.Configuration interface contains a getEntityStoreClassName() method to return the class name of your EntityStore implementation.

Customizing Functionality

For extended events and integration with external applications, you can code and configure customized functionality.

Extended Events

Implement the CallSessionEventListener class to generate your own custom events based on information in the ASC event being processed. For massive failure events, the CallSession Controller getSessionCountByAttributeValue returns counts of calls that have experienced media loss for failure size subscription.

Integration With External Applications

Sending events to and receiving instructions from external applications is custopm code that you must add to the application to integrate with your Call Reconnect application.

Troubleshooting the SDK

This section describes troubleshooting techniques to help you gather relevant data when issues arise with your Call Reconnect application.

Call Reconnect SDK troubleshooting steps include:

- Updating the ASC configuration to make collecting troubleshooting data easier
- Enabling SIP and Web Services traces to capture relevant data
- Reproducing the problem and terminating the traces
- Gathering the troubleshooting data

Updating the ASC Configuration

Prior to even encountering a situation requiring troubleshooting, Oracle recommends updating your ASC configuration that makes collecting troubleshooting information easier.

Oracle recommends the following updates:

- When running your Call Reconnect application on the ASC Web Services virtual host, configure a **services > collect > collect-group** to automatically collect the local Call Reconnect properties and Call Reconnect log files
- Enable call logs to be collected in the ASC database and add Web Services and other event logs

Including Call Reconnect Properties and Log Files in Collect

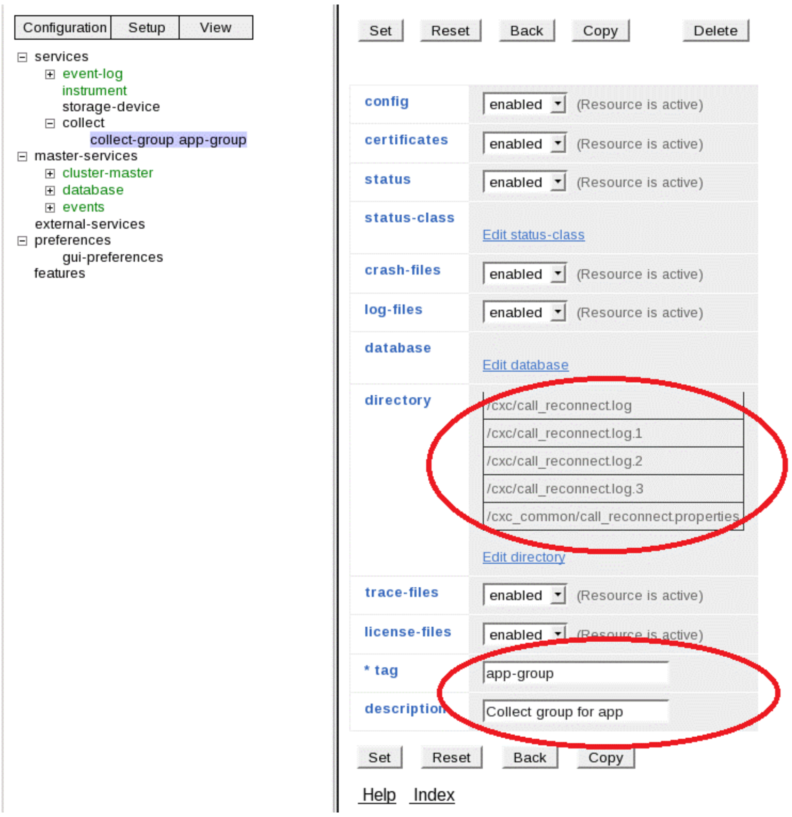
Using the ASC **collect** action, you can gather various files, logs, and traces on the ASC into a compressed file. When running your Call Reconnect application on the ASC cluster, you can configure your system so that running the **collect** action automatically includes the Call Reconnect properties and log files in the resulting tar.gz file.

To include Call Reconnect properties and log files in your collect file:

1. Launch the ASC web GUI and log in.
2. Select the **Services** tab.
3. Click the **services** object and click **collect**.

   Note: The ASC creates a default collect configuration object. Oracle recommends using the default values.
4. Click **Add collect-group**. A one-page wizard appears.
5. Specify **tag** as **app-group** and click **Create**.
6. Leave the default values and click **Edit directory**.
7. Add the following file names one at a time, clicking the **Add** button after adding each file specification.
   - /cxc/call_reconnect.log
   - /cxc/call_reconnect.log1
   - /cxc/call_reconnect.log2
   - /cxc/call_reconnect.log3

- /cxc_common/call_reconnect.properties



config	enabled (Resource is active)
certificates	enabled (Resource is active)
status	enabled (Resource is active)
status-class	Edit status-class
crash-files	enabled (Resource is active)
log-files	enabled (Resource is active)
database	Edit database
directory	/cxc/call_reconnect.log /cxc/call_reconnect.log.1 /cxc/call_reconnect.log.2 /cxc/call_reconnect.log.3 /cxc_common/call_reconnect.properties Edit directory
trace-files	enabled (Resource is active)
license-files	enabled (Resource is active)
* tag	app-group
description	Collect group for app

8. Click **Set**. Update and save the configuration.

Storing Call Log Files

To configure the ASC to store call logs in the ASC database:

1. Launch the ASC web GUI and log in.
2. Select the **Configuration** tab.
3. Select the **vsp** > **default-session-config** > **log-alert** object.
   - **message-alert**—Set to **enabled**.
   - **apply-to-methods-for-filtered-logs**—Click **Select All**.
4. Click **Set**.
5. Select the **Services** tab.
6. Select the **master-services** > **database** object (if the object has not already been configured) and configure it.
7. Click **Set**. Update and save the configuration.

Enabling SIP and Web Service Traces

You must enable two traces for Call Reconnect SDK troubleshooting: the SIP trace file to gather information about the SIP process and the Web Services trace to gather information about the Web Services WS process.

Enabling SIP Trace Files

By default, the ASC's **collect-group** > **traces** property is **enabled** so that SIP traces are collected as part of the collect-tar.gz file when you run the **collect** action.

To enable the SIP trace:

1. Launch a PuTTY (or equivalent client) session and log into the ASC.
2. Specify the settings in the following example for trace source, filter, and severity.

   Note: When prompted with **Do you want to start tracing for this target (y or n)?**, if you answer **y**, tracing begins immediately and **n** does not start tracing.

   ```
   shell sip
   SIP>trace source <source>
   trace app_sip>trace *error
   trace app_sip>trace sip_traffic info
   trace app_sip>trace scale_call debug
   trace app_sip>trace media debug
   trace app_sip>trace event debug
   trace app_sip>exit
   Do you want to save the settings for this target (y or n)? y
   Do you want to start tracing for this target (y or n)? n
   SIP>exit
   ```
3. Use the following ASC CLI commands to start and stop tracing. Trace <*source*> is the name specified in the previous step.
   - **trace start <*source*>**
   - **trace stop <*source*>**

Enabling Web Services Trace Files

To enable the Web Services Trace:

1. Launch a PuTTY (or equivalent client) session and log into the ASC.

   Note: Ensure the Window lines of scrollback for your PuTTY (or equivalent client) configuration is set to a large number (for example, 99999).
2. Issue the following commands:

   ```
   shell ws
   log tofile start /cxc_common/log/wsdebug.txt
   log last
   log on
   log stack on
   log level debug
   ```

   Note: If you do not specify **log tofile start**, copy the PuTTY output to a file on your system when the test is complete.

When you reproduce the problem, the Web Services debug information is printed out in the PuTTY screen and is stored on the ASC in the /cxc_common/log/wsdebug.txt file.

Terminating Traces and Saving Trace Files

To stop the SIP trace:

1. Type the following command:

   ```
   trace stop <source>
   ```

The SIP trace file is automatically saved in the /cxc_common/log directory.

To stop the Web Services trace:

> **Note:** It is important to terminate the Web Services trace file with the **log tofile stop** command. If you allow the trace to continue, the file will continue to grow, consuming disk space.

1. Type the following command:

   ```
   log tofile stop /cxc_common/log/wsdebug.txt
   ```

   Note: If you did not execute the **log tofile start** command, copy the output of the PuTTY session to a file on your system and SCP this file to /cxc_common/log.

2. Type exit to **exit** the ws shell.

The /cxc_common/log/wsdebug.txt file is automatically included in the collect.tar.gz if you do not change the **collect-group** > **app-group** > **log-files** property collect default value (enabled).

Collecting Debugging Information

To collect debugging information:

1. Get the Call Logs file and copy it to a specific directory on the ASC.
2. Execute the **collect** action.

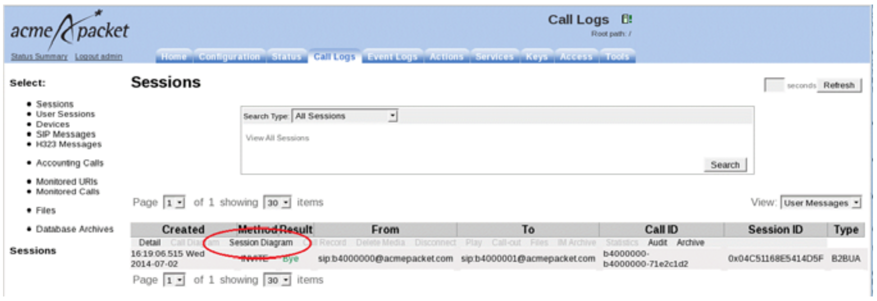
   Note: Follow instructions for the collect carefully to ensure the collect includes the configured **app-group** and is run on the cluster.

Obtaining the Call Logs File

To obtain the Call Logs file for a particular call:

1. Launch the ASC web GUI and log in.
2. Select the **Call Logs** tab.
3. Click **Sessions** and click on **Session Diagram** for your call.

   The Call Sequence is displayed for the selected session.



Sessions

Created	Method	Result	From	To	Call ID	Session ID	Type
16:19:06.515 Wed 2014-07-02	INVITE	Bye	sip:b4000000@acmepacket.com	sip:b4000001@acmepacket.com	b4000000-b4000000-71e2c1d2	0x04C51168E5414D5F	B2BUA

4. Click **Save as text**. This file contains SIP messages for the call flow and is saved in your local file system.
5. Rename the saved file to call_flow.txt and, using a utility such as SCP, push the file to your ASC directory /cxc_common/log/call_flow.txt.

Collecting Troubleshooting Files

Prior to collecting troubleshooting files, ensure the SIP and Web Services debug log files are in the directory /cxc_common/log/.

Once the call or test is complete, manually copy the SIP and Web Services debug log files to the primary ASC and issue the following command:

```
collect app-group cluster
```

Files for Call Reconnect troubleshooting are collected automatically and added to the collect set. When done, the file /cxc_common/collect/collect.tar.gz is created in both the primary and standby ASCs.

> **Note:** Ensure you run the **collect** action with these exact arguments. If not, the collect misses information from the standby box and does not include the Call Reconnect properties and log files.

When the **collect** action is complete, send the files /cxc_common/collect/collect.tar.gz from both the primary and standby ASCs to support for troubleshooting.

Troubleshooting an External Call Continuation Application

When you are working with Oracle field personnel or support to troubleshoot an external application using the Call Reconnect SDK, send the following files in addition to the cxc_common/collect/collect.tar.gz files:

- call_reconnect.properties
- call_reconnect.log
- Application server log (for example, Tomcat, IBM WebSphere, or Oracle WebLogic)

Checking Relevant Status Reports

The following status reports are helpful in diagnosing issues. You can either execute these status reports via the CLI with a show command or in the web management UI by clicking on the **Status** tab and clicking on the status report link.

To view ASC event handling information use the following show command:

```
show dynamic-event-services -v
```

> **Note:** If you do not see the Web Services SOAP endpoint for your application in this list, your application is not registered and is not receiving media loss events.

To view if your application is running on the Web Services virtual host, use the following show commands:

```
show web-services-virtual-hosts
show web-services-virtual-hosts-applications
```