Oracle® Communications Services Gatekeeper

Application Developer's Guide

Release 7.0

E95425-01

July 2018

ORACLE®

Oracle Communications Services Gatekeeper Application Developer's Guide, Release 7.0

E95425-01

Contents

8 Adding RESTful Short Messaging Support

9 Adding RESTful Multimedia Messaging Support

10 Adding RESTful Email Communication Service Support

11 Adding RESTful Terminal Location Support

12 Adding RESTful Payment Support

13 Adding RESTful Quality of Service Support

14 Adding RESTful Device Capabilities Support

15 Adding RESTful Binary Short Messaging Support

16 Adding RESTful Session Manager Support

17 Adding RESTful Subscriber Profile Support

18 Adding RESTful WAP Push Support

Part III Creating Applications Using the OneAPI RESTful Interfaces

19 Using the OneAPI RESTful Interfaces

24 About the Supported SOAP Parlay X 3.0 Facades

25 About the Supported SOAP Parlay X 4.0 Facades

26 About the Supported SOAP Native Facade

30 Adding Subscriber Profile Extended Web Service Support

Preface

This document describes how to integrate functionality provided by telecom networks into applications by using the SOAP and REST-based facades offered by Oracle Communications Services Gatekeeper. It includes a high-level overview of the application development process, including the login and security procedures, and a description of the interfaces and operations.

This document covers the SOAP and RESTful interfaces and the native interfaces available in Services Gatekeeper.

Audience

This book is intended for software developers who will integrate functionality provided by telecom networks into their applications using the SOAP-based, RESTful, and native interfaces.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

The following documents provide information related to creating applications that interact with Services Gatekeeper:

- *Oracle Communications Services Gatekeeper Concepts*
- *Oracle Communications Services Gatekeeper Portal Developer's Guide*
- *Oracle Communications Services Gatekeeper Alarms Handling Guide*
- *Oracle Communications Services Gatekeeper Communication Service Reference Guide*
- *Oracle Communications Services Gatekeeper Extension Developer's Guide*
- *Oracle Communications Services Gatekeeper Platform Test Environment User's Guide*

Part I

Overview of Creating Applications for Services Gatekeeper

Part I provides an overview of Oracle Communication Services Gatekeeper, and explains its capabilities and features.

Part I contains the following chapters:

1

About Creating Applications that Interact with Services Gatekeeper

This chapter presents an overview of how you can develop applications that interact with Oracle Communications Services Gatekeeper.

See *Services Gatekeeper Concepts* for a complete description of the Services Gatekeeper structure and functionality.

Basic Concepts

These are the basic concepts you should understand before creating applications that can interact with Services Gatekeeper:

- Understanding the Interfaces
- Understanding Communication Services
- Understanding Traffic Types
  - Understanding Application-Initiated Traffic
  - Understanding Network-Triggered Traffic
- Understanding Mobile Applications
- Understanding Management Structures
- Understanding How to Test Applications

Understanding the Interfaces

In order to interact with Services Gatekeeper, applications use SOAP-based, RESTful, OneAPI, or native interfaces. For details about using these interfaces, see:

- Using the RESTful Interfaces
- Using the SOAP Interfaces
- Using the OneAPI RESTful Interfaces
- Understanding the Extended Web Services Common Definitions

Also, see "Understanding the Supported Application Interfaces" for a list of the supported interfaces and a brief description of each.

Understanding Communication Services

The basic functional unit in Services Gatekeeper is the communication service. A communication service consists of a service type (Short Messaging, User Location, and

so on), an application-facing interface (also called a *north* interface), and a network-facing interface (also called a *south* interface). See "Understanding Traffic Types" for details about these traffic types.

A request for service enters the communication service through one interface, is processed internally, including evaluation for policy actions and protocol translation, and then sent on using the other interface.

For an overview of communication services, see *Services Gatekeeper Concepts*.

For details about the communication services that Services Gatekeeper includes, see *Services Gatekeeper Communication Service Reference Guide*.

For details about administering and deploying communication services, see *Services Gatekeeper System Administrator's Guide*.

For information about creating your own custom communication services, see *Services Services Gatekeeper Extension Developer's Guide*.

Note: A single application-facing interface may be using multiple protocols and hardware types in the underlying telecom network. However, an application is communicating, finally, with a specific communication service, and not only with the application-facing interface. So in some cases it is possible to send an application request to two different carriers that use different underlying network structures where the request behaves in slightly different ways, even though the initial request uses the same application-facing interface.

Understanding Traffic Types

In some communication services, request traffic can travel in two directions: from the application to the underlying network and from the underlying network to the application.

Understanding Application-Initiated Traffic

In application-initiated traffic, the application sends a request to Services Gatekeeper, the request is processed, and a response is returned synchronously. For example, an application could use the Third Party Call interface to set up a call. The initial request, **MakeCall**, is sent to Services Gatekeeper (which sends it on to the network). A string, the **callIdentifier**, is returned to the application synchronously. To find out the status of the call, the application makes a new request, **GetCallInformation**, using **callIdentifier** to identify the specific call. The application then receives the requested information back from Services Gatekeeper synchronously.

Understanding Network-Triggered Traffic

In many cases, application-initiated traffic provides all the functionality necessary to accomplish the desired tasks. But there are certain situations in which useful information may not be immediately available for return to the application. For example, the application might send an SMS to a mobile phone that the user has turned off. The network won't deliver the message until the user turns the phone back on, which might be hours or even days later. The application can poll to find out whether the message has been delivered by using the **GetSmsDeliveryStatus** request which functions such as **GetCallInformation** for application-initiated traffic. But it would be more convenient to have the network notify the application when the message has been delivered to the mobile phone. To do this, two things must happen:

- The application must inform Services Gatekeeper that it wants to receive information that originates from the network. It does this by subscribing or registering for notifications using an application-initiated request. (In certain cases, registering can also be accomplished by the operator, using Oracle Access Manager (OAM) procedures.) Often this subscription includes filtering criteria that describes exactly what kinds of traffic the application wants to receive. Depending on the underlying network configuration, Services Gatekeeper itself, or the operator using manual steps, informs the underlying network about the kind of data that is requested. These notifications may be status updates or, in some instances, may even include short or multimedia messages from a terminal on the telecom network.
- The application must arrange to receive the network-triggered information, either by implementing a Web service endpoint on its own site to which Services Gatekeeper dispatches the notifications, or by polling Services Gatekeeper to retrieve them. Notifications are kept in Services Gatekeeper for retrieval.

Securing Applications from Malicious Traffic

See "Securing Applications Against Malicious Traffic" in *Services Gatekeeper Security Guide* for details on how to protect Services Gatekeeper from malicious REST and SOAP traffic.

Adding Proxy Servers for Callbacks and Notifications

You can specify proxy servers to receive notifications or callback messages by adding them to the service provider or application SLA. Use the `<proxyhost>` and `<proxyport>` elements to specify the IP address and port number to listen on.

For details and an example, see the `<proxyhost>` and `<proxyport>` elements in *Services Gatekeeper Accounts and SLAs Guide*.

Understanding Mobile Applications

Application developers can create mobile applications running on devices such as smartphones and tablets that communicate with the Services Gatekeeper interfaces. Generally, the software development kit (SDK) for a mobile operating system, provided by the operating system vendor, includes the required tools for an application to communicate with Services Gatekeeper.

The following general guidelines list the basic steps used when developing a mobile application that communicates with Services Gatekeeper. The example provided uses the Google Android SDK. Though methods for interacting with web interfaces vary by operating system, the general procedure for other operating systems, such as Apple iOS or Microsoft Windows Phone, are similar.

To develop mobile applications that interface with Services Gatekeeper, consult the following sections:

- Preparing a Development Environment
- Creating a Mobile Application
- Testing a Mobile Application
- Distributing a Mobile Application

Preparing a Development Environment

You must set up a development environment before developing a mobile application for use with Services Gatekeeper. You must include downloading and configuring the

appropriate mobile operating system SDK, JDK and integrated development environment (IDE) such as Eclipse. For example, download the Android application development tools needed from the following web sites:

- Google Android Developer Tools (ADT) bundle:
  http://developer.android.com/sdk/index.html

  The link above also provides a bundle for use with an existing IDE.
- Java Platform (JDK):
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Eclipse IDE: http://www.eclipse.org/downloads/

Configure your application development environment according to the requirements listed by your mobile operating system vendor. After setting up your development environment, create a project in your IDE where you will develop your application.

Creating a Mobile Application

This section explains the general steps required for interfacing with Services Gatekeeper from an Android mobile application. Application requirements vary depending on the mobile operating system you are using and the functionality you are providing. See the documentation for your mobile operating system SDK for procedures and examples for developing mobile application elements such as the user interface and security.

The following steps show how to call a Services Gatekeeper API RESTful interface from an Android application. This example creates a RESTful POST method in the IDE mobile application project and sends the request to Services Gatekeeper. You interact with other Services Gatekeeper interfaces in a similar way.

To create a POST operation to Services Gatekeeper using the Android SDK and Eclipse IDE:

1. In the application project, create a JSON object using the **org.json.JSONObject** provided in the Android SDK.
2. Populate the fields of this JSON object using the Services Gatekeeper communication service (or plugin) resource WADL file. See *Communication Service Resource Guide* for details on the communication services.
3. Create an instance of the **org.apache.http.client.methods.HttpPost** object from the **android.jar** using the Services Gatekeeper RESTful service resource URL endpoint and the JSON object created in step 1 as the message parameters in the object.
4. Create an **org.apache.http.impl.client.DefaultHttpClient** object to send the object created in step 3.
5. Send the **HttpPost** object using the **DefaultHttpClient** object to the Services Gatekeeper RESTful interface. Services Gatekeeper provides a returned value as an object of **org.apache.http.HttpResponse**.
6. Extract the JSON string included in the **HttpResponse** object returned by Service Gatekeeper.
7. Create a JSON object from the extracted JSON string.
8. Extract the desired fields required by your application from this JSON object.

Testing a Mobile Application

You must test your mobile application ensuring that it functions correctly. Mobile OS SDKs include emulators that can be used for testing the application against Services

Gatekeeper. Alternatively, you can install the mobile application on a mobile device connected to your network for testing. See the mobile OS SDK documentation for more information about running your application for testing.

See "Understanding How to Test Applications" for information about testing your mobile application with Services Gatekeeper.

Distributing a Mobile Application

After completing sufficient testing of your application you distribute it to subscribers for use. Mobile OS vendors typically provide a store for application distribution. Alternative methods for distributing your application may also be available depending on mobile OS.

Understanding Management Structures

To help telecom operators organize their relationships with application providers, Services Gatekeeper uses a hierarchical system of accounts. Each application is assigned a unique application instance ID that is associated with an application account. Applications are assigned to service provider accounts. Each application account is associated with a service provider account. Application accounts with similar requirements are put into application groups, and service provider accounts with similar requirements are put into service provider groups. Each application group is associated with one application group service level agreement (SLA) and zero or more custom application group SLAs. Each service provider group is associated with one service provider group SLA and zero or more custom service provider group SLAs. These SLAs define and regulate the contractual agreements between the telecom operator and the application service provider. SLAs cover such things as which services the application may access and the maximum bandwidth available for use.

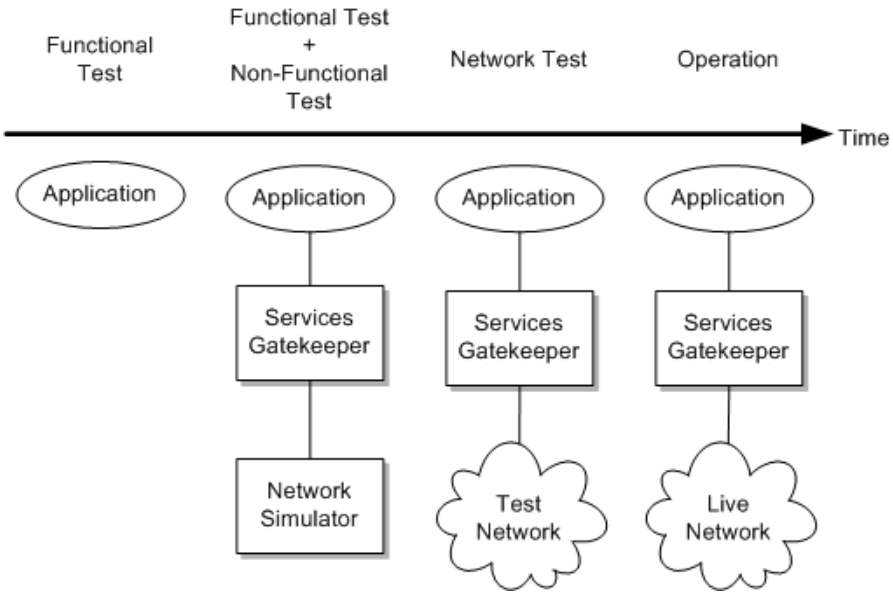
For more information about management structures, see *Services Gatekeeper Portal Developer's Guide*.

Understanding How to Test Applications

You test Applications in a telecom environment in stages. After basic functional issues are resolved, you can connect the application to an instance of Services Gatekeeper that is connected to the Platform Test Environment (PTE) network simulator for further testing. Next, the application is tested against a test network to eliminate any network related issues. Finally, the application can be placed into production on a live network.

Figure 1–1, "Application Testing Cycle" shows the application test flow, from the functional tests to deployment in a live network. Services Gatekeeper simulator-based tests can be performed in-house by an application service provider. However, the other tests require the cooperation of the target network operator.

Figure 1–1 Application Testing Cycle



See *Services Gatekeeper Platform Test Environment User's Guide* for details about testing how your application works with a network simulator.

2

Understanding the Supported Application Interfaces

This chapter introduces the interfaces that Oracle Communications Services Gatekeeper-hosted client applications can use. If you use HTTP-to-HTTP communication you can skip this chapter.

Overview of Services Gatekeeper Supported Interfaces

Services Gatekeeper enables you to provide client application developers with a choice of interface types, based on the needs of their applications. Services Gatekeeper provides these by default:

- SOAP-Based Interfaces for both traditional Web services and Oracle Service Bus environments
- RESTful Interfaces
- RESTful OneAPI Interfaces
- Native Interfaces

You can also createcustom interfaces, using the Network Service Supplier Portal.

SOAP-Based Interfaces

The SOAP-based Web services interfaces are based on the Parlay X standards and also include additional Extended Web services to cover functionality that is not supported by Parlay X. The SOAP-based interfaces include:

- Third Party Call (Parlay X 2.1 and 3.0; Part 2)

  Using the communication service based on this interface, an application can set up a call between a caller and a callee, poll for the status of the call, and end the call. Using the 3.0 communication, an application can set up a call among multiple participants and add, delete, or transfer those participants. The application can use the Audio Call communication service to play audio messages to one or multiple of the call participants set up using Third Party Call and, using notifications set up with Call Notification PX 3.0, can also collect digits in response to playing the audio message.

- Application-driven Quality of Service (QoS) (Parlay X 4.0; Part 17)

  Using the communication service based on this interface, an application can, in conjunction with a policy and charging rules function (PCRF) and policy and charging enforcement function (PCEF), apply, remove, modify, query, and receive

notifications for QoS feature profiles which can optimize various communication performance aspects.

- Audio Call (Parlay X 2.1 and 3.0; Part 11)

  Using the communication service based on this interface, an application can play audio to one or more call participants in an existing call session set up by the Parlay X 3.0 Third Party, find out if the audio is currently being played, and explicitly end playing the audio. It can also collect digits from a participant in response to an audio message, and with a notification set up using the Parlay X 3.0 Call communication service, can return that information to the application. It can also interrupt an ongoing interaction such as on-hold music.

- Call Notification (Parlay X 2.1 and 3.0)

  Using the communication service based on this interface, an application can set up and end notifications on call events. For example, an application can notify a caller that the callee's line is busy. The application can then reroute the call to another party. In addition, using the Parlay X 3.0 communication service, an application can interact with Parlay X 3.0 Audio Call to return digits collected from a call participant back to the application and to end calls. The Parlay X 3.0 version is deprecated.

- Device Capabilities (Parlay X 3.0; Part 18)

  Using the communication service based on this interface, an application can send a device's address (usually the telephone number) to an LDAP server and receive device capability information in return. The returned information can be either the device's equipment identifier (for example, an IMEI number), or the device capability information (the device's unique ID, device or model name, and a link to the User Agent Profile XML file).

- Short Messaging (Parlay X 2.1)

  Using the communication service based on this interface, an application can send SMS text messages, ringtones, or logos to one or multiple addresses, set up and receive notifications for final delivery receipts of those sent items, and arrange to receive SMS messages from the network that meet particular criteria.

- Multimedia Messaging (Parlay X 2.1; Part 5)

  Using this communication service based on this interface, an application can send multimedia messages to one or multiple addresses, set up and receive notifications for final delivery receipts of those sent items, and arrange to receive MMS messages from the network that meet particular criteria.

- Terminal Location (Parlay X 2.1; Part 9)

  Using the communication service based on this interface, an application can request the position of one or more terminals or the distance between a given position and a terminal. The application can also set up and receive notifications based on geographic location or time intervals.

- Terminal Status (Parlay X 2.1)

  Using the Terminal Status communication service, an application can:

  - Obtain the status (reachable, unreachable, or busy) of a single terminal or group of terminals as often as you specify, within a time period you specify.
  - Return the status of a terminal or group of terminals if the status changes. The terminal statuses are checked as frequently as you specify, for a time period you specify, and the status is returned if it changes.

- Presence (Parlay X 2.1) (Part 14)

  Using the communication service based on this interface, an application can act as either a *presentity* or as a *watcher* in a presence interaction. A presentity agrees to ensure data (called attributes) such as current activity, available. The data can be current activity, available means of communication, contact addresses, or other information. A watcher is a consumer of such data. As a watcher, an application can request to subscribe to all or a subset of a presentity's data, poll for that data, and start and end presence notifications. As a presentity, an application can publish presence data about itself, check whether any new watchers wish to subscribe to its presence data, authorize watchers to access presence data, block watchers from accessing presence data, and get a list of currently subscribed watchers.

- Payment (Parlay X 3.0; Part 6)

  Using the communication service based on this interface, an application can charge a user's account a specific amount, refund an amount, and split costs among multiple users. An application can also reserve an amount in an account, extend the amount associated with that reservation, make a charge against that reservation, or release the reservation.

- Application-driven Quality of Service (Extended Web Service)

  Using the communication service based on this interface, an application can, in conjunction with a PCRF and PCEF, apply, remove, modify and query QoS templates which can optimize various communication performance aspects.

- Binary Short Messaging (Extended Web Services)

  Using the communication service based on this interface, an application can send and receive generic binary objects (for example, a vCard) using SMS mechanisms, and set up and receive notifications. This interface is not based on the Parlay X standards, but instead belongs to the Services Gatekeeper Extended Web Services (EWS) set.

- WAP Push (Extended Web Services)

  Using the communication service based on this interface, an application can send a WAP Push message, send a replacement WAP Push message, or set up status notifications about previously sent messages. The application-facing interface of this communication service is not based on the Parlay X 2.1 specification. Many elements within it, however, are based on widely distributed standards.

- Subscriber Profile (Extended Web Services)

  Using the communication service based on this interface, an application can retrieve particular information or an entire profile (subject to internal filtering) for a subscriber from an LDAP server attached to the network. The application-facing interface of this communication service is based on a subset of that in a proposed Parlay X version.

- Session Manager (Extended Web Services)

  Using the communication service based on this interface, an application can establish a Services Gatekeeper session. Whether sessions are used is up to the operator.

For details, see "Creating Applications Using the SOAP Interfaces."

RESTful Interfaces

The RESTful interfaces provide access to functionality similar to the SOAP Facade. The RESTful interfaces include:

- Audio Call

  Using the communication service based on this interface, an application can play an audio file to one or more call participants in a call session. It is also possible to collect digits from a participant in response to the audio, and return that participant to the application.

- Call Notification

  Using the communication service based on this interface, an application can set up and remove call notifications. Call notifications notify the application about the particular state of the call, such as busy, unreachable, and so on. Call direction notifications query the application for information about how to handle a call that is in a particular state.

- Device Capabilities and Configuration

  Using the communication service based on this interface, an application can send a device's address (usually the telephone number) to an LDAP server and receive device capability information in return. The returned information can be either the device's equipment identifier (for example, an IMEI number), or device capability information (the device's unique ID, device or model name, and a link to the User Agent Profile XML file).

- Email

  Using the communication service based on this interface, an application can send and receive emails. This communication service uses MMS in the northbound interface and a plug-in that enables the sending of email through SMTP and receiving email through POP3 and IMAP protocols. See the discussion about "Parlay X 2.1 Multimedia Messaging/SMTP, POP3, and IMAP" in *Services Gatekeeper Communication Service Reference Guide*.

- Multimedia Messaging

  Using the communication service based on this interface, an application can send MMS messages, fetch MMS messages, and fetch information on MMS messages that have been received and stored on Services Gatekeeper. The application can also get delivery status on sent messages, and start and stop a notification.

- Payment

  Using the communication service based on this interface, an application can charge an amount to a user's account using Diameter, refund amounts to that account, and split charge amounts among multiple users. An application can also reserve amounts, reserve additional amounts, charge against the reservation, or release the reservation.

- Presence

  Using the communication service based on this interface, an application can act as either a presentity or a watcher in a presence interaction.

  A presentity agrees to ensure that data (called attributes) is available to others. The data can be current activity, available means of communication, contact addresses, or other information. As a presentity, an application can publish presence data about itself, check whether any new watchers wish to subscribe to its presence

data, authorize watchers it chooses to access the data, block watchers from accessing the data, and get a list of currently subscribed watchers.

A watcher is a consumer of such data. As a watcher, an application can request to subscribe to all or a subset of a presentity's data, poll for that data, and start and end presence notifications.

- Short Messaging

  Using the communication service based on this interface, an application can send an SMS message, a ringtone, or a logo, and fetch SMS messages and delivery status reports that have been received and stored on Services Gatekeeper, start and stop a notification.

- Terminal Location

  Using the communication service based on this interface, an application can get a location for an individual terminal or a group of terminals; get the distance of the terminal from a specific location, and start and stop notifications based on geographic location or on a periodic interval.

- Terminal Status (Parlay X 2.1)

  Using the communication service based on this interface, an application can request the status (such as reachable, unreachable, or busy) of a single terminal; request the status of a group of terminals; or request to be notified if a terminal status changes within a specified time period.

- Third Party Call

  Using the communication service based on this interface, an application can set up a call, get information on that call, cancel the call request before it is successfully completed, or end a call that has been successfully set up.

- WAP Push

  Using the communication service based on this interface, an application can send a WAP Push message and receive status notifications about that message.

- Session Manager

  Using the communication service based on this interface, an application can establish a session with the Services Gatekeeper operator. Operators can choose whether to use Services Gatekeeper in session-based mode.

- Subscriber Profile

  Using the communication service based on this interface, an application can query an operator's database for the attributes of an individual subscriber profile (for example, the terminal type), or for entire subscriber profiles.

For details, see "Creating Applications Using the RESTful Interfaces."

RESTful OneAPI Interfaces

The OneAPI interfaces provide access to RESTful OneAPI operation. The interfaces include:

- Short Messaging

  Using the communication service based on this interface, an application can send SMS messages, fetch SMS messages, deliver status reports, and to start and stop notifications.

- Multimedia Messaging

Using the communication service based on this interface, an application can send MMS messages, fetch MMS messages and delivery status reports, and to start and stop notifications.

- Terminal Location

  Using the communication service based on this interface, an application can get a location for an individual terminal or a group of terminals.

- Payment

  Using the communication service based on this interface, an application can charge an amount to a user's account using Diameter and to refund amounts to that account. Applications can also reserve amounts, reserve additional amounts, charge against the reservation or release the reservation.

For details, see "Creating Applications Using the OneAPI RESTful Interfaces."

Native Interfaces

The Native interfaces provide access to native telecom-specific protocols. The interfaces include:

- Native MM7

  Using the communication service based on this interface, an application can send and receive MMS messages and receive status notifications about previously sent messages. This application-facing interface is based on the 3GPP MM7 standard.

- Native SMPP

  Using the communication service based on this interface, an application can send and receive SMS messages and receive status notifications about previously sent messages. This application-facing interface is based on the SMS Forum standard.

- Native UCP

  Using the communication service based on this interface, an application can establish a session with application clients and the network, submit and receive SMS messages, and receive status notifications about previously sent messages. This application-facing interface is based on the Short Message Service Center EMI-UCP Interface 5.1 specification. It exposes Universal Computer Protocol (UCP) protocol to applications and uses UCP to connect to a Short Message Service Center (SMSC).

For details, see "About the Supported SOAP Native Facade."

3

Managing Communication Sessions

This chapter explains how to use the Oracle Communications Services Gatekeeper Session Manager Web service to manage the sessions that your applications use to communicate with Services Gatekeeper.

Understanding the Session Manager Web Service

The Session Manager Web service contains operations for managing a session with Services Gatekeeper, including establishing the session, changing the application's password, querying the amount of time remaining in the session, refreshing the session, and terminating the session.

> **Note:** Not all installations of Services Gatekeeper require session management. The contents of this chapter apply only to those installations that do.

When an operator requires it, an application must establish a session with Services Gatekeeper before the application can perform any operations using the Parlay X or Extended Web Services interfaces. When a session is established, a session ID is returned, which must be used in each subsequent operation toward Services Gatekeeper.

Session Management for SOAP, RESTful, and OneAPI Interfaces

In order to interact with Services Gatekeeper, applications use SOAP-based, RESTful, OneAPI, or SOAP native interfaces. Those applications using SOAP-based interfaces must manipulate the SOAP messages that they use to make requests in certain specialized ways. They must add specific information to the SOAP header, and, if they are using Multimedia Messaging, they must send their message payload as a SOAP attachment. Applications using the native interfaces use the normal, native interface mechanisms, which are not covered in this document.

How developers program applications to manipulate SOAP messages depend on the environment in which the application is being developed.

> **Note:** Clients created using Axis 1.2 or older do not work with some communication services. Developers should use Axis 1.4 or newer if they wish to use Axis.

For examples of using the Oracle WebLogic Server environment manipulate SOAP messages, see "Managing SOAP Headers and Attachments Programmatically".

You can configure whether SOAP, REST, or OneAPI-based applications must provide credentials and apply for a session ID before they can communicate with Services Gatekeeper. The default setting requires that these types of applications establish a session using the Session Manager Web service before they are allowed to run traffic through Services Gatekeeper. You can also make a session optional, or simply remove session checking completely.

The session requirement is useful as a security mechanism because it requires all applications to authorize themselves, and it allows Services Gatekeeper to keep track of all traffic for a session.

About Sessions

An application establishes a session in Services Gatekeeper by invoking the **getSession()** operation in the Session Manager Web service. **getSession()** is the only operation that does not require a session ID to be invoked. This operation returns a string representing the session ID to the client, and Services Gatekeeper establishes a session identified by the ID string. See "Operation: getSession" for details about this operation.

By default, the session is left open until the an application closes it. You can also set a time limit for sessions using the **validityTime** field in **ApplicationSessionMBean**. See "Operation: getSessionRemainingLifeTime" for details.

Each session is valid for the entire Services Gatekeeper domain, across clusters, and covers all communication services to which the application has contractual access. Once established, the session ID must appear in the `wlng:Session` element in the header of every subsequent SOAP request.

Example 3–1 Example of a Session Header element

```
<Session>
  <SessionId>app:-2810834922008400383</SessionId>
</Session>
```

You can configure the session mode that determines whether session IDs are required. The mode has these possible values:

- **Required** - The default value. Requires that all applications authorize themselves with credentials before requesting a session ID. Services Gatekeeper validates session IDs and rejects communication attempts if the IDs are invalid. If the mode is **Required**, a session ID is required for all communication through Services Gatekeeper.
- **Disabled** (**sessionless**) - Services Gatekeeper does not check whether a session ID exists. If applications successfully authenticate themselves, they receive a session ID string of `sessionless`, which is used in all communication within the session. If they do not authenticate, no session ID is provided or required. In this case the application uses whichever Web services Security (WS-Security) mechanism is required by the Services Gatekeeper operator for security.
- **Optional** - Services Gatekeeper does not require that an application log on or request a session ID. If the application successfully authenticates, it is provided with a session ID that is checked for validity. If found invalid, the request is rejected. If the application passes in a header with a session ID of `sessionless`, or if no session ID is passed in, the request is accepted.

Changing the Session Mode

To change the Services Gatekeeper session mode:

1. Start the MBean browser that you use to configure Services Gatekeeper.

   You can browse MBeans by using JConsole or PTE, which are supplied with Services Gatekeeper.

2. Navigate to **wlng**, then **AccountService**, then **ApplicationSessionMBean**, and then **SessionRequired**.

3. Check the box representing the session behavior that your implementation requires:

   - **Required**
   - **Disabled**
   - **Optional**

   See "About Sessions" for details on the different options.

Session Manager WSDL File

The WSDL file for the Session Manager Web service can be found here:

`http://host:port/session_manager/SessionManager`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

SessionManager WSDL Interface Reference

The Session Manager Web service interface includes these operations:

- Operation: getSession
- Operation: changeApplicationPassword
- Operation: getSessionRemainingLifeTime
- Operation: refreshSession
- Operation: destroySession

Operation: getSession

Establishes a session using WS-Security. Authentication information must be provided according to WS-Security. See "Understanding SOAP-Based Authentication" for more information.

Input message: getSession

Table 3–1 Input Message: getSession

Part name	Part type	Optional	Description
N/A	N/A	N/A	N/A

Output message: getSessionResponse

Table 3–2 Output Message: getSessionResponse

Part name	Part type	Optional	Description
getSessionReturn	xsd:String	N	The session ID to use in subsequent requests.

Referenced faults

GeneralException

Operation: changeApplicationPassword

Changes the password for an application.

Input message: changeApplicationPassword

Table 3–3 Input Message: changeApplicationPassword

Part name	Part type	Optional	Description
sessionId	xsd:string	N	The ID of an established session.
oldPassword	xsd:string	N	The current password.
newPassword	xsd:string	N	The new password.

Output message: changeApplicationPasswordResponse

Table 3–4 Output Message: changeApplicationPasswordResponse

Part name	Part type	Optional	Description
N/A	N/A	N/A	N/A

Referenced faults

None

Operation: getSessionRemainingLifeTime

Returns the remaining lifetime of an established session in minutes. This operation works with the **validityTime** field in the **ApplicationSessionMBean**. The default value for **validityTime** is 0 minutes, which keeps the session open indefinitely unless you specially destroy it. You can change the default to set a time limit for all sessions. When the time limit expires, the session is automatically destroyed.

If **validityTime** is set to the default (**0** minutes), the session is always valid unless you destroy it. **getSessionRemainingLifeTime** returns these example values if you use the default **validityTime** value:

- Immediately after the session is created, this operation returns **0**.
- 1 minute after the session is created, this operation returns **-1**.
- 3 minutes after the session is created, this operation returns **-3**.

If you change the **validityTime** value to a positive number of minutes, this operation destroys the session at the end of that time period. For example if you set **validityTime** to **5** minutes, you get this behavior:

- Immediately the session is created, this operation returns **5**.

- 1 minute after the session is created, this operation returns **4**.
- 3 minutes after the session is created, this operation returns **2**.

  5 minutes after the session is created, it is invalidated and destroyed.

See the "All Classes" section of the *Services Gatekeeper OAM Java API Reference* for details on **ApplicationSessionMBean**.

Input message: getSessionRemainingLifeTime

Table 3–5 Input Message: getSessionRemainingLifeTime

Part name	Part type	Optional	Description
sessionId	xsd:string	N	The ID of an established session.

Output message: getSessionRemainingLifeTimeResponse

Table 3–6 Output Message: getSessionRemainingLifeTimeResponse

Part name	Part type	Optional	Description
getSessionRemainingLifeTimeReturn	xsd:string	N	The remaining lifetime of the session. Given in milliseconds.

Referenced faults

None

Operation: refreshSession

Refreshes the lifetime of a session. The session can be refreshed during a time interval after the session has expired. This time interval is configured in Services Gatekeeper.

Input message: refreshSession

Table 3–7 Input Message: refreshSession

Part name	Part type	Optional	Description
sessionId	xsd:string	N	The ID of an established session.

Output message: refreshSessionResponse

Table 3–8 Output Message: refreshSessionResponse

Part name	Part type	Optional	Description
refreshSessionReturn	xsd:string	N	The session ID to be used in subsequent requests. The same ID as the original session ID is returned.

Referenced faults

None

Operation: destroySession

Destroys an established session.

Input message: destroySession

Table 3–9 Input Message: destroySession

Part name	Part type	Optional	Description
sessionId	xsd:string	N	The ID of an established session.

Output message: destroySessionResponse

Table 3–10 Output Message: destroySessionResponse

Part name	Part type	Optional	Description
destroySessionReturn	xsd:boolean	N	True if the session was destroyed.

Referenced faults

None

Session Manager Examples

Example 3–2 illustrates how to get the Session Manager Web service and how to prepare the generated stub with WS-Security information. The stub is generated from the Session Manager Web service.

Example 3–2 Get the Session Manager Web Service

```
protected ClientSessionManImpl(String sessionManagerURL, PolicyBase pbase) throws Exception {
  SessionManagerService accessservice =
    new SessionManagerService_Impl(sessionManagerURL+"?WSDL");
  port = accessservice.getSessionManager();
  pbase.prepareStub((Stub)port);
  }
```

Example 3–3 illustrates how to prepare the Session Manager Web service stub with username token information according to WS-Policy.

Example 3–3 Prepare the Session Manager with Username Token information

```
package com.bea.wlcp.wlng.client.access.wspolicy;
import weblogic.wsee.security.unt.ClientUNTCredentialProvider;
import weblogic.xml.crypto.wss.WSSecurityContext;
import javax.xml.rpc.Stub;
import java.util.ArrayList;
import java.util.List;
public class UsernameTokenPolicy implements PolicyBase {

  private String username;
  private String password;

public UsernameTokenPolicy(String username, String password) {
    this.username = username;
    this.password = password;
  }

  public void prepareStub(Stub stub) throws Exception {
    List<ClientUNTCredentialProvider> credProviders = new ArrayList<ClientUNTCredentialProvider>();
    credProviders.add(new ClientUNTCredentialProvider(username.getBytes(),
                                                      password.getBytes()));
    System.out.println("setting standard wssec");
```

```
    stub._setProperty(WSSecurityContext.CREDENTIAL_PROVIDER_LIST,
                      credProviders);
  }

}
```

Part II

Creating Applications Using the RESTful Interfaces

Part II describes how to use the interfaces in the RESTful facade to create applications that interact with Oracle Communications Services Gatekeeper.

Part II contains the following chapters:

4

Using the RESTful Interfaces

This chapter presents an overview of Oracle Communications Services Gatekeeper RESTful interfaces, and explains how to use them to create applications that interact with Services Gatekeeper.

Supported RESTful Interfaces

The RESTful interfaces provide applications with the operations they use to interact with Services Gatekeeper.

See *Understanding the Supported Application Interfaces* for a complete list and short description of the RESTful interfaces supported by Services Gatekeeper. These interfaces are explained in detail in the chapters that follow.

Understanding RESTful Operations

The following basic elements are present in the requests that an application makes to the RESTful interfaces and the responses it receives from the interface:

- Request-URI and HTTP Methods in requests
- Status-Line in responses
- Headers

  Sometimes messages (for example, multimedia messages) contain attachments. Special headers are provided to specify the attachment details when a message has multiple parts. See "Headers for Multipart Messages with Attachments".
- Message Body

Request-URI and HTTP Methods

Applications use one of four methods, "GET", "POST", "PUT", or "DELETE", to request a required action to be performed on an abstract or physical resource. The resource has a specific Uniform Resource Identifier (URI). The Request-URI identifies the abstract or physical resource that an HTTP method acts upon or uses and is therefore the most important part of any request that an application makes to the RESTful interfaces.

Here is the GET method used to query for the status of a terminal:

```
GET /rest/terminal_status/status?query="%7B%22address%22%3A%22tel%3A123%22%7D"
HTTP/1.1
```

where, the string `%7B%22address%22%3A%22telA123%22%7D` is the address of the terminal (or the `{"address":"tel:123"}` JavaScript Object Notation (JSON) object).

General Format of a Request-URI

A fully qualified Request-URI consists of a sequence of concatenated sections that are each separated by a forward slash. For example:

`https://host:port/rest_facade_context_root/URI/path_info_param/query-string`

where,

- *host*:*port*: The hostname or IP address and port of your Services Gatekeeper installation; for example, `127.0.0.1` and `8001`.
- *rest_facade_context_root*: The location of the set of resources that the particular interface uses; for example, `rest`, in the following example query for the status of a terminal:

  ```
  GET /rest/terminal_status/status?query="%7B%22address%22%3A%22tel%3A123%22%7D"
  HTTP/1.1
  ```

  In Services Gatekeeper, the *rest_facade_context_root* entry is always **rest**.
- *URI*: The location of a specific kind of functionality within the interface; for example, `terminal_status`.
- *path_info_param*: An identifier of a specific resource, for example, `calls`. This is seen in the Make Call request, `POST /rest/third_party_call/calls HTTP/1.1`. See Example 4–8. The path-info-param entry does not occur in all URIs.
- *query_string*: A set of name-value pairs that describes what is being requested; for example, `status?query="%7B%22address%22%3A%22tel%3A123%22%7D`, in the above GET query for *rest_facade_context_root*. The query-string entry is not seen in every URI.

POST

The POST method accesses a resource factory to create a resource that does not yet have a URI. Multiple requests to a resource factory can create multiple new resources.

The following example statement will set up a call between two parties:

Example 4–1 POST Statement

```
POST /rest/third_party_call/calls HTTP/1.1
```

For the POST method:

- The URI in the request represents the factory resource accessed to create a resource. In Example 4–1, **/rest/third_party_call/calls** is the factory resource accessed to create a resource.
- The request body contains the information required to create the resource. See Example 4–6.
- If the resource is created, the response body will contain the identifier for the new resource. See Example 4–7. If the operation fails, the response body will contain the error response.

PUT

The PUT method creates a resource that has a predetermined URI. This method can be used to update a resource (or to start a stateful process). For example, an application uses the following statement to start notifications on a specific terminal:

Example 4–2 PUT Statement

```
PUT /rest/terminal_location/periodic_notification HTTP/1.1
```

For the PUT method:

- The URI in the request represents the resource to update or the resource for which to start a stateful process. In Example 4–2, **/rest/terminal_location/periodic_notification** represents the resource accessed to start periodic notifications on a terminal's location.
- The request body will contain the required information. (The JSON object will contain, for example, information on the terminal, the criteria to monitor, the frequency and duration of the monitoring, where to place the notification, and the correlator to identify the session.
- The Location header in the response will specify the location that contains the resulting notifications. For example, the application may see the following header for the above PUT request:

  ```
  Location://terminalloc_host:port/rest/terminal/notifications
  ```

  In order to complete the operation, the application must access the specified location and use the correlator to retrieve the notifications.
- If the operation fails, the response body will contain the error response.

GET

The GET method retrieves the state of a specific resource that has been previously set up. The specific resource is identified in the query string as shown in Example 4–3, where an application attempts to retrieve the status of a terminal whose address is specified as `"tel:123"`.

Example 4–3 GET Statement

```
GET /rest/terminal_status/status?query=%7B%22address%22%3A%22tel%3A456%22%7D
HTTP/1.1
```

For the GET method:

- The URI in the request represents the query string that uniquely identifies the resource whose status the application wishes to retrieve. In Example 4–3, the value for **status?query** is (`{"address":"tel:456"}`, the unique address of the terminal in JSON representation).
- The request body will be empty.
- The response will provide information on the state of the resource.

  In order to complete the operation, the application must access the specified location and use the correlator to retrieve the notification.
- If the operation fails, the response body will contain the error response.

DELETE

The DELETE method removes a specified resource. The application provides the correlator or the identifier for the resource that must be removed in the Request-UR, as shown here:

Example 4–4 DELETE Statement

```
DELETE /rest/terminal_status/notifications/6789 HTTP/1.1
```

For the DELETE method:

- The URI in the request contains the correlator, which is a value that uniquely identifies the resource the application wishes to remove. In Example 4–4, `6789` is the value which the application provided as the correlator when it requested notifications on a terminal's status.
- The request body will be empty.
- The response body will be empty.
- If the operation fails, the response body will contain the error response.

Status-Line

The Status-Line is the first line in any response that an application receives when it interacts with a RESTful interface in Services Gatekeeper. The Status-Line has this syntax:

`HTTP/1.1` *`Status-Code Reason-Phrase`*

where,

- *Status-Code*: A three-digit indicator of the success or failure to fulfill the request.
- *Reason-Phrase*: A brief description of the (successful) action performed, or the reason for the failure.

For example:

```
HTTP/1.1 201 Created
```

Table 4–1 lists some of the status codes and reason-phrases commonly encountered when interacting with the Services Gatekeeper RESTful interfaces:

Table 4–1 A Sampling of Status Codes and Reason Phrases

Status-Code	Reason-Phrase	Description
200	OK	Success. The request has succeeded. The information returned with the response is dependent on the method used in the request.
201	Created	Success. The requested resource was created.
204	No Content	Success. The server has fulfilled the request but does not need to return an entity-body.
501	Internal Server Error	Indicates failure. An unexpected condition prevented the server from fulfilling the request.

For a complete listing of the HTTP status codes and their definitions, see RFC 2616 at:

`http://www.ietf.org/rfc/rfc2616.txt`

Headers

The requests and responses for RESTful operations include the following header fields:

- Authorization: The Authorization header field is required and is found in all requests. It indicates the type of authentication and security. For example:

  ```
  Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
  ```

  For more information, see "RESTful Authentication and Security".

- Session ID: When Services Gatekeeper is running in Session mode, the X-Session-ID header must be present in all request messages to identify the application.

  In session mode, an application's first task is to obtain a session ID from the Session Manager Web service. All traffic requests (for that session) include this identifier in the key-value pair for the X-Session-ID key. For example:

  ```
  X-Session-ID: app: -1780934689905632396
  ```

  The X-Session-ID header is not present when Services Gatekeeper is running in Sessionless mode. For more information on sessions, see "Adding RESTful Session Manager Support"

- Service correlation ID (X-SCID): The X-SCID header will be present if the application wishes to set up service correlation. This is a key-value pair of the format key=X-SCID. For more information on service correlation, see "Service Correlation".

- Tunneled parameters: Tunneled parameters (also called xparams) are present if the application wishes to supply parameters that are not supported in the RESTful interface itself and need to be passed on to the network. The key-value pairs are:

  – X-Param-Key and X-Param-Values: The X-Param-Key and X-Param-Values headers are found in the requests.

  – X-Plugin-Param-Key and X-Plugin-Param-Values: The X-Plugin-Param-Key and X-Plugin-Param-Values headers are returned in the response headers.

  See *Services Gatekeeper Communication Service Reference Guide* for descriptions of the tunneled parameters that are applicable to your communication service.

- Location: Location headers are found in responses to certain requests. They are used to identify a new resource or to redirect the recipient to a location other than the Request-URI for completion of the request.

  For example, the call identifier for a newly-setup call is returned in the Location header as:

  ```
  Location: http://local:host:8001/rest/third_party_
  call/call/app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling_
  host|-50d94925ab34bf0
  ```

  Below, the Location header specifies the location that the application must access to receive notifications about a terminal (for which the application had previously initiated a notification request).

  ```
  Location: http://notificationloc_host:port/rest/terminal_location/notifications
  ```

- Content-Length: The length of the request or response body.

- Content-Type: The MIME-type value for the Content-Type header field can be **multipart/form-data** or **application/json**. The **multipart/form-data** value for the Content-Type header field is described in the next section.

Headers for Multipart Messages with Attachments

The RESTful interfaces for Multimedia Messaging and WAP Push use HTTP attachments to transport their content. Both interfaces support multipart/form-data POST requests. When you use RESTful interfaces with Services Gatekeeper, multiple attachments are supported in both application-initiated and network-triggered messages.

When a request message contains one or more messages embedded within it, a specified boundary is placed between the parts of the message and at the beginning and end of the message. For multipart message requests:

- The MIME-type value for the Content-Type header field can be **multipart/form-data** or **application/json**. If the MIME-type value for the Content-Type header field is **multipart/form-data**, the **boundary** entry is used to provide a value for the boundary between the message parts.
- Each message part contains the following:
  - Content-Disposition header field that has a value of `form-data` and a name attribute with the appropriate value. For example, the message part name is `messagePart`.
  - Content-Type header field that has a value of `application/json` and the `charset` attribute with the appropriate value.
  - Content-Transfer-Encoding field with the appropriate value.
- If the content of the message is pure ASCII, the response body contains the message. Otherwise the response body contains an identifier that is used to fetch the actual message.

Example 4–5 Example of a Multipart Message Request

```
POST /rest/multimedia_messaging/messages HTTP/1.1
X-Session-ID: app: -123456789012346789
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
Content-Length: 1215
Content-Type: multipart/form-data; boundary=kboiiFPAakDPYKeY7hBAW9I5c0rT48

--kboiiFPAakDPYKeY7hBAW9I5c0rT48
Content-Disposition: form-data; name="messagePart"
Content-Type: application/json; charset=US-ASCII
Content-Transfer-Encoding: 8bit

{
  "addresses":["tel:8765"],
  "subject":"Hello World",
  "priority":null,
  "senderAddress":"tel:1234",
  "charging":null
  "receiptRequest":
         {
           "correlator":"981234",
           "endpoint":"http://endpt_host:port/jaxws/MessageNotification",
           "interfaceName":"interfaceName",
         }
 }
```

```
--kboiiFPAakDPYKeY7hBAW9I5c0rT48
Content-Disposition: form-data; name="Attachment-txt-1"
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 8bit

This sentence represents the attachment text.
--kboiiFPAakDPYKeY7hBAW9I5c0rT48
```

Service Correlation

In some cases the service that an application provides to its end-users may involve accessing multiple Services Gatekeeper communication services.

For example, a mobile user might send an SMS to an application asking for a pizza restaurant nearest to his current location. The application then makes a Terminal Location request to find the user's current location, looks up the address of the closest pizza restaurant, and then sends the user an MMS with all the appropriate information. Three Services Gatekeeper communication services are involved in executing what, for the application, is a single service.

Services Gatekeeper uses a service correlation ID to correlate the three communication service requests. The service correlation ID (SCID) is a string that is captured in all the charging data records (CDRs) and event data records (EDRs) generated by Services Gatekeeper. The CDRs and EDRs can then be orchestrated in order to provide special treatment for a given chain of service invocations, by, for example, applying charging to the chain as a whole rather than to the individual invocations.

How Service Correlation IDs are Provided Services Gatekeeper does not provide the service correlation ID. The type of request determines the service correlation ID:

- Application-Initiated Requests: When the chain of services is initiated by an application-initiated request, the application must provide and ensure the uniqueness of the SCID within the chain of service invocations.

  In certain circumstances, it is also possible for a custom service correlation service to supply the SCID, in which case it is the custom service's responsibility to ensure the uniqueness of the SCID.

- Network-Triggered Requests: When the chain of services is initiated by a network-triggered request, Services Gatekeeper calls an external interface to get the SCID.

  This interface must be implemented by an external system. Integration of such an external interface must be a part of a system integration project. It is the responsibility of the external system to provide and ensure the uniqueness of the SCID.

Message Body

The message body for a request or response is present only when required. The message body is a JSON object.

Request Body

When present, the request body provides additional data required to complete the specific request. The following request body for an example Make Call operation provides the addresses of the called and calling parties and any charges to apply for the call:

Example 4–6 Request Body for Make Call

```
{"calledParty":"sip:ann@sipcalled_host:port",
 "charging":null,
 "callingParty":"sip:zach@sipcalling_host:port"
}
```

Response Body

When present, the response body provides data that the application will need for later action. The following response body for the Make Call operation provides the application with the identifier for the call that was set up. The application will use this identifier to end the call, when necessary.

Example 4–7 Response Body for a Make Call Operation

```
{"result":"app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling_
host|-50d94925ab34bf0"}
```

Example of a Request and Response

Example 4–8 shows an application's request to set up a call between two parties using the Make Call operation in the Service Gatekeeper RESTful interface.

Example 4–8 Request associated with a Make Call Operation

```
POST /rest/third_party_call/calls HTTP/1.1
X-Session-ID: app: -1780934689905632396
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
Content-Length: 105
Content-Type: application/json

{"calledParty":"sip:alice@sipcalled_host:port",
 "charging":
      {
       "description":"init_call",
       "amount":"11",
       "code":"1111",
       "currency":"rmb"
      },
 "callingParty":"sip:bob@sipcalling_host:port"
}
```

Example 4–9 shows the response which the application receives for a successful setup of the requested call.

Example 4–9 Response associated with a Make Call Operation

```
HTTP/1.1 201 Created
Date: Wed, 20 Oct 2010 06:58:06 GMT
Location: http://local:host:8001/rest/third_party_
call/call/app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling_
host|-50d94925ab34bf0
Content-Length: 96
Content-Type: application/json
X-Plugin-Param-Keys:
```

```
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1

{"result":"app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling_
host|-50d94925ab34bf0"}
```

RESTful Authentication and Security

The RESTful interfaces use HTTP basic authentication, using username/password. SSL is required. For instance:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

For more information on HTTP basic authentication, see RFC 2617 at

http://www.ietf.org/rfc/rfc2617.txt

RESTful Notifications and Publish/Subscribe

When an application needs to receive a notification, (about a message delivery receipt for example), the application uses the publish/subscribe functionality in Services Gatekeeper.

An application can subscribe only to its own notifications (that is, to the notifications associated with its start notification requests). Any attempt to subscribe to notifications for other applications will be rejected.

Supported Endpoint Addresses

The application provides an endpoint address that resides on a publish/subscribe server. You can specify one of the following endpoint addresses:

- RESTful: a Bayeux protocol channel name
- SOAP: a Web service implemented by the application

A SOAP endpoint for the notification of a message sent using RESTful interfaces in Services Gatekeeper is valid only if the SOAP and RESTful interfaces reside in the same cluster.

Endpoint Addresses for RESTful Interfaces

The RESTful interfaces in Services Gatekeeper rely on the publish/subscribe model supported by the Publish-Subscribe Server functionality of Oracle WebLogic Server 10.3.1.

For more information on the publish/subscribe model, please see the discussion on "Using the HTTP Publish-Subscribe Server" in *Oracle Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Using the Bayeux (Cometd) Protocol to Communicate with the Server

When using RESTful interfaces, the application client must use the Bayeux protocol to communicate with the Web server. In this model, clients subscribe to a channel (similar to a topic in JMS) and receive messages (notifications) as they become available. The endpoint address resides on a Bayeux server.

Understanding Bayeux Connections and Subscriptions

The mechanisms for connecting to the Web server and subscribing to a channel are covered by the Bayeux protocol itself. The Bayeux client manages connections to the server and subscriptions to a channel. If the channel does not exist when the client subscribes to it, the channel is created.

Once the Bayeux client connects to the server and subscribes to a channel, the RESTful client can start sending notifications. It does so by providing the Bayeux protocol channel name as the endpoint entry in a start notification request.

Understanding the Bayeux Protocol Channel Name

The Bayeux protocol channel name begins with **/bayeux/***appInstanceID* where *appInstanceID* is the client application's application instance account ID. In Example 4–10, the *appInstanceID* is `domain-user`.

An application places the endpoint address for delivery notifications in the body of the request message. In the following example, it is inside a **reference** object.

Example 4–10 Example of an Endpoint Address in a Reference Object

```
...
 "reference":
        {"interfaceName":"interfaceName",
         "correlator":"6789",
         "endpoint":"/bayeux/domain-user/ts"
        }
...
```

For more information on application instances, see the discussion of application instances in *Services Gatekeeper Portal Developer's Guide*.

Using the Data at the Endpoint Address

Services Gatekeeper delivers notifications to the Bayeux channel name provided by the application in the associated request. It is the client's responsibility to interact with the publish/subscribe server to access the messages/data placed at the **endpoint** address. The mechanisms to do so are outside the scope of the Services Gatekeeper RESTful facades.

Understanding RESTful Errors and Exceptions

In the case of an error, the Status-Line in the response message indicates the protocol version, the three-digit status code, and the reason for the request failure.

Service exception and policy exception objects are represented in the response body as JSON with the following form:

```
{"error":
           {
           "type":"class name of the error object"
           "message":"error message"
           }
  }
```

For service exceptions, the value for **type** is:

```
"type":"org.csapi.schema.parlayx.common.v2_1.ServiceException"
```

For policy exceptions, the value for **type** is:

```
"type":"org.csapi.schema.parlayx.common.v2_1.PolicyException"
```

For example, when an MMS message sent by an application cannot be delivered to the multimedia messaging service (MMSC), the response from the MMSC contains the statusCode and statusText. Services Gatekeeper returns these values to the application in the **requestError** object. The **requestError** object contains the **SVC0001** serviceException with the error code **MMS-000005**.

The format for the error code is

```
MMS-000005:<StatusCode from MMSC>:<StatusText from MMSC>
```

The requestError object is:

```
{"requestError":
       {
       "serviceException":
            {
             "messageId":"SVC0001",
             "text":"A service error occurred. Error code is %1",
             "variables":["MMS-000005:3002:Message rejected"]
            }
       }
}
```

A variable substitution is performed for PX exceptions in error messages.

5

Adding RESTful Third Party Call Support

This chapter describes the operations in the Third Party Call interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the RESTful Third Party Call Interface

Applications use the RESTful Third Party Call interface in Services Gatekeeper to set up a call, get information on that call, cancel the call request before it is successfully completed, or end a call that has been successfully set up.

Additionally, applications use this interface to specify the data required for the billing operation associated with the call.

REST Service Descriptions Available at Runtime

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at:

`http://host:port/rest/third_party_call/index.html`

Where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

RESTful Third Party Call Interface Reference

The RESTful Third Party Call interface includes these operations:

- Make Call
- Get Call Information
- Cancel Call
- End Call

Make Call

To set up a call between two parties (referred to as the calling party and the called party), provide the URI of the calling party and the called party in the body of the request for the call. Optionally, the request can also indicate any cost-charging parameters to be applied to the call.

If the call setup is successful, the response header will contain the URI of the newly created resource as the value of the Location header field. Additionally, the response body will contain the call identifier for the newly created call object. Use this call identifier to reference the call later.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/third_party_call/calls`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the POST request accepts the following parameters:

- **calledParty**: String. Required. The address (URI) of the party to whom the call is made. Specified as `sip:user_name@destination_ip:destination_port`.
- **callingParty**: String. Required. The address (URI) of the party making the call. Specified as `sip:user_name@origin_ip:origin_port`.
- **charging**: a JSON object. Optional. This object defines the cost charging parameters for the call. A call with no charging parameters can be entered as `"charging": null`.

  If a charge is to be applied, provide values for the following in the **charging** object:

  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**: Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
```

```
  "calledParty": "URI",
  "callingParty": "URI",
  "charging": {
    "description": "String",
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  }
}
```

Response Header

The value in the Location header field is a single absolute URI in the following format:

`http://host:port/rest/third_party_call/{${result}`

where *result* is a string-formatted call identifier for the newly-created call. See Example 5–2.

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body for the POST request contains the call identifier returned in the Location header field. The call identifier is the value for the `result` term in the response body represented by the following name/value pair structure:

```
{"result": "String"}
```

See Example 5–2.

Examples

Example 5–1 HTTP POST Request to Set Up a Call

```
POST /rest/third_party_call/calls HTTP/1.1
X-Session-ID: app: -1780934689905632396
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
Content-Length: 105
Content-Type: application/json

{"calledParty":"sip:alice@sipcalled_host:port",
 "charging":
      {
       "description":"testing",
       "amount":"11",
       "code":"1111",
       "currency":"rmb"
      },
 "callingParty":"sip:bob@sipcalling_host:port"
}
```

Example 5–2 HTTP POST Response to Setting Up a Call

```
HTTP:/1.1 201 Created
```

```
Date: Wed, 20 Oct 2010 06:58:06 GMT
Location: http://local:host:8001/rest/third_party
_call/call/app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling
_host|-50d94925ab34bf0
Content-Length: 96
Content-Type: application/json
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1

{"result":"app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling
_host|-50d94925ab34bf0"}
```

Get Call Information

The Get Call Information operation retrieves the information on a previously-established call.

To retrieve the information on a previously-established call, provide the appropriate call identifier in the HTTP GET request. This call identifier should have been obtained by the set up request for the call as the value for `result` in the response received for the HTTP POST request to set up the call. See Example 5–2.

If the operation is successful, the response body will contain the time the call started and the current status of the call. Additionally, if the call was terminated, the response body will indicate the total duration of the call and the reason for its termination.

Authorization

Basic

HTTP Method

`GET`

URI

`http://`*`host`*`:`*`port`*`/rest/third_party_call/call/`*`${callIdentifier}`*

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *callIdentifier* is the call identifier obtained from the response to the POST request to set up the call.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

When the GET request is successful, the response body contains the appropriate values for the following parameters that describe the call:

- **callStatus**: String. The current status of the call as one of the following values:
  - **CallInitial**: The call is being established.
  - **CallConnected**: The call is active.
  - **CallTerminated**: The call was terminated.

- **duration**: Integer. The duration of the call in seconds. Present in the response body when **callStatus** is **CallTerminated**.
- **startTime**: String. The start time for the call in the ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.
- **terminationCause**: String. The reason for the termination of the call. Present in the response body only when **callStatus** is **CallTerminated**. Its value can be one of the following:
  - **CallAborted**
  - **CalledPartyBusy**
  - **CalledPartyNoAnswer**
  - **CalledPartyNotReachable**
  - **CallHangUp**
  - **CallingPartyBusy**
  - **CallingPartyNoAnswer**
  - **CallingPartyNotReachable**

The parameter values are placed in a data structure as the value for **result** in the following JSON structure, where the value part of each name/value pair indicates its data type:

```
{"result": {
  "callStatus": "CallInitial|CallConnected|CallTerminated",
  "duration": "Integer",
  "startTime": "Calendar",
 "terminationCause":"CallingPartyNoAnswer|CalledPartyNoAnswer|CallingPartyBusy|C
alledPartyBusy|CallingPartyNotReachable|CalledPartyNotReachable|CallHangUp|CallA
borted"
}}
```

Examples

Example 5–3 HTTP GET Request

```
GET /rest/third_party
_call/call/app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling
_host|-50d94925ab34bf0 HTTP/1.1
X-Session-ID: app: -123456789012346789
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
```

Example 5–4 HTTP GET Response

```
HTTP:/1.1 201 Created
Date: Wed, 20 Oct 2010 06:58:06 GMT
Content-Length: 124
Content-Type: application/json
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1

{"result":
```

```
  {"startTime":"2010-10-20T4:58:18.254+08:00",
   "terminationCause":null,
   "duration":"0",
   "callStatus":"CallConnected"
  }
}
```

Cancel Call

The Cancel Call operation cancels a previously-requested call that is in its initial state and not yet active. This operation will have no effect if the call is already established.

To cancel a call before it is established, provide the appropriate call identifier in the Request-URI for the POST method. This identifier should have been obtained by the initial setup request for the call.

There is no request or response body for the POST request to cancel a call. If the request fails, the body of the error response will contain the call identifier and the type of exception.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/third_party_call/cancel-call/${callIdentifier}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *callIdentifier* is the call identifier obtained from the response to the HTTP POST reqest to set up the call.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Examples

Example 5–5 HTTP POST Request to Cancel a Call

The POST command to cancel the call contains the required call identifier.

```
POST /rest/third_party
```

```
_call/cancel-call/app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling
_host|-50d94925ab34bf0 HTTP/1.1
X-Session-ID: app: -123456789012346789
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
Content-length: 0
```

Example 5–6 Response for the HTTP POST Request to Cancel a Call

If the request to cancel the call succeeds, you will see a response similar to the following:

```
HTTP:/1.1 204 No Content
Date: Wed, 20 Oct 2010 07:48:14 GMT
Content-Length: 0
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1
```

Example 5–7 Error Response Example

If the request to cancel the call fails, you will see an error response similar to the following. The error body will contain the call identifier as the value for the `correlator` attribute.

```
HTTP:/1.1 500 Internal Server Error
Date: Wed, 20 Oct 2010 07:48:26 GMT
Content-Length: 261
Content-Type: application/json
X-Powered-By: Servlet/2.5 JSP/2.1

{"error" :
   {"message":"Invalid input value for message part Could not find a plugin for
 this message: correlator:
 app-1q39oi07wpvjl|e9674e8214447c1663a016d434c@sipcalling
_host|-50d94925ab34bf0",
    "type":"org.csapi.schema.parlayx.common.v2_1.ServiceException"
   }
}
```

End Call

The End Call operation ends a call that is active.

To end a call, provide the appropriate call identifier in the Request-URI for this operation. This identifier should have been obtained by the initial setup request for the call.

There is no request or response body for this operation. If the request fails, the body of the error response will contain the call identifier and the type of exception.

Authorization

Basic

HTTP Method

POST

URI

`http://host:port/rest/third_party_call/end-call/${callIdentifier}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${callIdentifier}* is the call identifier obtained from the response to the Make Call request.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Examples

Example 5–8 End Call Request

```
POST /rest/third_party_call/end-call/app-1q39oi07wpvjl|
e9674e8214447c1663a016d434c@sipcalling_host|-50d94925ab34bf0 HTTP/1.1
X-Session-ID: app: -123456789012346789
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
```

```
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
Content-length: 0
```

Example 5–9 End Call Response

If the End Call operation succeeds, you will see a response similar to the following:

```
HTTP:/1.1 204 No Content
Date: Wed, 20 Oct 2010 07:48:14 GMT
Content-Length: 0
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1
```

6

Adding RESTful Anonymous Customer Reference Support

This chapter describes how you can add RESTful Anonymous Customer References (ACRs) to provide secure access for Web applications in Oracle Communications Services Gatekeeper.

About Anonymous Customer References

An Anonymous Customer Reference (ACR) represents a unique identifier that replaces a subscriber's secure information, such as MSISDN or phone number, ensuring privacy when the subscriber interacts with Web applications.

According to GSM Association, an ACR is *a string issued by the operator, which maps to a customer (or customers). The operator can pass the ACR in request headers, so you can create an ACR per application per user, and personalize your application based on the user's previous behavior.*

Information about ACR API is available at the GSM webs site:

http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/rest-netapi-acr-v1-0

See the discussion on anonymous customer reference specifications in *Services Gatekeeper Statement of Compliance* for supported specification.

A Web application requiring an ACR for a subscriber requests one from Services Gatekeeper using the RESTful interface. Services Gatekeeper generates and manages one or more ACRs for the subscriber when requested by the Web application.

Configuring ACR Support in Services Gatekeeper

Services Gatekeeper supports ACR operations by default. After you create an ACR plug-in instance, applications can create, query and refresh ACRs using the RESTful interface on a Services Gatekeeper system.

Creating an ACR Plug-in Instance

Services Gatekeeper and the Platform Test Environment MBean interface can be used to create and manage ACR plug-ins. For information on using the Platform Test Environment, see *Services Gatekeeper Platform Test Environment User's Guide*.

To create an instance of the ACR plug-in in Services Gatekeeper:

1. Log in to the Administration Console.

2. Expand the **OCSG** node under **Domain Structure**.
3. Click the name of the administration or managed server on which to create the ACR plug-in instance.
4. Expand the **Container Services** node under **Oracle Communications Services Gatekeeper**.
5. Select **PluginManager**.
6. Click **Operations**.
7. In the **Select An Option** menu, select **createPluginInstsance**.
8. Enter **Plugin_acr** in the **PluginServiceId** field.
9. Enter a unique name in the **PluginInstanceId** field.
10. Click **Invoke**.
11. Add a route to the ACR plug-in using the **pluginManager** Mbean.

Setting ACR Plug-in Parameters

To configure the ACR plug-in attributes in Table 6–1:

1. Log in to the Administration Console.
2. Expand the **OCSG** node under **Domain Structure**.
3. Select the administration or managed server where you created the ACR plug-in.
4. Expand the **Communication Services** node under **Oracle Communications Services Gatekeeper**.
5. Select the ACR plug-in instance to configure.
6. Click **Attributes**.
7. Select the checkboxes of the attributes you want to change.
8. Enter the new values for the attributes.
9. Click **Update Attributes**.

Table 6–1 ACR Plug-in Attributes

Attribute	Type	Description
Ncc	String	The network-code of the operator. In Services Gatekeeper, this is the same as a service provider group.
AcrExpiredLifeTime	Integer	The number of seconds that ACR is kept in Expired state before being deleted. Default Value: *60*
AcrLifeTime	Integer	The number of seconds a generated or refreshed ACR is valid. Default value: **3600**
TrafficAcrMappingEnabled	Boolean	Whether to enable ACR mapping in network traffic. Default value: **False**

Creating Multiple ACRs for a Single Subscriber

You can create multiple ACRs for the same subscriber or MSISDN. Services Gatekeeper creates a unique ACR for each application. Use this setup to route application requests containing ACR identifiers to the correct service provider group.

RESTful APIs for ACR Support

You can generate and manage ACRs in Services Gatekeeper using the RESTful API described below.

- Create ACR
- Query ACR
- Change ACR Status
- Errors and Exceptions

Create ACR

The Create ACR operation creates a new ACR for a subscriber based on the MSISDN.

Authorization

Basic or OAuth

HTTP Method

`POST`

URI

`http://host:port/customerReference/version/address`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *version* is the supported version of the RESTful Network API for Roaming Provisioning.
- *address* is the subscriber identifier MSISDN. Services Gatekeeper supports an address entry of **acr**:*authorization*, where *authorization* is an OAuth accessToken.

> **Note:** *address* must be URL-escaped in accordance with RFC 1738.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for Create ACR accepts the following parameters:

- **acr**: String.
- **status**: String.
- **expiry**: String.

Response Header

The response header indicates whether the ACR was successfully created. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Errors and Exceptions" for more information.

Response Body

The response body contains the following parameters:

- **acr**: String. The Services Gatekeeper generated ACR.
- **status**: String. The current status of the ACR: `Valid` or `Expired`.
- **expiry**: String. The expiration time of the ACR.

Examples

Example 6–1 shows a sample Create ACR request.

Example 6–1 Create ACR Request Example

```
POST http://example.com/customerReference/v1/tel%3A%2B7990123456
HTTP/1.1
Host: example.com:80
Accept: application/json

{"acr":{"status":"Valid"}}
```

Example 6–2 shows a sample Create ACR response.

Example 6–2 Create ACR Response Example

```
HTTP/1.1 201 Created

Content-Type: application/json
Content-Length: 1234
Date: Thu, 04 Jun 2009 02:51:59 GMT

{"acr": "acr:0123456890123456789;ncc=23415",
"status":"Valid",
"expiry":"Thu 11 Jun 2009 02:51:59 GMT"}
```

Query ACR

The Query ACR operation queries for the ACR status of the referenced subscriber. The status of the ACR indicates whether the reference is valid or expired.

Authorization

Basic or OAuth

HTTP Method

`GET`

URI

`http://host:port/customerReference/version/address/acr`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *version* is the supported version of the RESTful Network API for Roaming Provisioning.
- *address* is the subscriber MSISDN.
- *acr* is the ACR for which the status query is being made, including the network-code (ncc).

> **Note:** *address* and *acr* must be URL-escaped in accordance with RFC 1738.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Errors and Exceptions".

Response Body

The response body contains the following parameters:

- **acr**: String. The Services Gatekeeper generated ACR.
- **status**: String. The current status of the ACR: `Valid` or `Expired`.

Examples

Example 6–3 shows a sample Query ACR request.

Example 6–3 Query ACR Request Example

```
GET
http://example.com/customerReference/v1/tel%3A%2B7990123456/acr%3A0123456890123456
789%3Bncc=23415
```

Example 6–4 shows a sample Query ACR response.

Example 6–4 Query ACR Response Example

```
{"acr":{"status":"Valid"}}
```

Change ACR Status

The Change ACR Status operation refreshes an expired ACR to a valid status.

Authorization

Sessionless: Basic, Sessionful: Basic and Session ID, or OAuth

HTTP Method

`POST`

URI

`http://host:port/customerReference/version/address/acr`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *version* is the supported version of the RESTful Network API for Roaming Provisioning.
- *address* is the subscriber MSISDN.
- *acr* is the ACR for which the status query is being made, including the network-code (ncc).

> **Note:** *address* and *acr* must be URL-escaped in accordance with RFC 1738.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for Change ACR Status accepts the following parameters:

- **acr**: String.
- **status**: String.
- **expiry**: String.
- **developerId**: String. Required. The RESTful developer ID.
- **applicationId**: String. Optional. The Services Gatekeeper application ID.

Response Header

The response header indicates whether the ACR status was successfully changed. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Errors and Exceptions" for more information.

Response Body

The response body contains the following parameters:

- **acr**: String. The Services Gatekeeper generated ACR.
- **status**: String. The updated status of the ACR.
- **expiry**: String. The new expiration time of the ACR.

Examples

Example 6–5 shows a sample Change ACR request.

Example 6–5 Change ACR Status Request Example

```
POST
http://example.com/customerReference/v1/tel%3A%2B7990123456/acr%3A0123456890123456
789%3Bncc=23415
HTTP/1.1
Host: example.com:80
Accept: application/json

{"acr":{"status":"Valid"}}
```

Example 6–6 shows a sample Change ACR response.

Example 6–6 Change ACR Status Response Example

```
Content-Type: application/json
Content-Length: 1234
Date: Thu, 04 Jun 2009 02:51:59 GMT

{"acr": "acr:0123456890123456789;ncc=23415",
"status":"Valid",
"expiry":"Thu 11 Jun 2014 02:51:59 GMT"
```

Errors and Exceptions

The Status-Line in the response message indicates the protocol version, the three-digit status code, and the reason for the failure of a request. Table 6–2 lists the possible error codes for failed requests.

Table 6–2 ACR Operations Error Codes

Error Code	Cause
303	The request to create the ACR failed, because the ACR for the MSISDN exists.
400	Bad request. Check the error message and correct the request syntax. For example, a request with {address} whose value is "MSISDN B" is attempting to query/change acr of an {address} whose value is "MSISDN A"
401	The request from network-code A is attempting to change/query acr of network-code B
404	The request is attempting to query an invalid or expired ACR
503	Server busy and service unavailable. Retry the request.

EDRs

Table 6–3 lists the EDRs generated by ACR operations.

Table 6–3 ACR Operations EDRs

EDR	Service	Method	Description
408001	oracle.ocsg.parlayrest.plugin.AcrPlugin	CreateAcrResp createAcr	create acr code for address
408002	oracle.ocsg.parlayrest.plugin.AcrPlugin	QueryAcrResp queryAcr	query acr code for address
408003	oracle.ocsg.parlayrest.plugin.AcrPlugin	ChangeAcrResp ChangeAcr	change acr code for address

7

Adding RESTful Application Subscription Management Support

This chapter describes the operations in the Application Subscription Management interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About Application Subscription Management

The Services Gatekeeper Application Subscription Management supports Open Mobile Alliance (OMA) General Service Subscription Management (GSSM) functionality including subscription management, subscription profile access, and subscription validation.

For information on the OMA GSSM specification see:

http://technical.openmobilealliance.org/Technical/release_program/gssm_v1_0.aspx

Application Subscription Management includes both a communication service and RESTful interfaces for managing and querying service subscription status. Applications use the RESTful interfaces to manage subscriptions and query subscription status. Application Subscription Management grants or restricts application access to a subscriber's communication services based on the subscription status. Application Subscription Management also supports OAuth authentication when required.

For information on using the Application Subscription Management communication service, including deploying, configuring and monitoring, see *Services Gatekeeper Communication Service Reference Guide*.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of Service Subscription Management interface operations can be found at

`http://`*`host`*`:`*`port`*`/subscription/application.wadl`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Subscribe

The Subscribe operation creates a new application service subscription request for a subscriber based on the MSISDN.

Authorization

Basic or OAuth

HTTP Method

`POST`

URI

`http://host:port/subscription`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for Subscribe accepts the following parameters:

- **applicationName**: String. The name of the application subscribed to.
- **subscriberAddress**: String. The MSISDN of the subscriber making the request.

Response Header

The response header indicates whether the subscription request was successfully created. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The response body contains the following parameters:

- **subscriptionResponse**: String. The Services Gatekeeper generated response containing the subscription ID.
  - **subscriptionID**: String. The ID of the new subscription.

Examples

Example 7–1 shows a sample Subscribe request.

Example 7–1 Subscribe Request Example

```
POST /subscription HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: Basic YXBwX3VzZXI6YXBwX3VzZXI=
```

```
Accept: application/json
Content-Length: 60
Host: vm-at1:8001
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
.
applicationName=Mickey News&subscriberAddress=tel:0703322124
.
```

Example 7–2 shows a sample Subscribe response.

Example 7–2 Subscribe Response Example

```
HTTP/1.1 201 Created
Date: Wed, 01 Jul 2015 07:13:07 GMT
Content-Length: 78
Content-Type: application/json
.
{"subscriptionResp":{"subscriptionId":"1a7afc44-9e68-4dc6-aaf4-07610a9fdef0"}}
```

Unsubscribe

The Unsubscribe operation is used to request a subscription deletion.

Authorization

Basic or OAuth

HTTP Method

`DELETE`

URI

`http://host:port/subscription`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The Unsubscribe request header contains the subscription ID to be deleted. The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no response body.

Response Header

The response header indicates whether the ACR status was successfully changed. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The response body contains no content.

Examples

Example 7–3 shows a sample Unsubscribe request.

Example 7–3 Unsubscribe Request Example

```
DELETE /subscription/e4e3cb51-994b-46e1-b0d1-0757a8bca25f HTTP/1.1
Host: example.com:80
Accept: application/json
```

Example 7–4 shows a sample Unsubscribe response.

Example 7–4 Unsubscribe Response Example

```
HTTP/1.1 204 No Content
```

Suspend

The Suspend operation suspends an existing service subscription for a subscriber based on the subscription ID.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://host:port/subscription/suspend/subscriptionID`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *subscriptionID* is the ID of the subscription to be suspended.

Request Header

The Suspend request header contains the ID of the subscription to be suspended. The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for Suspend is empty.

Response Header

The response header indicates whether the subscription was successfully suspended. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The response body contains no content.

Examples

Example 7–5 shows a sample Suspend request.

Example 7–5 Suspend Request Example

```
PUT /subscription/suspend/e4e3cb51-994b-46e1-b0d1-0757a8bca25f HTTP/1.1
Host: example.com:80
Accept: application/json
```

Example 7–6 shows a sample Suspend response.

Example 7–6 Suspend Response Example

```
HTTP/1.1 204 No Content
```

Unsuspend

The Unsuspend operation unsuspends an existing service subscription for a subscriber based on subscription ID.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://host:port/subscription/unsuspend/subscriptionID`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *subscriptionID* is the ID of the subscription to be suspended.

Request Header

The Unsuspend request header contains the subscription ID to be unsuspended. The MIME-type for the Content-Type header field is **application/json**

Request Body

The request body for unsuspend is empty.

Response Header

The response header indicates whether the subscription was successfully removed from being suspended. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The response body contains no content.

Examples

Example 7–7 shows a sample Unsuspend request.

Example 7–7 Unsuspend Request Example

```
PUT /subscription/unsuspend/e4e3cb51-994b-46e1-b0d1-0757a8bca25f HTTP/1.1
Host: example.com:80
Accept: application/json
```

Example 7–8 shows a sample Unuspend response.

Example 7–8 Unsuspend Response Example

```
HTTP/1.1 204 No Content
```

Notify

The Notify operation notifies a registered application of a subscription management request.

Authorization

Basic or OAuth

HTTP Method

`POST`

URI

`http://host:port/notify`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for Notify accepts the following parameters:

- **subscriptionNotification**. The Services Gatekeeper generated notification containing the following parameters:
  - **subscriptionID**: String. The ID of the subscription request.
  - **applicationName**: String. The name of the application receiving the request.
  - **subscriberAddress**: String. The MSISDN of the requesting subscriber.
  - **operation**: String. The subscription request type. For example, **Subscribe** or **Unsubscribe**.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The response body contains no content.

Examples

Example 7–9 shows a sample Notify request.

Example 7–9 Notify Request Example

```
POST /notifysubscription HTTP/1.1
```

```
Host: example.com:80
Accept: application/json
Content-Type: application/json
Content-Length: 173
{"subscriptionNotification":{"subscriptionId":
"e4e3cb51-994b-46e1-b0d1-0757a8bca25f","applicationName":"Oracle
News","subscriberAddress":"tel:1111","operation":"Subscribe"}}
```

Example 7–10 shows a sample Notify response.

Example 7–10 Notify Response Example

```
HTTP/1.1 204 No Content
```

Confirm

The Confirm operation is used by applications to confirm subscription requests.

Authorization

Basic or OAuth

HTTP Method

`PUT`

URI

`http://host:port/subscription/confirm`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for Confirm accepts the following parameters:

- **operation**: String. The subscription request operation being confirmed. For example, **Subscribe** or **Unsubscribe**.
- **confirmResult**: String. The subscription request status. For example, **approved**.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The response body contains the following parameter:

- **oauthAccessToken**: String. If OAuth is required, Services Gatekeeper returns an OAuth accessToken in the response.

Examples

Example 7–11 shows a sample Confirm request.

Example 7–11 Confirm Request Example

```
PUT /subscription/confirm/e4e3cb51-994b-46e1-b0d1-0757a8bca25f HTTP/1.1
Host: example.com:80
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Content-Length: 42
operation=Subscribe&confirmResult=Approved
```

Example 7–12 shows a sample Confirm response with an OAuth accessToken.

Example 7–12 Confirm Response Example

```
HTTP/1.1 200 OK
Content-Type: application/json
Date:  Wed, 07 Nov 2012 06:39:50 GMT
{"oauthAccessToken":{"oauthAccessToken":"e4e3cb51-994b-46e1-b0d1-0757a8bca25f"}}
```

queryBySubscriberAddress

The queryBySubscriberAddress operation retrieves a user's subscription information based on the subscriber address.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/subscription/query/queryBySubscriberAddress`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The queryBySubscriberAddress request header contains the subscriber address (MSISDN) to be queried. The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for queryBySubscriberAddress is empty.

Response Header

The response header includes the request status. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The request body for queryBySubscriberAddress contains the following parameters:

- **subscriptionList**. The Services Gatekeeper generated list containing the following parameters:
  - **subscriptionInfo**. Array. Contains the following parameters for each subscription.
    - **applicationName**: String. The name of the subscribed application.
    - **subscriberAddress**: String. The MSISDN of the subscriber.
    - **status**: String. The subscription status. For example, **Pending**.

Examples

Example 7–13 shows a sample queryBySubscriberAddress request.

Example 7–13 queryBySubscriberAddress Request Example

```
GET /subscription/query/queryBySubscriberAddress/tel:1111?offSet=0&batchSize=2
 HTTP/1.1
Host: example.com:80
Accept: application/json
```

Example 7–14 shows a sample queryBySubscriberAddress response.

Example 7–14 queryBySubscriberAddress Response Example

```
HTTP/1.1 200 OK
Content-Type: application/json
Date:  Wed, 07 Nov 2012 06:56:05 GMT

{"subscriptionList":{"subscriptionInfo":[{"applicationName":"coderslist",
"subscriberAddress":"tel:1111","status":"SubscribePending"},{"applicationName":
"Oracle News","subscriberAddress":"tel:1111",
"status":"UnSubscribePending"}]}}
```

queryByApplicationName

The queryByApplicationName operation retrieves a list of subscribers to an application based on the application ID.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/subscription/query/queryByApplicationName`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The queryByApplicationName request header contains the application name to be queried. The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for queryBySubscriberAddress is empty.

Response Header

The response header includes the request status. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The request body for queryByApplicationName contains the following parameters:

- **subscriptionList**. The Services Gatekeeper generated list containing the following parameters:
  - **subscriptionInfo**. Array. Contains the following parameters for each subscribed user.
    - **applicationName**: String. The name of the subscribed application.
    - **subscriberAddress**: String. The MSISDN of the subscriber.
    - **status**: String. The subscription status. For example, **Pending**.

Examples

Example 7–15 shows a sample queryByApplicationName request.

Example 7–15 queryByApplicationName Request Example

```
GET /subscription/query/queryByApplicationName/Oracle%20News?offSet=0&
batchSize=2 HTTP/1.1
Host: example.com:80
Accept: application/json
```

Example 7–16 shows a sample queryByApplicationName response.

Example 7–16 queryByApplicationName Response Example

```
HTTP/1.1 200 OK
Content-Type: application/json
Date:  Wed, 07 Nov 2012 06:56:05 GMT

{"subscriptionList":{"subscriptionInfo":[{"applicationName":"Oracle
News","subscriberAddress":"tel:1111","status":"UnSubscribePending"},
{"applicationName":"Oracle
 News","subscriberAddress":"tel:1112","status":"Active"}]}}
```

queryBySubscriptionID

The queryBySubscriptionID operation retrieves application subscription information for a subscriber based on the subscription ID.

Authorization

Basic

HTTP Method

GET

URI

`http://host:port/subscription/query/queryBySubscriptionID`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The queryBySubscriptionID request header contains the subscription ID to be queried. The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for queryBySubscriptionID is empty.

Response Header

The response header includes the request status. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See *Services Gatekeeper Communication Service Reference Guide*, for more information on error messages.

Response Body

The request body for queryBySubscriptionID contains the following:

- **subscriptionInfo**. The Services Gatekeeper generated list with the following parameters:
  - **applicationName**: String. The name of the subscribed application.
  - **subscriberAddress**: String. The MSISDN of the subscriber.
  - **status**: String. The subscription status. For example, Pending.

Examples

Example 7–17 shows a sample queryBySubscriptionID request.

Example 7–17 queryBySubscriptionID Request Example

```
GET /subscription/query/queryBySubscriptionId/e4e3cb51-994b-46e1-b0d1-0757
a8bca25f HTTP/1.1
```

```
Host: example.com:80
Accept: application/json
```

Example 7–18 shows a sample queryBySubscriptionID response.

Example 7–18 queryBySubscriptionID Response Example

```
HTTP/1.1 200 OK
Content-Type: application/json
Date:  Wed, 07 Nov 2012 06:56:05 GMT

{"subscriptionInfo":{"applicationName":"Oracle
 News","subscriberAddress":"tel:1111","status":"UnSubscribePending"}}
HTTP/1.1 200 OK
```

8

Adding RESTful Short Messaging Support

This chapter describes the operations in the Short Messaging interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Short Messaging Interface

Applications use the RESTful Short Messaging interface to send an SMS, a ringtone, or a logo, to fetch SMS messages and delivery status reports; and to start and stop a notification.

When the request body for an SMS operation contains a request for a delivery receipt, the application provides a correlator for the message being sent and includes an endpoint address for returning the delivery notification.

For such operations, the application client subscribes to and uses the Bayeux channel to which Services Gatekeeper is subscribed. The application provides the Bayeux channel name as the location for Services Gatekeeper to publish the notifications that the application client requires. See "RESTful Notifications and Publish/Subscribe" for more information about publishing and subscribing to RESTful notifications.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://host:port/rest/sms/index.html`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Send SMS

The Send SMS operation delivers a text message.

To send an SMS message, provide the URI of the addresses which must receive the message in the request body. If there is to be a charge for the messaging, the request body should contain the required charging object. If the sender requires a delivery receipt, specify the required parameters for the receipt.

If the Send SMS operation is successful, the **Location** header field in the response will contain the request identifier (which is also provided in the response body for this operation).

If the application requested a receipt for delivery of the message, the application must access the **endpoint** address to retrieve the delivery notifications.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/sms/messages`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Send SMS operation accepts the following parameters:

- **addresses**: Array of string values. Required. The set of addresses of the recipients as an array of URIs.
- **message**: String. Required. The text of the message.
- **charging**: A JSON object. Optional. This object defines the cost charging properties for the operation. The entry `"charging": null` indicates no charge. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**: Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **receiptRequest**: A JSON object. Optional. If a delivery receipt is required, provide values for each of the following parameters which define this object:

  - **correlator**: String. Required. Used to correlate the receipt with the initial message.

    If the callback reference **correlator** is defined in the **receiptRequest** object of the Send SMS request message, Services Gatekeeper will invoke the Notify SMS Delivery Receipt operation and discard the delivery status information.

  - **endpoint**: String. Required. The endpoint address (URI) to which the receipt must be delivered.

  - **interfaceName**: String. Required. A description provided to identify the type of receipt.

- **senderName**: String. Optional. The sender's name.

The request body for this operation is represented by the following JSON structure, where the value part of each name/value pair indicates its data type:

```
{
  "addresses": ["URI"],
  "message": "String",
  "charging": {
    "description": "String",
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "receiptRequest": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "senderName": "String"
}
```

Response Header

The Location header field contains the URI:

`http://`*`host`*`:`*`port`*`/rest/sms/delivery_status/`*`result`*

where, *result* is the string identifier returned in the response body.

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The body of the response contains the request identifier as the string value for the **result** object; it is the request identifier returned in the **Location** header field of the response message. The application uses this request identifier (in the Get SMS Delivery Status operation) to retrieve the delivery status for the sent message.

The response body for this operation is represented by the following JSON structure, where the value part of the name/value pair indicates its data type:

```
{"result": "String"}
```

Send SMS Ringtone

The Send SMS Ringtone operation delivers a ringtone.

To send an SMS Ringtone message, provide the URI of the addresses which must receive the message, the ringtone data and its format in the request body for this operation. If there is to be a charge for the messaging, the request body should contain the required charging object. If the sender requires a delivery receipt, specify the required parameters for the receipt.

If the Send SMS Ringtone operation is successful, the **Location** header field in the response will contain the request identifier (which is also provided in the response body for this operation).

If the application requested a receipt for delivery of the message, the application must access the endpoint address to retrieve the delivery notifications.

Authorization

Basic

HTTP Method

`POST`

URI

`http://`*`host`*`:`*`port`*`/rest/sms/ringtones`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body is a nested JSON object containing the following parameters:

- **addresses**: Array of string values. Required. The set of addresses of the recipients as an array of URIs.
- **ringtone**: String. Required. The ringtone data in Extended Ringtone (RTX) format.
- **smsFormat**: String. Required. The encoding format for the ringtone entered as one of the following:
  - **Ems**
  - **SmartMessaging**
- **charging**: A JSON object. Optional. This object defines the cost charging properties for the operation. The entry `"charging": null` indicates no charge. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**: Number (integer, or decimal). Optional. The amount to be charged.

  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **receiptRequest**: A JSON object. Optional. If a delivery receipt is required, provide values for each of the following parameters which define this object:
  - **correlator**: String. Required. Used to correlate the receipt with the initial message.

    If the callback reference **correlator** is defined in the **receiptRequest** object of the Send SMS request message, Services Gatekeeper will invoke the Notify SMS Delivery Receipt operation and discard the delivery status information.
  - **endpoint**: String. Required. The endpoint address (URI) to which the receipt must be delivered.
  - **interfaceName**: String. Required. A description provided to identify the type of receipt.
- **senderName**: String. Optional. The sender's name.

The request body for this operation is represented by the following JSON structure, where the value part of each name/value pair indicates its data type:

```
{
  "addresses": ["URI"],
  "ringtone": "String",
  "smsFormat": "Ems|SmartMessaging",
  "charging": {
    "description": "String",
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "receiptRequest": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "senderName": "String"
}
```

Response Header

`http://`*`host`*`:`*`port`*`/rest/sms/delivery_status/`*`result`*

where, *result* is the string identifier returned in the request body.

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The body of the response contains the **result** object whose value is the request identifier returned in the Location header field of the response message. The application uses this request identifier (in the Get SMS Delivery Status operation) to retrieve the delivery status for the sent message.

The response body for this operation is represented by the following JSON structure, where the value part of the name/value pair indicates its data type:

```
{"result": "String"}
```

Send SMS Logo

The Send SMS Logo operation delivers a logo.

To send an SMS Logo, provide the URI of the addresses which must receive the message, the logo data and its format in the request body for this operation. If there is to be a charge for the messaging, the request body should contain the required charging object. If the sender requires a delivery receipt, specify the required parameters for the receipt.

If the Send SMS Logo operation is successful, the **Location** header field in the response will contain the request identifier (which is also provided in the response body for this operation).

If the application requested a receipt for delivery of the message, the application must access the **endpoint** address to retrieve the delivery notifications.

Authorization

Basic

HTTP Method

POST

URI

`http://host:port/rest/sms/logo`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Send SMS operation accepts the following parameters:

- **addresses**: Array of string values. Required. The set of addresses of the recipients as an array of URIs.
- **image**: String. Required. The logo data, a base64-encoded image in GIF, PNG or JPG format.
- **smsFormat**: String. Required. The encoding format for the logo entered as one of the following:
  - **Ems**
  - **SmartMessaging**
- **charging**: A JSON object. Optional. Use this object to define the cost charging properties for the operation. The entry `"charging": null` indicates no charge. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**: Number (integer, or decimal). Optional. The amount to be charged.

  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **receiptRequest**: A JSON object. Optional. If a delivery receipt is required, provide values for each of the following parameters which define this object:
  - **correlator**: String. Required. Used to correlate the receipt with the initial message.

    If the callback reference **correlator** is defined in the **receiptRequest** object of the Send SMS request message, Services Gatekeeper will invoke the Notify SMS Delivery Receipt operation and discard the delivery status information.
  - **endpoint**: String. Required. The endpoint address (URI) to which the receipt must be delivered.
  - **interfaceName**: String. Required. A description provided to identify the type of receipt.
- **senderName**: String. Optional. The sender's name.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "addresses": ["URI"],
  "image": "base64Binary",
  "smsFormat": "Ems|SmartMessaging",
  "charging": {
    "description": "String",
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "receiptRequest": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "senderName": "String"
}
```

Response Header

The Location header field contains the URI:

`http://host:port/rest/sms/delivery_status/result`

where, *result* is the string identifier returned in the request body.

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains the **result** object whose value is the request identifier returned in the Location header field of the response message. The application uses this request identifier (in the Get SMS Delivery Status operation) to retrieve the delivery status for the sent message.

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": "String"}
```

Get Received SMS

The Get Received SMS operation polls Services Gatekeeper for the SMS messages that have been received from the network for an application.

The request header for the Get Received SMS operation contains the registration identifier necessary to retrieve the required SMS messages intended for the application. This registration value should have been set up with the off-line provisioning step that enables the application to receive notification of SMS reception.

There is no request body.

If the Get Received SMS operation is successful, the response body will contain the message, the URI of the sender, the SMS service activation number and the date and time when the message was sent.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/sms/receive-messages/${registrationIdentifier}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *registrationIdentifier* is the value previously set up to enable the application to receive notification of SMS reception according to specified criteria.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains an array of structures as the value for **result**. Each element in the array contains values for the following parameters.

- **message**: String. The text of the SMS message.
- **senderAddress**: String. The URI of the sender.

- **smsServiceActivationNumber**: String. URI. The number associated with the invoked message service, that is the destination address used to send the message.
- **dateTime**: String. The start time for the call in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.

The response body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"result": [{
  "message": "String",
  "senderAddress": "URI",
  "smsServiceActivationNumber": "URI",
  "dateTime": "Calendar"
}]}
```

Get SMS Delivery Status

The Get SMS Delivery Status operation retrieves the delivery status of a message that was previously sent using Send SMS, Send SMS Ringtone or Send SMS Logo operation.

The Get SMS Delivery Status operation is valid only if the **correlator** callback reference was not defined in the **receiptRequest** object of the initial request to send the short message. In such a scenario, Services Gatekeeper stores the delivery status for a configurable period.

If the callback reference correlator is defined in the **receiptRequest** object of the initial request to send the short message, Services Gatekeeper will invoke the Notify SMS Delivery Receipt operation and discard the delivery status information.

The request header for Get SMS Delivery Status contains the request identifier from the initial Send operation for the short message.

If Get SMS Delivery Status is successful, the response body will contain the URIs from the **address** field in the request body of the Send operation for the short message and corresponding delivery status for each message.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/sms/delivery-status/${requestIdentifier}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *requestIdentifier* is the identifier returned in the **result** object of the corresponding Send operation.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains an array of structures as the value for **result**. Each element in the array contains values for the following parameters.

- **address**: String. The URI to which the initial message was sent.

- **deliveryStatus**: Enumeration value. Table 8–1 lists the possible status:

Table 8–1 Enumeration Values for Delivery Status

Value	Description
DeliveredToNetwork	Successful delivery to network. For concatenated messages, returned only when all the SMS-parts have been successfully delivered to the network.
DeliveryUncertain	Delivery status unknown; for example, if the message was handed off to another network.
DeliveryImpossible	Unsuccessful delivery; the message could not be delivered before it expired.
MessageWaiting	The message is still queued for delivery. This is a temporary state, pending transition to one of the preceding states.
DeliveredToTerminal	Successful delivery to terminal. For concatenated messages, returned only when all the SMS-parts have been successfully delivered to the terminal.
DeliveryNotificationNotSupported	Unable to provide delivery receipt notification.

The response body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"result": [{
  "address": "URI",
  "deliveryStatus":
"DeliveredToNetwork|DeliveryUncertain|DeliveryImpossible|MessageWaiting|DeliveredT
oTerminal|DeliveryNotificationNotSupported"
}]}
```

Start SMS Notification

The Start SMS Notification operation starts a notification for the application.

To set up an SMS notification, provide the criteria which will trigger notifications and a reference object for the delivery of the notifications. The criteria can be a string which, when matched, could be the notification of an SMS received or of a delivery receipt. The reference object (also a JSON object) contains the correlator for the notification, the **endpoint** address (a specific Bayeux channel name) to which the call direction notifications must be sent and, optionally, the interface name (a string to identify the notification).

If the Start SMS Notification request is successful:

- The response header will contain the URI of the publish/subscribe server.
- A data object associated with the result of the short message operation will be sent to the **endpoint** address specified in the request body. This data object will contain the appropriate notification (that the message was received or a delivery receipt for the call).

When the Start SMS notification request is successful, the application can access the **endpoint** address to retrieve the specific call event notifications.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://host:port/rest/sms/notification`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Start SMS Notification operation accepts the following parameters:

- **criteria**: String. Optional. The text to match against in determining whether or not the application should receive the notification:
  - **notifySmsReception**
  - **notifySmsDeliveryReceipt**
- **reference**. JSON object. Required. Use this object to provide the following information about the endpoint that is to receive the notification:
  - **correlator**: String. Required. The correlator used to identify the notification.

  - **endpoint**: String. Required. The URI which represents the endpoint address to which the notification should be delivered. This string should be a Bayeux protocol channel name that begins with **/bayeux/***appInstanceID* where *appInstanceID* is the client application's application instance account ID.

    For more information on application instances, see the discussion on managing application instances in *Services Gatekeeper Portal Developer's Guide*.
  - **interfaceName**: String. Required. A descriptive string to identify the type of notification.
- **smsServiceActivationNumber**: String. Required. The number associated with the invoked message service, that is the destination address used to send the message

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
  "reference": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "smsServiceActivationNumber": "URI",
  "criteria": "String"
}
```

Response Header

The Location header field contains the URI of the publish/subscribe server:

`http://host:port/rest/sms/notifications`

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body. The appropriate notifications (described below) are sent to the **endpoint** address provided by the application in the request body of this operation.

Notification Data Object for SMS Reception (notifySmsReception)

When an SMS message is successfully retrieved by the called party and the value for **criteria** in the request body is **notifySmsReception**, Services Gatekeeper will send a nested JSON data object to the **endpoint** address.

notifySmsReception is a nested JSON object and contains the following values:

- **correlator**: String. Required. The correlator used to identify the notification.
- **message**: Nested JSON object. Required. It contains the following:
  - **message**: String. Required. The contents of the SMS message.
  - **senderAddress**: String. Required. The URI of the sender of the SMS message.
  - **smsServiceActivationNumber**: String. Required. The number associated with the invoked message service, that is the destination address used to send the message.

– **dateTime**: String. Optional. The time at which the message was sent in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.

The notification data object delivered to the **endpoint** address when the criteria is **notifySmsReception** is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"notifySmsReception": {
  "correlator": "String",
  "message": {
    "message": "String",
    "senderAddress": "URI",
    "smsServiceActivationNumber": "URI",
    "dateTime": "Calendar"
  }
}}
```

Notification Data Object for SMS Delivery Receipt (notifySmsDeliveryReceipt)

When an SMS message is successfully retrieved by the called party and the value for **criteria** in the request body is , Services Gatekeeper sends a nested JSON data object to the **endpoint** address.

notifySmsDeliveryReceipt is a nested JSON object and contains the following values:

- **correlator**: String. The correlator used to identify the notification.
- **deliveryStatus**. Enumeration value. Table 8–2 lists the possible status values:

Table 8–2 Enumeration Values for Delivery Status

Value	Description
DeliveredToNetwork	Successful delivery to network. For concatenated messages, this is the value only when all the parts of the SMS message have been successfully delivered to the network.
DeliveryUncertain	Delivery status unknown; for example, if the message was handed off to another network.
DeliveryImpossible	Unsuccessful delivery; the message could not be delivered before it expired.
DeliveredToTerminal	Successful delivery to terminal. For concatenated messages, this is the value only when all the parts of the SMS message have been successfully delivered to the terminal.
DeliveryNotificationNotSupported	Unable to provide delivery receipt notification.

The notification data object delivered to the **endpoint** address when the criteria is **notifySmsDeliveryReceipt** is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"notifySmsDeliveryReceipt": {
  "correlator": "String",
  "deliveryStatus": {
    "address": "URI",
    "deliveryStatus":
"DeliveredToNetwork|DeliveryUncertain|DeliveryImpossible|MessageWaiting|
DeliveredToTerminal|DeliveryNotificationNotSupported"
  }
```

```
}}
```

Stop SMS Notification

The Stop SMS Notification operation terminates a previously set up SMS notification for the application.

To stop a previously set up SMS notification, provide the correlator for the notification passed earlier in the Start SMS Notification request.

There is no request or response body for the Stop SMS Notification operation. If the request fails, the body of the error response will contain the identifier for the notification and the type of exception.

Authorization

Basic

HTTP Method

`DELETE`

URI

`http://host:port/rest/sms/notification/${correlator}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *correlator* is the correlator for the notification provided in the **reference** object of the initial Start SMS Notification request body.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Examples

Example 8–1 Stop Message Notification Request

```
DELETE /rest/sms/notification/12345 HTTP/1.1
X-Session-ID: app:4130997928482260925
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
```

Example 8–2 Stop Message Notification Response

```
HTTP/1.1 204 No Content
Connection: close
Date: Thu, 04 Nov 2101 09:59:05 GMT
Content-Length: 0
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1
```

9

Adding RESTful Multimedia Messaging Support

This chapter describes the operations in the Multimedia Messaging interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Multimedia Messaging Interface

Applications use the RESTful Multimedia Messaging interface to send a multimedia message (MMS message) and to fetch information on MMS messages that have been received for the applications and stored on Services Gatekeeper.

Applications use the interface to fetch those messages, get delivery status on sent messages, and start and stop a notification.

The actual message is sent as an attachment. See "Headers for Multipart Messages with Attachments" for more information.

For an application to be informed that a party has received an MMS message or that a message delivery receipt has been sent, the application includes an application **endpoint** address for such delivery notifications in the body of the request message. This address resides on a publish/subscribe server. See "RESTful Notifications and Publish/Subscribe" for more information about publishing and subscribing to RESTful notifications.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://host:port/rest/multimedia_messaging/index.html`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Send Message

The Send Message operation delivers a multimedia message.

To send a multimedia message, provide the URI of the addresses which the must receive the message in the request body. If there is to be a charge for the messaging, the request body should contain the required charging object. If the sender requires a delivery receipt, specify the required parameters for the receipt and the URI address of the sender. Also, a priority value, and a subject line can be provided in the request body.

If the Send Message operation is successful, the response header will contain the URI of the publish/subscribe server and the request identifier which is also provided in the response body for this operation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/multimedia_messaging/messages`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The request headers depend on the type of message:

- If the message content is in the form of an attachment, the request will be multipart and the request header must contain the header fields that describe the parts of the message.
- If the message does not contain an attachment, the special headers associated with the multipart messaged are not used.

See Example 9–1 for a listing of both types of headers.

Message Part Content

The Message Part Content for the Send Message operation accepts the following parameters:

- **addresses**: Array of string values. Required. The set of addresses of the recipients as an array of URIs.
- **subject**: String. Optional. The text of the message.
- **priority**: String. Optional. The message priority specified as one of the following:
  - **Default**
  - **Low**
  - **Normal**

  - **High**

- **senderAddress**: String. Optional. The sender's address (URI).
- **charging**: A JSON object. Optional. This object defines the cost charging properties for the operation. The entry `"charging":null` indicates no charge. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**: Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **receiptRequest**: A JSON object. Optional. If a delivery receipt is required, provide values for each of the following parameters which define this object:
  - **correlator**: String. Required. Used to correlate the receipt with the initial message.

    If the callback reference **correlator** is defined in the **receiptRequest** object of the Send Message request, Services Gatekeeper will invoke Notify Message Delivery Receipt and discard the delivery status information.
  - **endpoint**: String. The endpoint address (URI) to which the receipt must be delivered: a Bayeux protocol channel name that is the client application's application instance account ID.
  - **interfaceName**: String. Required. A description provided to identify the type of receipt.

The message part Content for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "addresses": ["URI"],
  "charging": {
    "description": "String",
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "priority": "Default|Low|Normal|High",
  "receiptRequest": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "senderAddress": "String",
  "subject": "String"
}
```

Response Header

The Location header field contains the URI of the publish/subscribe server as:

`http://`*`host`*`:`*`port`*`/rest/multimedia_messaging/delivery-status/`*`result`*

where, *result* is the string identifier for the request that is returned in the response body.

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The body of the response contains the request identifier returned in the **Location** header field of the response message as value for **result**.

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": "String"}
```

Examples

Example 9–1 Send Message Requests (SIngle Part Message)

This first example shows a POST method with a message in a single part.

```
POST /rest/multimedia_messaging/messages HTTP/1.1
X-Session-ID: app:-7603991555266551180
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
Content-Length: 253
Content-Type: application/json

{
  "addresses":["tel:8765","tel:7654","tel:6543"],
  "subject":"Meeting starts at 2 p.m.",
  "priority":null,
  "senderAddress":"tel:1234",
  "charging":null
  "receiptRequest":
         {
           "correlator":"981234",
           "endpoint":"http://endpt_host:port/jaxws/MessageNotification",
           "interfaceName":"interfaceName",
         }
 }
```

Example 9–2 Send Message Requests (SIngle Part Message)

The second example shows a POST method with a multipart message. Note the use of header fields to describe the content and the boundary attribute to distinguish the message parts.

```
POST /rest/multimedia_messaging/messages HTTP/1.1
X-Session-ID: app: -123456789012346789
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
```

```
Content-Length: 1215
Content-Type: multipart/form-data; boundary=kboiiFPAakDPYKeY7hBAW9I5c0rT48

--kboiiFPAakDPYKeY7hBAW9I5c0rT48
Content-Disposition: form-data; name="messagePart"
Content-Type: application/json; charset=US-ASCII
Content-Transfer-Encoding: 8bit

{
  "addresses":["tel:8765","tel:7654","tel:6543"],
  "subject":"Hello World",
  "priority":null,
  "senderAddress":"tel:1234",
  "charging":null,
  "receiptRequest":
         {
           "correlator":"981234",
           "endpoint":"http://endpt_host:port/jaxws/MessageNotification",
           "interfaceName":"interfaceName"
         }
 }
--kboiiFPAakDPYKeY7hBAW9I5c0rT48
Content-Disposition: form-data; name="Attachment-txt-1"
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 8bit

Oracle Communications Services Gatekeeper is a powerful, flexible, secure
interface which provides a simple way for application developers to include
telephony-based functionality in their software applications and guarantee the
security, stability, and performance required by network operators and demanded by
their subscribers.
--kboiiFPAakDPYKeY7hBAW9I5c0rT48
```

Get Received Messages

The Get Received Messages operation retrieves an array of network-triggered messages that have arrived at Services Gatekeeper.

The request body for Get Received Messages contains the registration identifier necessary to retrieve the required multimedia messages intended for the application and, optionally, it may also contain a priority for the retrieval. The registration identifier should have been set up with the off-line provisioning step completed earlier.

If the Get Received Messages operation is successful, the response body will contain the data about the message. If the content of the message is pure ASCII, the response body contains the message. Otherwise the response body contains an identifier that is used to fetch the actual message.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/multimedia_messaging/receive-messages`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Get Received Messages operation accepts the following parameters:

- **registrationIdentifier**: String. Required. The value previously set up to enable the application to receive notification of multimedia message reception according to specified criteria.
- **priority**: String. Optional. The message priority specified as one of the following:
  - **Default**
  - **Low**
  - **Normal**
  - **High**

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "registrationIdentifier": "String",
  "priority": "Default|Low|Normal|High"
```

```
}
```

Response header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains an array of structures as the value for **result**. Each element in the array contains values for the following parameters.

- **messageServiceActivationNumber**: String. The number associated with the invoked message service, that is the destination address used to send the message.
- **priority**: String. The message priority value from the request.
- **senderAddress**: String. The URI of the sender.
- **dateTime**: String. The start time for the call in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.
- **message**: String. Present if the text of the message is purely ASCII text.
- **messageIdentifier**: String. An identifier used to fetch the message, when it is not pure ASCII text. This value is used in the URI of the request for the "Get Message" operation.
- **subject**: String. The subject line of the multimedia message.

The response body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"result": [{
  "messageServiceActivationNumber": "String",
  "priority": "Default|Low|Normal|High",
  "senderAddress": "URI",
  "dateTime": "Calendar",
  "message": "String",
  "messageIdentifier": "String",
  "subject": "String"
}]}
```

Get Message

The Get Message operation fetches a network-triggered message for the application from Services Gatekeeper.

To get a message, provide the appropriate message identifier in the URI for the GET method.

The actual message is returned as an attachment.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/multimedia_messaging/message/${messageRefIdentifier}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *messageRefIdentifier* is the value for *messageIdentifier* in:
  - The response body of the "Get Received Messages" operation.
  - The **notifyMessageReception** data object. See "Notification Data Object for Message Reception (notifyMessageReception)".

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

The MIME-type for the Content-Type header field is **multipart/mixed**.

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Get Message Delivery Status

The Get Message Delivery Status operation retrieves the delivery status of a previously sent message.

This operation is valid only if the **correlator** callback reference was not defined in the **receiptRequest** object of the initial request to send the multimedia message. In such a scenario, Services Gatekeeper stores the delivery status for a configurable period. If the callback reference correlator is defined in the **receiptRequest** object of the initial multimedia message request, Services Gatekeeper will invoke the Notify Message Delivery Receipt operation and discard the delivery status information.

The request header for the Get Message Delivery Status operation contains the request identifier from **result** object of the initial request.

If the Get Message Delivery Status operation is successful, the response body will contain the URI from the address field in the Send Message request and corresponding delivery status for the message.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/multimedia_messaging/delivery-status/${requestIdentifier}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *requestIdentifier* is a string returned in the **result** object of the initial Send Message request.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains an array of structures as the value for **result**. Each element in the array contains values for the following parameters.

- **address**: String. The URI to which the initial message was sent.

- **deliveryStatus**. Enumeration value. Table 9–1 lists the possible statuses:

Table 9–1 Enumeration Values for Delivery Status

Value	Description
DeliveredToNetwork	Successful delivery to network. For concatenated messages, returned only when all the MMS parts have been successfully delivered to the network.
DeliveryUncertain	Delivery status unknown; for example, if it was handed off to another network.
DeliveryImpossible	Unsuccessful delivery; the message could not be delivered before it expired.
MessageWaiting	The message is still queued for delivery. This is a temporary state, pending transition to one of the preceding states.
DeliveredToTerminal	Successful delivery to terminal. For concatenated messages, returned only when all the MMS parts have been successfully delivered to the terminal.
DeliveryNotificationNotSupported	Unable to provide delivery receipt notification.

The response body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"result": [{
  "address": "URI",
  "deliveryStatus":
"DeliveredToTerminal|DeliveryUncertain|DeliveryImpossible|MessageWaiting|
DeliveredToNetwork|DeliveryNotificationNotSupported"
}]}
```

Start Message Notification

The Start Message Notification operation starts a notification for the application.

To set up a multimedia message notification, provide the criteria which will trigger notifications and a reference object for the delivery of the notifications. The criteria can be a string which, when matched, could be the notification of a multimedia message received or of a delivery receipt. The reference object (also a JSON object) contains the correlator for the notification, the endpoint address (a specific Bayeux channel name) to which the call direction notifications must be sent and, optionally, the interface name (a string to identify the notification).

If the Start Message Notification request is successful:

- The response header will contain the URI of the publish/subscribe server.
- A data object associated with the result of the short message operation will be sent to the endpoint address specified in the request body. This data object will contain the appropriate notification (that the message was received or a delivery receipt for the call).

When the application receives such a response, it must access the endpoint address to retrieve the specific call event notifications.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://host:port/rest/multimedia_messaging/notification`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body is a nested JSON object containing the following parameters:

- **reference**. JSON object. Required. Use this object to provide the following information about the endpoint that is to receive the notification:
  - **correlator**: String. Required. The correlator used to identify the notification.
  - **endpoint**: String. Required. The URI which represents the endpoint address to which the notification should be delivered. This string should be a Bayeux protocol channel name that begins with **`/bayeux/`***`appInstanceID`* where *appInstanceID* is the client application's application instance account ID.

    For more information on managing application instances, see *Services Gatekeeper Portal Developer's Guide*.

  - **interfaceName**: String. Required. A descriptive string to identify the type of notification.
- **messageServiceActivationNumber**: String. Required. The number associated with the invoked message service, that is the destination address used to send the message
- **criteria**: String. Optional. The text to match against in determining whether or not the application should receive the notification:
  - **notifyMessageReception**
  - **notifyMessageDeliveryReceipt**

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "messageServiceActivationNumber": "URI",
  "reference": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "criteria": "String"
}
```

Response Header

The Location header field contains the URI of the publish/subscribe server:

```
http://host:port/rest/multimedia_messaging/notifications
```

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body. The appropriate notifications (described below) are sent to the **endpoint** address provided by the application in the request body of this operation.

Notification Data Object for Message Reception (notifyMessageReception)

When a multimedia message is successfully retrieved by the called party and the value for **criteria** in the request body is **notifyMessageReception**, Services Gatekeeper sends a nested JSON data object to the **endpoint** address.

This nested JSON object contains the following as the value of the attribute name **notifyMessageReception**:

- **correlator**: String. The correlator used to identify the notification.
- **message**. Nested JSON object. It contains the following:
  - **message**: String. The contents of the multimedia message.
  - **senderAddress**: String. The URI of the sender of the multimedia message.
  - **messageServiceActivationNumber**: String. The number associated with the invoked Message service, that is the destination address used to send the message.

– **dateTime**: String. The time at which the message was sent in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.

The notification data object delivered to the **endpoint** address when the criteria is **notifyMessageReception** is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"notifyMessageReception": {
  "correlator": "String",
  "message": {
    "messageServiceActivationNumber": "String",
    "priority": "Default|Low|Normal|High",
    "senderAddress": "URI",
    "dateTime": "Calendar",
    "message": "String",
    "messageIdentifier": "String",
    "subject": "String"
  }
}}
```

Notification Data Object for MMS Delivery Receipt (notifyMessageDeliveryReceipt)

When a multimedia message is successfully retrieved by the called party and the value for **criteria** in the request body is **notifyMessageDeliveryReceipt**, Services Gatekeeper sends a nested JSON data object to the **endpoint** address.

This nested JSON object contains the following as the value of the attribute name **notifyMessageDeliveryReceipt**:

- **correlator**: String. The correlator used to identify the notification.
- **deliveryStatus**: Enumeration value. Table 9–2 lists the possible statuses:

Table 9–2 Enumeration Values for Delivery Status

Value	Description
DeliveredToNetwork	Successful delivery to network. For concatenated messages, returned only when all the MMS parts have been successfully delivered to the network.
DeliveryUncertain	Delivery status unknown; for example, if it was handed off to another network.
DeliveryImpossible	Unsuccessful delivery; the message could not be delivered before it expired.
DeliveredToTerminal	Successful delivery to terminal. For concatenated messages, returned only when all the MMS parts have been successfully delivered to the terminal.
DeliveryNotificationNotSupported	Unable to provide delivery receipt notification.

The notification data object delivered to the **endpoint** address when the criteria is **notifyMessageDeliveryRecipt** is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"notifyMessageDeliveryReceipt": {
  "correlator": "String",
  "deliveryStatus": {
    "address": "URI",
    "deliveryStatus":
"DeliveredToTerminal|DeliveryUncertain|DeliveryImpossible|MessageWaiting|
```

```
DeliveredToNetwork|DeliveryNotificationNotSupported"
  }
}}
```

Stop Message Notification

The Stop Message Notification operation terminates a previously set up multimedia message notification for the application.

To stop a previously set up multimedia message notification, provide the correlator for the notification passed earlier in the Start Message Notification request.

There is no request or response body for the Stop Message Notification operation. If the request fails, the body of the error response will contain the identifier for the notification and the type of exception.

Authorization

Basic

HTTP Method

`DELETE`

URI

`http://`*`host`*`:`*`port`*`/rest/multimedia_messaging/notification/`*`correlator`*

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *correlator* is the correlator for the notification provided in the **reference** object of the initial Start Message Notification operation.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Examples

Example 9–3 Stop Message Notification Request

```
DELETE /rest/multimedia_messaging/notification/6789 HTTP/1.1
X-Session-ID: app:4130997928482260925
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
```

Example 9–4 Stop Message Notification Response

```
HTTP/1.1 204 No Content
Connection: close
Date: Thu, 04 Nov 2101 09:59:05 GMT
Content-Length: 0
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1
```

10

Adding RESTful Email Communication Service Support

This chapter describes the operations in the Email Communication interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Email Communication Interface

Applications use the RESTful Email Communication interface to send an email message and to fetch information on email messages that have been received for the applications and stored on Services Gatekeeper.

Applications use the interface to fetch those messages, get delivery status on sent messages, and start and stop a notification.

The actual message is sent as an attachment. See "Headers for Multipart Messages with Attachments" for more information.

For an application to be informed that a party has received an MMS message or that a message delivery receipt has been sent, the application includes an application **endpoint** address for such delivery notifications in the body of the request message. This address resides on a publish/subscribe server. See "RESTful Notifications and Publish/Subscribe" for information on publishing and subscribing to RESTful notifications.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://host:port/rest/multimedia_messaging/index.html`

where *host* and *port* are the host name and port of the Services Gatekeeper access tier server.

Send Message

The Send Message operation delivers an email message.

To send an email message, provide the URI of the addresses which the must receive the message in the request body. If there is to be a charge for the messaging, the request body should contain the required charging object. If the sender requires a delivery receipt, specify the required parameters for the receipt and the URI address of the sender. Also, a priority value, and a subject line can be provided in the request body.

The Plug-In Manager uses the **email:** prefix in the sender address and destination address to route the email message to the email plug-in instance.

If the Send Message operation is successful, the response header will contain the URI of the publish/subscribe server and the request identifier which is also provided in the response body for this operation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/multimedia_messaging/messages`

where *host* and *port* are the host name and port number of the Services Gatekeeper access tier.

Request Header

The request headers depend on the type of message:

- If the message content is in the form of an attachment, the request will be multipart and the request header must contain the header fields that describe the parts of the message.
- If the message does not contain an attachment, the special headers associated with the multipart messaged are not used.
- The X-Parameter **ContentInFirstAttachment** is added to indicate whether or not there is an email body. If this X-Parameter is set to **true** or is absent, then the first attachment contains the email body and remaining attachments are handled as regular attachments. If this X-Parameter is set to **false**, then all attachments are handled as regular attachments and the email body is empty.

See Example 10–1 for a listing of both types of headers.

Message Part Content

The message part content for the Send Message operation accepts the following parameters:

- **addresses**: Array of string values. Required. Use the **email:** prefix in the sender address and destination address to indicate to the Plug-in Manager that it is an email message intended for the email plug-in.
- **subject**: String. Optional. The text of the message.
- **priority**: String. Optional. The message priority specified as one of the following:
  - **Default**
  - **Low**
  - **Normal**
  - **High**
- **senderAddress**: String. Use the **email:** prefix.
- **receiptRequest**: A JSON object. Provide values for each of the following parameters which define this object:
  - **correlator**: String. Required. Used to correlate the receipt with the initial message.

    If the callback reference **correlator** is defined in the **receiptRequest** object of the Send Message request, Services Gatekeeper will invoke notifyMessage Delivery Receipt and discard the delivery status information.
  - **endpoint**: String. The endpoint address (URI) to which the receipt must be delivered. The address is a Bayeux protocol channel name that is the client application's application instance account ID.
  - **interfaceName**: String. Required. A description provided to identify the type of receipt.

The message part content for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "addresses": ["URI"],
  "priority": "Default|Low|Normal|High",
  "receiptRequest": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "senderAddress": "String",
  "subject": "String"
}
```

See Example 10–1 for a listing of the message part.

Response Header

The Location header field contains the URI of the publish/subscribe server as:

`http://host:port/rest/multimedia_messaging/delivery-status/result`

where, *result* is the string identifier for the request that is returned in the response body.

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The body of the response contains the request identifier returned in the **Location** header field of the response message as value for **result**.

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": "String"}
```

Examples

Example 10–1 Send Message Operation (Single Part MMS)

This example shows a POST method with a message.

```
POST /rest/multimedia_messaging/messages HTTP/1.1
X-Session-ID: app:-7603991555266551180
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys: ContentInFirstAttachment
X-Param-Values: false
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
Content-Length: 253
Content-Type: application/json

{

"addresses":["email:sam@example.com","email:john@example.com","email:tom@example.c
om"],
  "subject":"Meeting starts at 2 p.m.",
  "priority":null,
  "senderAddress":"email:bob@example.com",
  "charging":null
  "receiptRequest":
         {
           "correlator":"981234",
           "endpoint":"http://endpt_host:port/jaxws/MessageNotification",
           "interfaceName":"interfaceName",
         }
 }
```

Example 10–2 Send Message Operation (Multipart MMS)

This example shows a POST method with a multipart message. Note the use of header fields to describe the content and the boundary attribute to distinguish the message parts.

```
POST /rest/multimedia_messaging/messages HTTP/1.1
X-Session-ID: app: -123456789012346789
Authorization: Basic ZG9tY1uX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys: ContentInFirstAttachment
X-Param-Values: true
User-Agent: Jakarta Commons-HttpClient/3.0
Host: localhost:8001
Content-Length: 1215
Content-Type: multipart/form-data; boundary=kboiiFPAakDPYKeY7hBAW9I5c0rT48

--kboiiFPAakDPYKeY7hBAW9I5c0rT48
```

```
Content-Disposition: form-data; name="messagePart"
Content-Type: application/json; charset=US-ASCII
Content-Transfer-Encoding: 8bit

{

"addresses":["email:sam@example.com","email:john@example.com","email:tom@example.c
om"],
  "subject":"Hello World",
  "priority":null,
  "senderAddress":"email:bob@example.com",
  "charging":null,
  "receiptRequest":
         {
           "correlator":"981234",
           "endpoint":"http://endpt_host:port/jaxws/MessageNotification",
           "interfaceName":"interfaceName"
         }
 }
--kboiiFPAakDPYKeY7hBAW9I5c0rT48
Content-Disposition: form-data; name="Attachment-txt-1"
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 8bit

Oracle Communications Services Gatekeeper is a powerful, flexible, secure
interface which provides a simple way for application developers to include
telephony-based functionality in their software applications and guarantee the
security, stability, and performance required by network operators and demanded by
their subscribers.
--kboiiFPAakDPYKeY7hBAW9I5c0rT48
```

Get Received Messages

The Get Received Messages operation retrieves an array of network-triggered messages that have arrived at Services Gatekeeper.

The request body for Get Received Messages contains the registration identifier necessary to retrieve the required email messages intended for the application and, optionally, it may also contain a priority for the retrieval. The registration identifier should have been set up with the off-line provisioning step completed earlier.

If Get Received Messages is successful, the response body will contain the data about the message. If the content of the message is pure ASCII, the response body contains the message. Otherwise the response body contains an identifier that is used to fetch the actual message.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/multimedia_messaging/receive-messages`

where *host* and *port* are the host name and port number of the Services Gatekeeper access tier.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Get Received Messages operation accepts the following parameters:

- **registrationIdentifier**: String. Required. The value previously set up to enable the application to receive notification of email reception according to specified criteria.
- **priority**: String. Optional. The message priority specified as one of the following:
  - **Default**
  - **Low**
  - **Normal**
  - **High**

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "registrationIdentifier": "String",
  "priority": "Default|Low|Normal|High"
}
```

Response header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains an array of structures as the value for **result**. Each element in the array contains values for the following parameters.

- **messageServiceActivationNumber**: String. The number associated with the invoked message service, that is the destination address used to send the message.
- **priority**: String. The message priority value from the request.
- **senderAddress**: String.
- **dateTime**: String. The start time for the call in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.
- **message**: String. Present if the text of the message is purely ASCII text.
- **messageIdentifier**: String. An identifier used to fetch the message, when it is not pure ASCII text. This value is used in the URI of the request for the "Get Message" operation.
- **subject**: String. The subject line of the email message.

The response body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"result": [{
  "messageServiceActivationNumber": "String",
  "priority": "Default|Low|Normal|High",
  "senderAddress": "URI",
  "dateTime": "Calendar",
  "message": "String",
  "messageIdentifier": "String",
  "subject": "String"
}]}
```

Get Message

The Get Message operation fetches a network-triggered message for the application from Services Gatekeeper.

To get a message, provide the appropriate message identifier in the URI for the GET method.

The actual message is returned as an attachment.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/multimedia_messaging/message/${messageRefIdentifier}`

where:

- *host* and *port* are the host name and port number of the Services Gatekeeper access tier.
- *messageRefIdentifier* is the value for *messageIdentifier* in:
  - the response body of the "Get Received Messages" operation
  - **notifyMessageReception** data object. See "Notification Data Object for Message Reception (notifyMessageReception)"

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

The MIME-type for the Content-Type header field is **multipart/mixed**.

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Get Message Delivery Status

The Get Message Delivery Status operation retrieves the delivery status of a previously sent message.

This operation is valid only if the **correlator** callback reference was not defined in the **receiptRequest** object of the initial request to send the email message. In such a scenario, Services Gatekeeper stores the delivery status for a configurable period.

If the callback reference correlator is defined in the **receiptRequest** object of the initial send email request, Services Gatekeeper will invoke the Notify Message Delivery Receipt operation and discard the delivery status information.

The request header for the Get Message Delivery Status operation contains the request identifier from **result** object of the initial request.

If Get Message Delivery Status operation is successful, the response body will contain the URI from the address field in the Send Message request and corresponding delivery status for the message.

Authorization

Basic

HTTP Method

`GET`

URI

`http://`*`host`*`:`*`port`*`/rest/emailmultimedia_messaging/delivery_status/${requestIdentifier}`

where:

- *host* and *port* are the host name and port number of the Services Gatekeeper access tier.
- *requestIdentifier* is a string returned in the **result** object of the initial Send Message request.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains an array of structures as the value for **result**. Each element in the array contains values for the following parameters.

- **address**: String. The URI to which the initial message was sent.
- **deliveryStatus**: Enumeration value. Table 10–1 lists the possible status:

Table 10–1 Enumeration Values for Delivery Status

Value	Description
DeliveredToNetwork	Successful delivery to network. For concatenated messages, returned only when all the email-parts have been successfully delivered to the network.
DeliveryUncertain	Delivery status unknown; for example, if it was handed off to another network.
DeliveryImpossible	Unsuccessful delivery; the message could not be delivered before it expired.
MessageWaiting	The message is still queued for delivery. This is a temporary state, pending transition to one of the preceding states.
DeliveryNotificationNotSupported	Unable to provide delivery receipt notification.

The response body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"result": [{
  "address": "URI",
  "deliveryStatus":
"DeliveredToTerminal|DeliveryUncertain|DeliveryImpossible|MessageWaiting|Delivered
ToNetwork"
}]}
```

Start Message Notification

The Start Message Notification operation starts a notification for the application.

To set up an email message notification, provide the criteria which will trigger notifications and a reference object for the delivery of the notifications. The criteria can be a string which, when matched, could be the notification of an email received or of a delivery receipt. The reference object (also a JSON object) contains the correlator for the notification, the endpoint address (a specific Bayeux channel name) to which the call direction notifications must be sent and, optionally, the interface name (a string to identify the notification).

If the Start Message Notification request is successful:

- The response header will contain the URI of the publish/subscribe server.
- A data object associated with the result of the short message operation will be sent to the endpoint address specified in the request body. This data object will contain the appropriate notification (that the message was received or a delivery receipt for the call).

When the application receives such a response, it must access the endpoint address to retrieve the specific call event notifications.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://`*`host`*`:`*`port`*`/rest/multimedia_messaging/notification`

where *host* and *port* are the host name and port number of the Services Gatekeeper access tier.

Request Header

The request header depends on the type of message:

- The MIME-type for the Content-Type header field is **application/json**.
- The X-Parameter **Password** to indicate the credential of the email service activation number. The value should be encrypted by AES (Advanced Encryption Standard) or 3DES (Triple Data Encryption Standard) algorithm.
- The X-Parameter **SizeLimit** to indicate the maximum total size (in kilobyte) of an email message attachment accepted by Services Gatekeeper.

Request Body

The request body is a nested JSON object containing the following parameters:

- **reference**. JSON object. Required. Use this object to provide the following information about the endpoint that is to receive the notification:
  - **correlator**: String. Required. The correlator used to identify the notification.

  - **endpoint**: String. Required. The URI which represents the endpoint address to which the notification should be delivered. This string should be a Bayeux protocol channel name that begins with **`/bayeux/`***`appInstanceID`* where *appInstanceID* is the client application's application instance account ID.

    For more information on application instances, see *Services Gatekeeper Portal Developer's Guide*.

  - **interfaceName**: String. Required. A descriptive string to identify the type of notification.

- **messageServiceActivationNumber**: String. Required. The number associated with the invoked message service, that is the destination address used to send the message
- **sizeLimit**: Integer. The maximum size for the email allowed in Services Gatekeeper.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "messageServiceActivationNumber": "URI",
  "reference": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "sizeLimit": "Integer"
}
```

Response Header

The Location header field contains the URI of the publish/subscribe server:

`http://`*`host`*`:`*`port`*`/rest/multimedia_messaging/notifications`

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body. The appropriate notifications (described below) are sent to the **endpoint** address provided by the application in the request body of this operation.

Notification Data Object for Message Reception (notifyMessageReception)

When an email message is successfully retrieved by the called party and the value for **criteria** in the request body is **notifyMessageReception**, Services Gatekeeper sends a nested JSON data object to the **endpoint** address.

This nested JSON object contains the following as the value of the attribute name **notifyMessageReception**:

- **correlator**: String. The correlator used to identify the notification.
- **message**. Nested JSON object. It contains the following:
  - **message**: String. The contents of the email message.
  - **senderAddress**: String.

- **messageServiceActivationNumber**: String. The number associated with the invoked Message service, that is the destination address used to send the message.
- **dateTime**: String. The time at which the message was sent in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.

The notification data object delivered to the **endpoint** address when the criteria is **notifyMessageReception** is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"notifyMessageReception": {
  "correlator": "String",
  "message": {
    "messageServiceActivationNumber": "String",
    "priority": "Default|Low|Normal|High",
    "senderAddress": "URI",
    "dateTime": "Calendar",
    "message": "String",
    "messageIdentifier": "String",
    "subject": "String"
  }
}}
```

Notification Data Object for Email Delivery Receipt (notifyMessageDeliveryReceipt)

When an email message is successfully retrieved by the called party and the value for **criteria** in the request body is **notifyMessageDeliveryReceipt**, Services Gatekeeper sends a nested JSON data object to the **endpoint** address.

This nested JSON object contains the following as the value of the attribute name **notifyMessageDeliveryReceipt**:

- **correlator**: String. The correlator used to identify the notification.
- **deliveryStatus**. Enumeration value. Table 10–2 lists the possible statuses:

Table 10–2 Enumeration Values for Delivery Status

Value	Description
DeliveredToNetwork	Successful delivery to network. For concatenated messages, returned only when all the email-parts have been successfully delivered to the network.
DeliveryUncertain	Delivery status unknown; for example, if it was handed off to another network.
DeliveryImpossible	Unsuccessful delivery; the message could not be delivered before it expired.
DeliveredToTerminal	Successful delivery to terminal. For concatenated messages, returned only when all the email-parts have been successfully delivered to the terminal.
DeliveryNotificationNotSupported	Unable to provide delivery receipt notification.

The notification data object delivered to the **endpoint** address when the criteria is **notifyMessageDeliveryReceipt** is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"notifyMessageDeliveryReceipt": {
  "correlator": "String",
```

```
  "deliveryStatus": {
    "address": "URI",
    "deliveryStatus":
"DeliveredToTerminal|DeliveryUncertain|DeliveryImpossible|MessageWaiting|
DeliveredToNetwork"
  }
}}
```

Stop Message Notification

The Stop Message Notification operation terminates a previously set up email message notification for the application.

To stop a previously set up email notification, provide the correlator for the notification passed earlier in the Start Message Notification request.

There is no request or response body for the Stop Message Notification operation. If the request fails, the body of the error response will contain the identifier for the notification and the type of exception.

Authorization

Basic

HTTP Method

`DELETE`

URI

`http://`*`host`*`:`*`port`*`/rest/multimedia_messaging/notification/`*`correlator`*

where:

- where *host* and *port* are the host and port number of the Services Gatekeeper access tier.
- *correlator* is the correlator for the notification provided in the **reference** object of the initial Start Message Notification operation.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Examples

Example 10–3 Stop Message Notification Request

```
DELETE /rest/multimedia_messaging/notification/6789 HTTP/1.1
X-Session-ID: app:4130997928482260925
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
```

Example 10–4 Stop Message Notification Response

```
HTTP/1.1 204 No Content
Connection: close
Date: Thu, 04 Nov 2101 09:59:05 GMT
Content-Length: 0
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1
```

11

Adding RESTful Terminal Location Support

This chapter describes the operations in the Terminal Location interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Terminal Location Interface

Applications use the RESTful Terminal Location interface to get a location for an individual terminal or a group of terminals; to get the distance of the terminal from a specific location; and to start and stop notifications, based on geographic location or on a periodic interval.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://host:port/rest/terminal_location/index.html`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Get Location

The Get Location operation retrieves the location of a single terminal. The interface will return an error if the query object contains more than one address.

To retrieve the location of a specific terminal, provide its URI as the address value of the query object in the Request-URI of the GET method.

If the Get Location operation is successful, the response body contains a nested JSON data object containing the physical coordinates of each terminal and the date and time for when such data was last collected.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/terminal_location/location?${query}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${query}* is a nested JSON data object.

The parameters accepted by *query* are:

- **acceptableAccuracy**: Integer. Required. The range that the application considers useful. If the location cannot be determined within this range, the application would prefer not to receive this information.
- **address**: String. Required. The address of the terminal whose location is required, as a URI.
- **requestedAccuracy:** Integer. Required. The range over which the application wants to receive location information. This may influence the choice of location technology to use (for instance, cell sector location may be suitable for requests specifying 1000 meters, but GPS technology may be required for requests below 100 meters).
- **tolerance:** String. Required. Enumerated value denoting the priority of response time versus accuracy. Table 11–1 lists the possible values:

Table 11–1 Enumeration Values for Tolerance Attribute

Value	Description
NoDelay	The server should immediately return any location estimate that it currently has. If no estimate is available, the server returns a failure indication. The server may optionally initiate procedures to obtain a location estimate (for example, to be available for a later request).

Table 11–1 (Cont.) Enumeration Values for Tolerance Attribute

Value	Description
LowDelay	The response time is more important than the requested accuracy. The server attempts to fulfil any accuracy requirement, but not if it adds delay. A quick response with lower accuracy is more desirable than waiting for a more accurate response.
DelayTolerant	The network is expected to return a location with the requested accuracy even if this means not complying with the requested response time.

- **maximumAge**: JSON object. Optional. The maximum acceptable age of the location information. This object is defined by the following:
  - **metric**: String. The unit of time. Required if the **maximumAge** object is present in the body of the query. Possible entries are **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month** or **Year**.
  - **units**: Integer. Required if the **maximumAge** object is present in the body of the query. The number of units for the metric.
- **responseTime**: JSON object. Optional. The maximum response time that the application will accept. This object is defined by the following:
  - **metric**: String. The unit of time. Required if the **responseTime** object is present in the body of the query. Possible entries are **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month**, or **Year**.
  - **units**: Integer. Required if the **maximumAge** object is present in the body of the query. The number of units for the metric.

The maximum duration and interval is 8916039 seconds, which converts to 99 days, 99 hours, 99 minutes, and 99 seconds. Do not enter a value that exceeds the 8916039 second interval. In other words, do not use the **Year** time metric, and only about three **Month**s is allowed.

The *${query}* object in the URI is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "acceptableAccuracy": "Integer",
  "address": "URI",
  "requestedAccuracy": "Integer",
  "tolerance": "NoDelay|LowDelay|DelayTolerant",
  "maximumAge": {
    "metric": "Millisecond|Second|Minute|Hour|Day|Week|Month|Year",
    "units": "Integer"
  },
  "responseTime": {
    "metric": "Millisecond|Second|Minute|Hour|Day|Week|Month|Year",
    "units": "Integer"
  }
}
```

Example 11–1 Request URI for Get Location

```
GET /rest/terminal_
location/location?query=%7B%22aceptableAccuracy%22%3A%22100%%22%2C%22address%22%3
A%22tel%3A1234%22%2C%22requestAccuracy%22%3A%22100%%22%2C
%22tolerance%22%3A%22NoDelay%22%2C%22maximumAge%22%3Anull%2C%22responseTime%22%3
```

```
Anull%7D HTTP/1.1
```

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The location of the specific terminal is returned in the body of the response as the value of the **result** JSON object. The parameters in this object are:

- **accuracy**: Integer.
- **latitude**: Number (floating point).
- **longitude**: Number (floating point).
- **timestamp**: String. The date and time when the terminal's geographical coordinates were collected, given in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.
- **altitude**: Number (floating point).

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": {
  "accuracy": "Integer",
  "latitude": "Float",
  "longitude": "Float",
  "timestamp": "Calendar",
  "altitude": "Float"
}}
```

Example 11–2 Response Body for Get Location

```
{
   "result":
       {
        "accuracy":"10",
        "latitude":"37.78843",
        "longitude":"-122.4374",
        "timestamp":"2010-11-05T21:56:21+08:00"
        }
}
```

Examples

Example 11–3 Get Location Request

```
GET /rest/terminal_
```

```
location/location?query=%7B%22aceptableAccuracy%22%3A%22100%%22%2C%22address%22%3
A%22tel%3A1234%22%2C%22requestAccuracy%22%3A%22100%%22%2C
%22tolerance%22%3A%22NoDelay%22%2C%22maximumAge%22%3Anull%2C%22responseTime%22%3
Anull%7D HTTP/1.1
X-Session-ID: app:5198750923966743997
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
```

Example 11–4 Get Location Response

```
HTTP/1.1 200 OK
Date: Thu, 05 Nov 2101 05:52:41 GMT
Content-Length: 131
Content-Type: application/json
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1

{
   "result":
       {
        "accuracy":"10",
        "latitude":"37.78843",
        "longitude":"-122.4374",
        "timestamp":"2010-11-05T21:56:21+08:00"
        }
}
```

Get Location for Group

The Get Location for Group operation retrieves the location information for a group of terminals.

To retrieve the location information of a specific terminals, provide their URIs as the address values of the query object in the Request-URI of the GET method.

If the Get Location for Group operation is successful, the response body contains a JSON data object indicating the physical location for each terminal (whether or not the specific terminal is reachable, unreachable or busy).

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/terminal_location/location?queryForGroup=${queryForGroup}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${queryForGroup}* is a nested JSON data object.

The parameters accepted by *queryForGroup* are:

- **acceptableAccuracy**: Integer. Required. The range that the application considers useful. If the location cannot be determined within this range, the application would prefer not to receive this information.
- **addresses**: Array of Strings. Required. The addresses of the terminals whose location is required, as URIs.
- **requestedAccuracy:** Integer. Required. The range over which the application wishes to receive location information. This may influence the choice of location technology to use (for instance, cell sector location may be suitable for requests specifying 1000 meters, but GPS technology may be required for requests below 100 meters).
- **tolerance:** String. Required. Enumerated value denoting the priority of response time versus accuracy. Table 11–2 lists the possible values:

Table 11–2 Enumeration Values for Tolerance Attribute

Value	Description
NoDelay	The server should immediately return any location estimate that it currently has. If no estimate is available, the server return a failure indication. The server may optionally initiate procedures to obtain a location estimate (for example, to be available for a later request).

Table 11–2 (Cont.) Enumeration Values for Tolerance Attribute

Value	Description
LowDelay	The response time is more important than the requested accuracy. The server attempts to fulfil any accuracy requirement, but not if it adds delay. A quick response with lower accuracy is more desirable than waiting for a more accurate response.
DelayTolerant	The network is expected to return a location with the requested accuracy even if this means not complying with the requested response time.

- **maximumAge**: JSON object. Optional. The maximum acceptable age of the location information. This object is defined by the following:
  - **metric**: String. The unit of time. Required if the **maximumAge** object is present in the body of the query. Possible entries are **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month**, or **Year**.
  - **units**: Integer. Required if the **maximumAge** object is present in the body of the query. The number of units for the metric.
- **responseTime**: JSON object. Optional. The maximum response time that the application will accept. This object is defined by the following:
  - **metric**: String. The unit of time. Required if the **responseTime** object is present in the body of the query. Possible entries are **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month**, or **Year**.
  - **units**: Integer. Required if the **maximumAge** object is present in the body of the query. The number of units for the metric.

The *${queryForGroup}* object in the URI is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "acceptableAccuracy": "Integer",
  "addresses": ["URI"],
  "requestedAccuracy": "Integer",
  "tolerance": "NoDelay|LowDelay|DelayTolerant",
  "maximumAge": {
    "metric": "Millisecond|Second|Minute|Hour|Day|Week|Month|Year",
    "units": "Integer"
  },
  "responseTime": {
    "metric": "Millisecond|Second|Minute|Hour|Day|Week|Month|Year",
    "units": "Integer"
  }
}
```

Example 11–5 Request URI for Get Location for Group

```
GET /rest/terminal_
location/location?queryForGroup=%7B%22aceptableAccuracy%22%3A%22100%%22%2C%22
addresses%22%3A%%5B22tel%3A1234%22%2C%22tel%3A123%22%5D%2C%22requestAccuracy%22
%3A%22100%%22%2C%22tolerance%22%3A%22NoDelay%22%2C%22maximumAge%22%3Anull%2C%22
responseTime%22%3Anull%7D HTTP/1.1
```

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body is the attribute **result** whose value consists of an array of JSON objects. Each element in the array contain the report status, the location information (if any) and the error information (if any) for the terminals whose addresses were specified in the Request-URI for **status?queryForGroup**.

The following parameters make up the information associated with an individual terminal's location:

- **address**: String. The address of a terminal whose location is required, as a URI.
- **reportStatus**: String. The status of the terminal. It can be one of the following:
  - **Retrieved**: The terminal's location information is available. It is provided in this object as the current location of the terminal.
  - **Not Retrieved**: The terminal's location information is not available.
  - **Error**: There was an error in the attempt to get the location information for this terminal. The error data is provided in this object.
- **currentLocation**: String. This parameter will be present if the value for **reportStatus** is **Retrieved**. The current location of the terminal as one of the following:
  - **accuracy**: Integer.
  - **latitude**: Number (floating point).
  - **longitude**: Number (floating point).
  - **altitude**: Number (floating point).
  - **timestamp**: String. The date and time when the terminal's geographical coordinates were collected, in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.
- **errorInformation**: JSON object. This object will be present if the value for `reportStatus` is `Error`. It will contain the following error information about the terminal:
  - **messageID**: String. The error message ID.
  - **text**: String. The text for the error message.
  - **variables**: Array of string values. Variables to substitute into text strings.

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": [{
```

```
  "address": "URI",
  "reportStatus": "Retrieved|NotRetrieved|Error",
  "currentLocation": {
    "accuracy": "Integer",
    "latitude": "Float",
    "longitude": "Float",
    "timestamp": "Calendar",
    "altitude": "Float"
  },
  "errorInformation": {
    "messageId": "String",
    "text": "String",
    "variables": ["String"]
  }
}]}
```

Examples

Example 11–6 Get Location for Group Request

```
GET /rest/terminal_
location/location?queryForGroup=%7B%22aceptableAccuracy%22%3A%22100%%22%2C%22
addresses%22%3A%%5B22tel%3A1234%22%2C%22tel%3A123%22%5D%2C%22requestAccuracy%22
%3A%22100%%22%2C%22tolerance%22%3A%22NoDelay%22%2C%22maximumAge%22%3Anull%2C%22
responseTime%22%3Anull%7D HTTP/1.1
X-Session-ID: app:4130997928482260925
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
```

Example 11–7 Get Location for Group Response

```
HTTP/1.1 200 OK
Date: Thu, 04 Nov 2101 08:12:43 GMT
Content-Length: 438
Content-Type: application/json
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1

{"result":
       [{"address":"tel:1234",
           "errorInformation":null,
           "reportStatus":"Retrieved",
           "currentLocation":
                    {
                     "accuracy":"10",
                     "latitude":"37.78843",
                     "longitude":"-122.4374",
                     "altitude":"0.0"
                     "timestamp":"2010-11-05T22:13:28+08:00"
          },
          {"address":"tel:123",
           "errorInformation":null,
           "reportStatus":"Retrieved",
           "currentLocation":
```

```
                                {
                                 "accuracy":"10",
                                 "latitude":"55.2776",
                                 "longitude":"7.012778",
                                 "altitude":"20.0"
                                 "timestamp":"2010-11-05T22:13:28+08:00"
                        }
                }
```

Get Terminal Distance

The Get Terminal Distance operation retrieves the distance between a specified terminal and a required location. The terminal distance is calculated in meters.

To retrieve the distance, provide the URI for the terminal and the geographic coordinates of the required location in the Request-URI of the GET method.

If the Get Terminal Distance operation is successful, the response body contains a JSON data object indicating the distance between a specified terminal and a required location, in meters.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/terminal_distance/distance?query=${query}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${query}* is a nested JSON data object.

The parameters accepted by the *query* attribute are:

- **address**: String. Required. The address of the terminal, as a URI.
- **latitude:** Number (floating point). Required. The latitude of the location.
- **longitude:** Number (floating point). Required. The longitude of the location

The *${query}* object in the URI is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "address": "URI",
  "latitude": "Float",
  "longitude": "Float"
}
```

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The distance between the specified terminal and the required location is returned as the value for **result** in a data object. The unit for the distance is meters and the value is an integer.

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": "String"}
```

Examples

Example 11–8 Get Terminal DIstance Request

```
GET /rest/terminal_
distance/distance?query=%7B%22address%22%3A%22tel%3A1234%22%2C%22longitude%22
%3A%2237.7707%%22%2C %22latittude%22%3A%22122.4177%227D HTTP/1.1
X-Session-ID: app:5198750923966743997
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
```

Example 11–9 Get Terminal Distance Response

```
HTTP/1.1 200 OK
Date: Thu, 05 Nov 2101 05:52:41 GMT
Content-Length: 131
Content-Type: application/json
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1

{
   "result":"9316351"
}
```

Start Geographical Notification

The Start Geographical Notification operation provides notifications based on whether terminals enter or leave a specified geographic area. The area to be monitored is circular with specified latitude and longitude as the center and having a specified radius.

To set up for such notifications, provide the URI of the terminal addresses for which the application must receive notifications, the criteria which will trigger notifications and a reference object for the delivery of the notifications. Additionally, you can specify the request frequency, the total number of notifications and the duration for the notification and whether or not the check must start immediately or not. The reference object (also a JSON object) contains the correlator for the notification, the endpoint address (a specific Bayeux channel name) to which the notifications must be sent and, optionally, the interface name (a string to identify the notification).

If the request for the Start Geographical Notification operation is successful, the endpoint address specified in the request body will receive a notification when:

- The terminal location has been successfully retrieved.
- The notification limit or the specified duration has been reached.
- An error has been encountered in obtaining the location of the terminal.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://host:port/rest/terminal_location/geographical-notification`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Start Geographical Notification operation is a nested JSON object. It accepts an array of addresses for the terminals in whose geographical information the application is interested, and the parameters for the check:

- **addresses**: Array of string values. Required. Each element in the array is a address for a terminal, as a URI.
- **checkImmediate**: String. Required. Set to:
  - **True**: The application requires the geographical notification to start immediately and as often as required thereafter.
  - **False**: The application requires the geographical notification to start at the end of the time period.

- **count**: Integer. Optional. The maximum number of notifications sent to the application. If this number is reached while notification operation is active a **locationEnd** notification is delivered to the endpoint address. If you omit this option, or set it to **0** there is no limit to the number of notifications.
- **criteria**: String. Required. The status of the terminal. It can be one of the following:
  - **Entering**: The notification should be made when the terminal enters the area being monitored.
  - **Leaving**: The notification should be made when the terminal leaves the area being monitored.
- **duration**: JSON object. Optional. Specifies a time limit for notifications. The maximum duration period can be limited by the underlying network. Omit this option, or set it to **0** to specify no time limit.
  - **metric**: String. Required. The unit of time for the duration specified as **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month** or **Year**.
  - **units**: Integer. Required. The number of **metric** units that specify the duration. If this number is reached (and the application has not ended the notification operation), the **locationEnd** notification is delivered to the **endpoint** address.
- **frequency**: JSON object. Required. Specifies the time between notifications. The value must be in the range of 1-8916039 seconds, which converts to 99 days, 99 hours, 99 minutes, and 99 seconds. Do not enter a value that exceeds the 8916039 second interval. In other words, do not use the **Year** time metric, and only about three **Month**s is allowed.
  - **metric**: String. Required. The unit of time for the notification frequency. Can be specified in **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month**, or **Year**.
  - **units**: Integer. Required. The number of **metric** units between notifications.

  For example this entry:

  ```
  metric=second
  unik=10
  ```

  specifies a minimum time between notifications of 10 seconds. However this entry:

  ```
  metric=month
  unit=6
  ```

  is invalid because it exceeds the 8916039 second (99 day) limit.
- **latitude**: Number (floating point). Required. The latitude of the location which will be the center of the area under surveillance.
- **longitude**: Number (floating point). Required. The longitude of the location which will be the center of the area under surveillance.
- **radius**: Number (floating point) in meters. Required. The radius of the circle around the location (center point), in meters.
- **reference**: JSON object. Required. Use this object to provide the following information about the endpoint that is to receive the notification:
  - **correlator**: String. Required. The correlator used to identify the notification.
  - **endpoint**: String. Required. The URI which represents the endpoint address to which the notification should be delivered. This string should be a Bayeux protocol channel name that begins with **`/bayeux/`***`appInstanceID`* where *appInstanceID* is the client application's application instance account ID.

For more information on managing application instances, see *Services Gatekeeper Portal Developer's Guide*.

  - **interfaceName**: String. Required. A descriptive string to identify the type of notification.

- **trackingAccuracy**: Number (floating point). Required. The acceptable error in the tracking, in meters.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "addresses": ["URI"],
  "checkImmediate": "Boolean",
  "criteria": "Entering|Leaving",
  "frequency": {
    "metric": "Millisecond|Second|Minute|Hour|Day|Week|Month|Year",
    "units": "Integer"
  },
  "latitude": "Float",
  "longitude": "Float",
  "radius": "Float",
  "reference": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "trackingAccuracy": "Float",
  "count": "Integer",
  "duration": {
    "metric": "Millisecond|Second|Minute|Hour|Day|Week|Month|Year",
    "units": "Integer"
  }
}
```

Response Header

The Location header field contains the URI of the publish/subscribe server:

`http://host:port/rest/terminal_location/notifications`

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body. The appropriate notifications (described below) are sent to the **endpoint** address provided by the application in the request body of this operation.

Notification When Terminal Location is Successfully Retrieved (locationNotification)

When there is a match for the **criteria** value (specified in the request body) and the terminal's location is successfully retrieved, Services Gatekeeper sends a nested JSON data object to the **endpoint** address.

This nested JSON object contains the following as the value of the attribute name **locationNotification**:

- **correlator**: String. The correlator used to identify the notification and provided in the request body of the Start Geographical Notification operation.
- **criteria**: String. The state of the terminal. It can be one of the following:
  - **Entering**: The terminal entered the area being monitored.
  - **Leaving**: The terminal left the area being monitored.
- **data**: Array of JSON objects. Each element in the array contains the current location information of a terminal being monitored and contains the following parameters:
  - **address**: String. The URI of the terminal being monitored.
  - **reportStatus**: String. Retrieval status for this terminal which can be **Retrieved**, **NotRetrieved**, or **Error**. This allows for partial reports to avoid timeouts, and so on.
  - **currentLocation**: String. This object will be present only if the **reportStatus** is **Retrieved**. It contains the location information for the terminal, as of the date and time specified in this object. Table 11–3 lists the attributes in this object:

Table 11–3 Attributes in the currentLocation JSON Object

Attribute	Description
accuracy	Number (floating point). The accuracy error in arriving at the terminal's location, in meters.
latitude	Number (floating point). The latitude for the terminal.
longitude	Number (floating point). The longitude for the terminal.
timestamp	String. The date and time when the terminal's geographical coordinates were collected, given in ISO 9601's extended format.
altitude	Number (floating point). The altitude for the terminal.

  - **errorInformation**: JSON object. This object will be present if the value for **reportStatus** is **Error**. Table 11–4 lists the attributes in this object.

Table 11–4 Attributes in the errorInformation Object

Attribute	Description
messageID	String. Message identifier for the fault
text	String. The text of the message. If this string contains replacement variables, the variables entry hold the
variables	Array of string values. Optional. An array of variables to substitute into text strings.

The location notification data object delivered to the **endpoint** address is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"locationNotification": {
  "correlator": "String",
  "data": [{
    "address": "URI",
    "reportStatus": "Retrieved|NotRetrieved|Error",
    "currentLocation": {
      "accuracy": "Integer",
      "latitude": "Float",
```

```
      "longitude": "Float",
      "timestamp": "Calendar",
      "altitude": "Float"
    },
    "errorInformation": {
      "messageId": "String",
      "text": "String",
      "variables": ["String"]
    }
  }],
  "criteria": "Entering|Leaving"
}}
```

Example 11–10 locationNotification Object Delivered to the Application

```
{
  "locationNotification":
            {
              "correlator":"6789",
              "data":
                  [{
                     "address":"tel:123",
                     "reportStatus":"Retrieved",
                     "errorInformation":null,
                     "currentLocation":
                            {
                              "accuracy":"8",
                              "latitude":"37.80",
                              "longitude":"-122.56",
                              "altitude":"90",
                              "timestamp":"2010-11-08T10:29:38"
                            }
                  }],
              "criteria":"Entering"
            }
}
```

Notification of Error in Retrieving Terminal Location (locationError)

When there is an error in retrieving the location for the specified terminal, Services Gatekeeper sends a nested JSON data object to the **endpoint** address.

The nested JSON object contains the following as the value of the attribute name **locationError**:

- **correlator**: String. The correlator used to identify the notification and provided in the request body of the Status Notification operation.
- **reason**: JSON object. The explanation of the error specified by the following:
  - **messageID**: String. The error message identifier.
  - **text**: String. The error message description.
  - **variables**. An array of string values. The array of variables to substitute into text strings.
- **address**: String. The URI of a terminal to monitor.

The location error notification data object delivered to the **endpoint** address is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"locationError": {
  "correlator": "String",
  "reason": {
    "messageId": "String",
    "text": "String",
    "variables": ["String"]
  },
  "address": "URI"
}}
```

Notification Signalling End to Monitoring Terminal's Location (locationEnd)

When the notification count specified as the value for **count** is reached or when the duration specified for `units` in the **duration** object is reached, Services Gatekeeper sends a nested JSON data object to the **endpoint** address.

The JSON object contains the following as the value of the attribute name **locationEnd**:

- **correlator**: String. The correlator used to identify the notification and provided in the request body of the Status Notification operation.

Here is the structure of this notification:

```
{"locationEnd": {"correlator": "String"}}
```

> **Note:** This notification is not delivered in the case of an error, or if the application ended the notification using **endNotification**.

Examples

Example 11–11 Start Geographic Notification Request

```
PUT /rest/terminal_location/geographical-notification HTTP/1.1
X-Session-ID: app:4130997928482260925
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
Content-Length: 366
Content-Type: application/json

{
   "addresses":["tel:123"],
   "checkImmediate:"true",
   "criteria":"Entering",
   "frequency":
        {
          "metric": "Second",
          "units": "5"
        },
   "latitude":"37.7707",
   "longitude":"-122.4177",
   "radius":"5000.0",
```

```
   "reference":
        {
          "correlator":"6789",
          "endpoint":"bayeux/app_instance_1/tl",
          "interfaceName":"interfaceName"
        },
   "trackingAccuracy":"10.0",
   "count":"5",
   "duration":
        {
          "metric":"Minutes",
          "units":"30"
        }
}
```

Example 11–12 Status Notification Response

```
HTTP/1.1 200 Created
Date: Fri, 05 Nov 2101 09:59:05 GMT
Location: http://terminalloc_host:port/rest/terminal_location/notifications
Content-Length: 0
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1
```

Start Periodic Notification

The Start Periodic Notification operation provides notifications for the locations of a set of terminals at a defined interval.

To set up for such location notifications, provide the URI of the terminal addresses for which the application must receive notifications, the request frequency, the total number of notifications and the duration for the notification and whether or not the check must start immediately. The reference object (also a JSON object) contains the correlator for the notification, the endpoint address (a specific Bayeux channel name) to which the notifications must be sent and, optionally, the interface name (a string to identify the notification).

If the request for the Start Periodic Notification operation is successful, the endpoint address specified in the request body will receive a notification when:

- The terminal location has been successfully retrieved.
- The notification limit or the specified duration has been reached.
- An error has been encountered in obtaining the location of the terminal.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://host:port/rest/terminal_location/periodic-notification`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Start Periodic Notification operation is a nested JSON object. It contains an array of addresses for the terminals in whose information the application is interested, and the parameters for the check provided by the following:

- **addresses**: Array of string values. Required. Each element in the array is a address for a terminal, as a URI.
- **duration**: JSON object. Optional. Specifies the total time for receiving notifications The underlying network sometimes limits this count. Omit this setting or set it to **0** to specify no limit.
  - **metric**: String. Required. The unit of time for the notifications specified as **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month**, or **Year**.
  - **units**: Integer. Required. The number of **metric** units that specify the duration. If this number is reached while this operation is still active, the **locationEnd** notification is delivered to the **endpoint** address.

- **frequency**: JSON object. Required. Specifies a minimum time between notifications:
  - **metric**: String. Required. The unit of time for the frequency specified as **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month**, or **Year**.
  - **units**: Integer. Required. The frequency of the specified **metric**.
  - **metric**: String. Required. The unit of time for the frequency specified as **Millisecond**, **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month**, or **Year**.
  - **units**: Integer. Required. The frequency of the specified **metric**.

  This value must be in the range of 1-8916039 seconds, which converts to 99 days, 99 hours, 99 minutes, and 99 seconds. Do not enter a value that exceeds the 8916039 second interval. In other words, do not use the **Year** time metric, and only about three **Month**s is allowed.
- **reference**: JSON object. Required. Use this object to provide the following information about the endpoint that is to receive the notification:
  - **correlator**: String. Required. The correlator identifies the notification.
  - **endpoint**: String. Required. The URI which represents the endpoint address to which the notification should be delivered. This string should be a Bayeux protocol channel name that begins with **`/bayeux/`***`appInstanceID`* where *appInstanceID* is the client application's application instance account ID.

    For more information on managing application instances, see *Services Gatekeeper Portal Developer's Guide*.
  - **interfaceName**: String. Required. A descriptive string to identify the type of notification.
- **requestedAccuracy**: Number (floating point). Required. The acceptable error in the tracking, in meters.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "addresses": ["URI"],
  "frequency": {
    "metric": "Millisecond|Second|Minute|Hour|Day|Week|Month|Year",
    "units": "Integer"
  },
  "reference": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "requestedAccuracy": "Integer",
  "duration": {
    "metric": "Millisecond|Second|Minute|Hour|Day|Week|Month|Year",
    "units": "Integer"
  }
}
```

Response Header

The Location header field contains the URI of the publish/subscribe server:

```
http://host:port/rest/terminal_location/notifications
```

If the request fails, the Status-Line header field will contain the status code and the reason for the failure.

Response Body

There is no response body. The appropriate notifications (described below) are sent to the **endpoint** address provided by the application in the request body of this operation.

Examples

Example 11–13 Start Periodic Notification Request

```
PUT /rest/terminal_location/periodic-notification HTTP/1.1
X-Session-ID: app:4130997928482260925
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
Content-Length: 366
Content-Type: application/json

{
   "addresses":["tel:123"],
   "frequency":
        {
          "metric": "Second",
          "units": "5"
        },
   "reference":
        {
          "correlator":"6789",
          "endpoint":"bayeux/app_instance_1/tl",
          "interfaceName":"interfaceName"
        },
   "requestedAccuracy":"10.0",
   "duration":
        {
          "metric":"Minutes",
          "units":"30"
        }
}
```

Example 11–14 Start Periodic Notification Response

```
HTTP/1.1 200 Created
Date: Fri, 05 Nov 2101 09:59:05 GMT
Location: http://terminalloc_host:port/rest/terminal_location/notifications
Content-Length: 0
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1
```

Notification When Terminal Location is Successfully Retrieved (locationNotification)

See "Notification When Terminal Location is Successfully Retrieved (locationNotification)".

Notification of Error in Retrieving Terminal Location (locationError)

See "Notification of Error in Retrieving Terminal Location (locationError)".

Notification Signalling End to Monitoring Terminal (locationEnd)

See "Notification Signalling End to Monitoring Terminal's Location (locationEnd)".

End Notification

The End Notification operation terminates an application's previously set up notification to get the geographical and periodic information for a specified terminal.

To stop a previously set up Start Geographical Notification or Start Periodic Notification, provide the correlator for the notification passed earlier in the appropriate start request.

There is no request or response body for the End Notification operation. If the request fails, the body of the error response will contain the identifier for the notification and the type of exception.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://`*`host`*`:`*`port`*`/rest/terminal_location/notification/`*`${correlator}`*

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *correlator* is the correlator for the notification provided in the **reference** object of the initial request for the Start Geographical Notification or Start Periodic Notification operation.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure.

Response Body

There is no response body.

Examples

Example 11–15 End Notification Request

```
DELETE /rest/terminal_location/status-notification/6789 HTTP/1.1
X-Session-ID: app:4130997928482260925
Authorization: Basic ZG9tYWluX3VzZXI6ZG9tYWluX3VzZXI=
X-Param-Keys:
X-Param-Values:
User-Agent: Jakarta Commons-HttpClient/3.0
Host:servgtkpr_host.port
```

Example 11–16 End Notification Response

```
HTTP/1.1 204 No Content
Connection: close
Date: Thu, 04 Nov 2101 09:59:05 GMT
Content-Length: 0
X-Plugin-Param-Keys:
X-Plugin-Param-Values:
X-Powered-By: Servlet/2.5 JSP/2.1
```

12

Adding RESTful Payment Support

This chapter describes the operations in the Payment interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Payment Interface

Applications use the RESTful Payment interface to charge an amount to an end-user's account using Diameter, refund amounts to that account, and split charge amounts among multiple end-users. Applications can also reserve amounts, reserve additional amounts, charge against the reservation or release the reservation.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://host:port/rest/payment/index.html`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Charge Amount

The Charge Amount operation charges an amount directly to an end-user's application using the Diameter protocol.

To charge an amount for a call, provide the URI of the address of the end-user, a reference code in case there is any dispute regarding the charges, and the billing information to charge for the call.

There is no response body for the Charge Amount operation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/charge-amount`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Charge Amount operation accepts the following parameters:

- **charging**: A JSON object. Optional. This object defines the cost-charging properties for the call. A call with no charging parameters can be entered as `"charging": null`. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**. Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **endUserIdentifier**: String. Required. The address of the end-user's application that is to be charged.
- **referenceCode**: String. Required. A unique identifier in case of disputes with respect to the charges.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "charge": {
    "description": "String",
```

```
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "endUserIdentifier": "URI",
  "referenceCode": "String"
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Refund Amount

The Refund Amount operation refunds an amount directly to an end-user's application using Diameter.

To refund an amount for a call, provide the URI of the address of the end-use, a reference code in case there is any dispute regarding the charges, and the billing information to charge for the call. receive the message in the request body.

There is no response body for the Refund Amount operation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/refund-amount`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Refund Amount operation accepts the following parameters:

- **charging**. a JSON object. Optional. This object defines the cost-charging properties for the call. A call with no charging parameters can be entered as `"charging": null`. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the `charging` object is present in the body of the request. The text to be used for information and billing.
  - **amount**. Number (integer, or decimal). Optional. The amount to be refunded.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **endUserIdentifier**: String. Required. The address of the end-user's application that is to receive the refund.
- **referenceCode**: String. Required. A unique identifier in case of disputes with respect to the refund.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "charging": {
    "description": "String",
```

```
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "endUserIdentifier": "URI",
  "referenceCode": "String"
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Charge Split Amount

The Charge Split Amount operation charges an amount directly to multiple end-users concurrently (for example, for charging multiple participants in a conference.

To split the charge an amount for a call, provide the billing information to charge for the call, a reference code in case there is any dispute regarding the charges, the address of the end-user, and the percentage of the charges for which the end-user is liable.

There is no response body for the Charge Split Amount operation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/charge-split-amount`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Charge Split Amount operation accepts the following parameters:

- **charge**. a JSON object. Required. Use this object to define the cost-charging properties for the call. A call with no charging parameters can be entered as `"charging": null`. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**. Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **referenceCode**: String. Required. A unique identifier in case of disputes with respect to the charges.
- **splitInfo**. An array of JSON objects. Required. For each entry, the end-user identifier and the method by which the charges must be split.
  - **endUserIdentifier**: String. Required. The address of the end-user.
  - **percent**. Integer. Required. The percentage this end-user should be charged.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "charge": {
    "description": "String",
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "referenceCode": "String",
  "splitInfo": [{
    "endUserIdentifier": "URI",
    "percent": "Integer"
  }]
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Reserve Amount

The Reserve Amount operation reserves an amount for an account specified by the end-user identifier.

To reserve an amount for a call, provide the address of the end-user and the billing information for the call.

If the Reserve Amount operation is successful, the response body will contain the string identifier for the reservation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/reserve-amount`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Reserve Amount operation accepts the following parameters:

- **charge**. a JSON object. Optional. Use this object to define the cost-charging properties for the call. A call with no charging parameters can be entered as `"charging": null`. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**. Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **endUserIdentifier**: String. Required. The address of the end-user against whose account the reservation is made.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "charge": {
    "description": "String",
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
```

```
  },
  "endUserIdentifier": "URI"
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains the **result** attribute whose value is a String-formatted identifier for the reservation (used as **reservationIdentifier** in subsequent related operations).

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": "String"}
```

Reserve Additional Amount

The Reserve Additional Amount operation reserves an additional amount for an account specified by the end-user identifier.

To reserve an additional amount for a call, provide the reservation identifier obtained from the initial request to reserve an amount for the end-user and the billing information for the call.

If the Reserve Additional Amount operation is successful, the response body will contain the string identifier for the reservation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/reserve-additional-amount`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Reserve Additional Amount operation accepts the following parameters:

- **charge**. a JSON object. Optional. Use this object to define the cost-charging properties for the call. A call with no charging parameters can be entered as `"charging":null`. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**. Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **reservationIdentifier**: String. Required. The string identifier **result** obtained from the initial "Reserve Amount" operation for this account.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "charge": {
    "description": "String",
```

```
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "reservationIdentifier": "String"
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Charge Volume

The Charge Volume operation charges the specified volume to the account specified by the end-user identifier.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/charge-volume`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Charge Volume operation accepts the following parameters:

- **endUserIdentifier**: String. Required. Identifies the end-user account to be charged.
- **volume**: Long. Required. Identifies the volume amount to be charged. (This is not a currency amount.)
- **billingText**: String. Required. Textual information to appear on the bill.
- **referenceCode**: String. Required. Code to uniquely identify the request.
- **parameters**: JSON object. Optional. Additional name/value pairs to use to perform rating.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "endUserIdentifier": "URI"
  "volume": "BigDecimal"
  "billingText": "String"
  "referenceCode": "String"
  "parameters": {
    "name": "String",
    "value": "String",
  },
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Refund Volume

The Refund Volume operation directly applies a refund volume to the account specified by the end-user identifier.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/refund-volume`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Refund Volume operation accepts the following parameters:

- **endUserIdentifier**: String. Required. Identifies the end-user account to be refunded.
- **volume**: Long. Required. Identifies the volume amount to be refunded.
- **billingText**: String. Required. Textual information to appear on the bill.
- **referenceCode**: String. Required. Code to uniquely identify the request.
- **parameters**: JSON object. Optional. Additional name/value pairs to use to perform rating.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "endUserIdentifier": "URI"
  "volume": "BigDecimal"
  "billingText": "String"
  "referenceCode": "String"
  "parameters": {
    "name": "String",
    "value": "String",
  },
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Charge Split Volume

The Charge Split Volume operation applies a split volume charge to multiple end-user accounts.

The portion of the volume charge applied to each account is expressed as a percentage.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/charge-split-volume`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Charge Split Volume operation accepts the following parameters:

- **splitInfo**: A JSON object. Required. Use this object to define the end-user accounts and the percentage of the volume for the request to be billed to each account. Percentages must total 100:
  - **endUserIdentifier**: String. Required. Identifies an end-user account to be charged.
  - **percent**: Number (integer, or decimal). Required. The percentage of the transaction to be charged to the end-user account.
- **volume**: Long. Required. Identifies the volume amount to be refunded. (This is not a currency amount.)
- **billingText**: String. Required. Textual information to appear on the bill.
- **referenceCode**: String. Required. Code to uniquely identify the request.
- **parameters**: JSON object. Optional. Additional name/value pairs to use to perform rating.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "splitInfo": {
    "endUserIdentifier": "URI"
    "percent": "BigDecimal",
  },
  "volume": "BigDecimal"
  "billingText": "String"
```

```
  "referenceCode": "String"
  "parameters": {
    "name": "String",
    "value": "String",
  },
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Get Amount

The Get Amount operation converts a volume to a currency amount for the end-user account.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/get-amount`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Get Amount operation accepts the following parameters:

- **endUserIdentifier**: String. Required. Identifies the end-user account for the currency calculation.
- **volume**: Long. Required. Identifies the volume to be converted to a currency amount.
- **parameters**: A JSON object. Optional. Additional name/value pairs to use to perform rating.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "endUserIdentifier": "URI"
  "volume": "BigDecimal"
  "parameters": {
    "name": "String",
    "value": "String",
  },
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

A **charging** object is returned, with the **currency** element containing the converted currency amount.

Charge Reservation

The Charge Reservation operation charges a previously reserved amount against an end-user account.

To charge a previously reserved amount to an end-user account, provide the information for billing, the reservation identifier obtained from the initial request to reserve an amount for the end-user, and the reference code for any possible disputes.

Authorization

Basic

HTTP Method

POST

URI

`http://host:port/rest/payment/charge-reservation`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Charge Reservation operation accepts the following parameters:

- **charge**: A JSON object. Required. Use this object to define the cost-charging properties for the call. A call with no charging parameters can be entered as `"charging":null`. If a charge is to be applied, provide values for the following in the **charge** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**. Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].
- **referenceCode**: String. Required. A unique identifier in case of disputes with respect to the charges.
- **reservationIdentifier**: String. Required. The string identifier **result** obtained from the initial Reserve Amount operation for this account.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "charge": {
    "description": "String",
```

```
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  },
  "referenceCode": "String",
  "reservationIdentifier": "String"
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Release Reservation

The Release Reservation operation returns funds left in a reservation to the account against which this reservation was made.

To returns funds left in a reservation to an account, provide the reservation identifier obtained from the initial request to reserve an amount for the end-user.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/release-reservation`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Release Reservation operation accepts the following parameter:

- **reservationIdentifier**: String. Required. The string identifier **result** obtained from the initial Reserve Amount operation for this account.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"reservationIdentifier": "String"}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Reserve Volume

The Reserve Volume operation reserves a volume for the account specified by the end-user identifier.

If the Reserve Volume operation is successful, the response body contains the string identifier for the reservation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/reserve-volume`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Reserve Volume operation accepts the following parameters:

- **endUserIdentifier**: String. Required. Identifies the end-user account for which the reservation should be placed.
- **volume**: Long. Required. Identifies the volume amount to be reserved.
- **billingText**: String. Required. Textual information to appear on the bill.
- **parameters**: A JSON object. Optional. Additional name/value pairs to use to perform rating.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "endUserIdentifier": "URI"
  "volume": "BigDecimal"
  "billingText": "String"
  "parameters": {
    "name": "String",
    "value": "String",
  },
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body consists of a String containing an identifier for the newly created reservation.

Reserve Additional Volume

The Reserve Additional Volume operation adds or subtracts a volume to or from an existing volume reservation for the account specified by the end-user identifier.

To reserve an additional volume for a call, provide the reservation identifier obtained from the initial request to reserve an amount for the end-user and the billing information for the call.

If the Reserve Additional Volume operation is successful, the response body will contain the string identifier for the reservation.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/reserve-additional-volume`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Reserve Additional Volume operation accepts the following parameters:

- **reservationIdentifier**: String. Required. Identifies the reservation to be amended.
- **volume**: Long. Required. Identifies the volume amount to be added to or subtracted from to the existing reservation.
- **billingText**: String. Required. Textual information to appear on the bill.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "reservationIdentifier": "String"
  "volume": "BigDecimal"
  "billingText": "String"
  },
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

Get Amount Reserve Charging

The Get Amount Reserve Charging operation converts a reserved volume to a currency amount for the end-user account.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/payment/get-amount-reserve-charging`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Get Amount Reserve Charging operation accepts the following parameters:

- **endUserIdentifier**: String. Required. Identifies the end-user account for the currency calculation.
- **volume**: Long. Required. Identifies the volume amount to be converted to a currency amount.
- **parameters**: A JSON object. Optional. Additional name/value pairs to use to perform rating.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "endUserIdentifier": "URI"
  "volume": "BigDecimal"
  "parameters": {
    "name": "String",
    "value": "String",
  },
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

A **charging** object is returned, with the **currency** element containing the converted reserved currency amount.

13

Adding RESTful Quality of Service Support

This chapter describes the RESTful interface for the Oracle Communications Services Gatekeeper Quality of Service (QoS) communication service.

About the QoS Interface

An application can use the QoS RESTful interface to apply a QoS policy, query, modify and remove that policy and register as well as unregister for QoS-related notifications. A Policy Control and Charging Rules Function (PCRF) provider can also return QoS events to registered applications.

See *Services Gatekeeper Communication Service Reference Guide* for details on using the Extended Web Service Quality of Service/Diameter communication service.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of QoS operations can be found at

`http://host:port/application.wadl`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

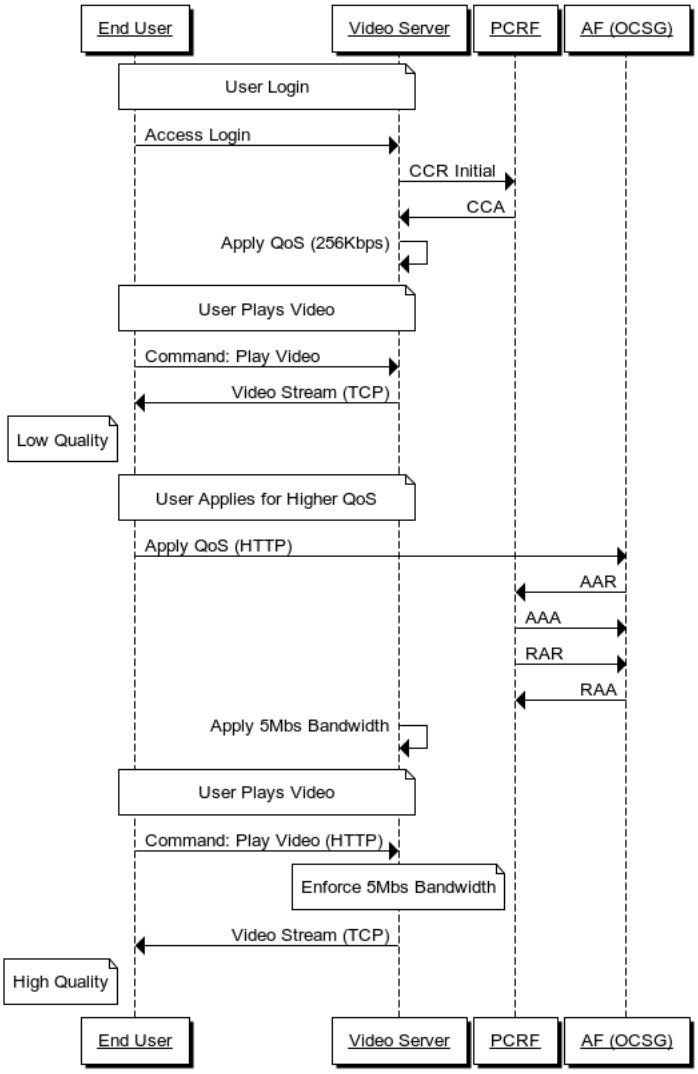
Example QoS Scenario

A typical QoS scenario involves a subscriber using a handset to access a video feed using a video application installed on the handset. Initially, because the default QoS is set to a low bandwidth, the video stops and stutters frequently as it is buffered repeatedly over the low speed connection. The subscriber requests a faster QoS through the application, presumably with a corresponding billing charge. Services Gatekeeper forwards that request to a PCRF which then applies the upgraded QoS. The subscriber's video now streams at the upgraded speed, without stuttering.

Note: While QoS frequently refers to raw bandwidth speed, it can apply to any factors that affect network performance: for example, connection latency and time-out.

Figure 13–1 shows a detailed QoS call flow sequence.

Figure 13–1 Example QoS Call Sequence



In Figure 13–1:

1. A user logs in to the video server.
2. The video server initiates an initial Credit Check Request (CCR) to the PCRF.
3. The PCRF returns a Credit Check Authorization (CCA) to the video server and the low bandwidth, 256Kbps, QoS plan is applied.
4. The user plays the video using the low bandwidth QoS plan; video playback is low quality with stuttering and continual buffering requests.
5. The user requests a better QoS plan using the applyQos RESTful request from the handset's host application.
6. Upon receiving the applyQoS request, Services Gatekeeper issues an Authorization and Authentication Request (AAR) to the PCRF which then returns an Authorization and Authentication Answer (AAA).
7. The PCRF issues a Re-Authorization Request (RAR) to Services Gatekeeper, which then returns a Re-Authorization Answer (RAA), and the high bandwidth, 5Mbps QoS plan is applied.

8. The user plays the video, and the new 5Mbps QoS plan is enforced. The video plays smoothly, without stuttering or continued buffering.

Configuring QoS for Services Gatekeeper

Before you can implement QoS functionality, a QoS plug-in must be deployed and configured in Services Gatekeeper. For information on deploying and configuring QoS plug-ins, see *Services Gatekeeper Communication Service Reference Guide*.

Using OAuth with QoS

The Services Gatekeeper QoS communication service fully supports OAuth 2.0 authentication between the QoS communication service itself and an AT application.

To establish OAuth authentication between the QoS communication service and your application, do the following:

1. From your application, contact the QoS communication service and request an OAuth token.

   The QoS communication service will return an OAuth token to your application.

2. Add the access_token (UUID) to your application's request header:

   ```
   access_token: 3ddc24b2-5b17-4d46-8818-6e14726b217c
   ```

3. In addition, add the Authorization parameter to your application's HTTP header:

   ```
   Authorization: Bearer 3ddc24b2-5b17-4d46-8818-6e14726b217c
   ```

For more information on using OAuth authentication, see *Services Gatekeeper OAuth Guide*.

Apply QoS

The Apply QoS operation requests that a QoS plan be applied to end user IDs as specified in a Services Gatekeeper QoS plug-in regular expression matching rule.

Authorization

Basic or OAuth 2.0

HTTP Method

`POST`

URI

`http://host:port/ApplicationQoSService/${endUserId}/qos`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *endUserId* is a valid end user identifier.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Apply QoS operation accepts the following parameters:

- **duration**: Unsigned Long. Optional. The duration of the applied QoS in seconds. If no duration is specified, the QoS session will not time out and will end only when it is explicitly removed.
- **applicationIdentifier**: String. Optional. Identifies the service to which the application-facing service session belongs.
- **mediaComponentDescription**: Complex. Optional. Service information for a single media component within an application-facing session.

  See "mediaComponentDescription" for details on each of the mediaComponentDescription parameters.
- **serviceInfoStatus**: Complex. Optional. Indicates the status of the service information that the application facing interface is providing to the PCRF. If this parameter is not provided, the value **FINAL_SERVICE_INFORMATION** is assumed. The service info status is specified as one of the following:
  - **FINAL_SERVICE_INFORMATION**
  - **PRELIMINARY_SERVICE_INFORMATION**
- **chargingIdentifier**: String. Optional. Application facing charging identifier.
- **sipForkingIndication**. Enumerated. Optional. Indicates if multiple dialogs are related to a single Diameter session. If this parameter is not provided, the value **SINGLE_DIALOGUE** is assumed. The forking indication is specified as one of the following:

  - **SINGLE_DIALOGUE**
  - **SEVERAL_DIALOGUES**

- **subscriptionId**: Complex. Optional. An end user's subscription ID.

  See "subscriptionID" for details on each of the subscriptionId parameters.

- **supportedFeatures**: Complex. Optional. If present, informs the destination host about the features that the origin host requires to successfully complete the command exchange.

  See "supportedFeatures" for details on each of the supportedFeatures parameters.

- **reservationPriority**. Enumerated. Optional. Applies to all IP flows within the media component and describes the relative importance of the IP flow as compared to other IP flows. If this parameter is not specified, the default value is **0**. The reservation priority is specified as one of the following:
  - **0**
  - **7**
- **framedIPAddress**: String. Optional. The valid routable IPv4 or IPv6 address that is applicable for the IP flows toward the user equipment at the PCEF.
- **framedIPv6Prefix**: String. Optional. A valid full IPv6 address that is applicable to one or more IP flows toward the user equipment at the PCEF.
- **calledStationId**: String. Optional. If a private IP address is being used, the ID of the packet data network.
- **serviceURN**. Enumerated. Optional. Indicates that the application function (AF) session is used for emergency traffic. The service URN is specified as one of the following:
  - **counseling**
  - **counseling.children**
  - **counseling.mental-health**
  - **counseling.suicide**
  - **sos**
  - **sos.ambulance**
  - **sos.animal-control**
  - **sos.fire**
  - **sos.gas**
  - **sos.marine**
  - **sos.mountain**
  - **sos.physician**
  - **sos.poison**
  - **sos.police**
- **sponsoredConnectivityData**: Complex. Optional. Indicates the data associated with the sponsored data connectivity that the AF is providing to the PCRF.

  See "sponsoredConnectivityData" for details on each of the sponsoredConnectivityData parameters.

- **mPSIdentifier**: String. Optional. Indicates that an assured-forwarding (AF) session relates to a Malware Protection Systems (MPS) session. It contains the national variant for the MPS service name, for example, Next Generation Network GETS/priority service (NGN GETS).
- **applicationIdentifier**: String. Optional. Identifies the particular service to which the media component belongs. If the parameter is not present, the applicationIdentifier from the main body of the request is used.

mediaComponentDescription

These are the parameters for the optional mediaComponentDescription.

- **mediaComponentNumber**: Unsigned Integer. Required. Ordinal number of the media component.
- **mediaSubComponent**: Complex. Optional. The requested bitrate and filters for the set of IP flows identified by their common flow identifier.

  See "mediaSubComponent" for details on each of the mediaSubComponent parameters.
- **applicationIdentifier**: String. Optional. Identifies the particular service to which the media component belongs. If the parameter is not present, the applicationIdentifier from the main body of the request is used.
- **mediaType**: Enumerated. Optional. Determines the media type of the session component. The media type is specified as one of the following:
  - **AUDIO**
  - **VIDEO**
  - **DATA**
  - **APPLICATION**
  - **CONTROL**
  - **TEXT**
  - **MESSAGE**
  - **OTHER**
- **maxRequestedBandwidthUL**. Unsigned Integer. Optional. The maximum requested bandwidth in bits per second for an uplink IP flow.
- **maxRequestedBandwidthDL**. Unsigned Integer. Optional. The maximum requested bandwidth in bits per second for a downlink IP flow.
- **minRequestedBandwidthUL**. Unsigned Integer. Optional. The minimum requested bandwidth in bits per second for an uplink IP flow.
- **minRequestedBandwidthDL**. Unsigned Integer. Optional. The minimum requested bandwidth in bits per second for a downlink IP flow.
- **flowStatus**. Enumerated. Optional. Describes whether the IP flows are enabled or disabled. The flow status is specified as one of the following:
  - **ENABLED-UPLINK**
  - **ENABLED-DOWNLINK**
  - **ENABLED**
  - **DISABLED**

  - **REMOVED**

- **reservationPriority**. Enumerated. Optional. Applies to all those IP flows within the media component and describes the relative importance of the IP flow as compared to other IP flows. If this parameter is not specified, the value is **0**. The reservation priority is specified as one of the following:
  - **0**
  - **7**
- **rSBandwidth**. Unsigned Integer. Optional. Indicates the maximum required bandwidth in bits per second for RTCP sender reports within the session component.
- **rBBandwidth**. Unsigned Integer. Optional. Indicates the maximum required bandwidth in bits per second for RTCP receiver reports within the session component.
- **codecData**: String. Optional. Codec-related information known at the AF. The encoding rule should follow 3gpp TS 29.214 [5.3.7].

mediaSubComponent

These are the parameters for the optional mediaSubComponent parameter.

- **flowNumber**. Unsigned Integer. Required. Ordinal number of the IP flow.
- **flowDescription**: String. Optional. Filters for an IP flow. The format must follow RFC3588 [4.3] IPFilterRule and 3gpp TS 29.214 [5.3.8].
- **flowStatus**. Enumerated. Optional. Describes whether the IP flows are enabled or disabled. The flow status is specified as one of the following:
  - **ENABLED-UPLINK**
  - **ENABLED-DOWNLINK**
  - **ENABLED**
  - **DISABLED**
  - **REMOVED**
- **flowUsage**. Enumerated Optional. Provides information about the usage of IP flows. The flow usage is specified as one of the following:
  - **NO_INFORMATION**
  - **RTCP**
  - **AF_SIGNALLING**
- **maxRequestedBandwidthUL**. Unsigned Integer. Optional. The maximum requested bandwidth in bits per second for an uplink IP flow.
- **maxRequestedBandwidthDL**. Unsigned Integer. Optional. The maximum requested bandwidth in bits per second for a downlink IP flow
- **signallingProtocol**. Enumerated. Optional. Indicates the protocol used for signalling between the UE and the AF. If this parameter is absent, the value **NO_INFORMATION** is assumed. The signalling protocol is specified as one of the following:
  - **NO_INFORMATION**

subscriptionID

These are the parameters for the optional subscriptionID parameter.

- **subscriptionIdType**. Enumerated. Required. Type of the end user's subscription ID. The subscription ID type is specified as one of the following:
  - **END_USER_E164**
  - **END_USER_IMSI**
  - **END_USER_NAI**
  - **END_USER_PRIVATE**
- **subscriptionIdData**: String. Required. Value of the end user's subscription ID.

supportedFeatures

These are the parameters for the optional supportedFeatures parameter.

- **vendorId**. Unsigned Integer. Required. The vendor ID.
- **featureListID**. Unsigned Integer. Required. The feature list ID.
- **featureList**. Unsigned Integer. Required. A list of the application's supported features.

sponsoredConnectivityData

These are the parameters for the optional sponsoredConnectivityData parameter.

- **sponsorIdentity**: String. Optional. String identifying the sponsor.
- **applicationServiceProviderIdentity**: String. Optional. String identifying the application service provider.
- **grantedServiceUnit**: Complex. Optional. Provides a usage threshold level to the PCRF if the volume of traffic allowed during the sponsored data connectivity is monitored.

  See "grantedServiceUnit" for details on each of the grantedServiceUnit parameters.
- **usedServiceUnit**: Complex. Optional. Provides the number of used units from the point at which the service became active, or, if interim measurements are used during the session, the point at which the previous session ended.

  See "grantedServiceUnit" for details on each of the usedServiceUnit parameters. Note that the parameters are identical to those of grantedServiceUnit.

grantedServiceUnit

These are the parameters for the optional grantedServiceUnit parameter.

- **tariffTimeChange**: Enumerated. Optional. Determines the timing of the unit relative to a tariff time change. The tariff time change parameter can take the following values:
  - **UNIT_BEFORE_TARIFF_CHANGE**
  - **UNIT_AFTER_TARIFF_CHANGE**
  - **UNIT_INDETERMINATE**
- **cCTime**. Unsigned Integer. Optional. Indicates the length of requested, granted or used time in seconds.
- **cCMoney**: Complex. Optional. Specifies the monetary amount in a given currency. See "cCMoney" for details.

- **cCTotalOctets**. Unsigned 64-bit Integer. Optional. Specifies the total number of the granted, requested or used octets, regardless of the flow direction.
- **cCInputOctets**. Unsigned 64-bit Integer. Optional. Specifies the total number of the granted, requested or used octets, that either can be or have been received from an end user.
- **cCOutputOctets**. Unsigned 64-bit Integer. Optional. Specifies the total number of the granted, requested or used octets, that either can be or have been sent to an end user.
- **cCServiceSpecificUnits**. Unsigned 64-bit Integer. Optional. Specifies the number of service specific units provided in a particular service.

cCMoney

These are the parameters for the optional ccMoney parameter.

- **unitValue**. Decimal. Required. Specifies a multiplier that converts between units of a particular unit type and abstract units within the service credit pool.
- **currencyCode**. Unsigned Integer. Optional. Specifies in which currency the cost was given. Must follow the numeric values defined in the ISO 4217 standard.

Custom AVPs in QoS Requests

In addition to the preset elements, Services Gatekeeper QoS requests can accommodate custom AVP definitions as long as they are supported by the Diameter server. Such custom AVP definitions can be added to the following elements in any number:

- **mediaSubComponent**
- **supportedFeatures**
- **sponsoredConnectivityData**
- **grantedServiceUnit**
- **usedServiceUnit**

Example 13–1 shows a portion of a JSON request specifying a parameter with the name **myCustomDiameterParameter** and the value **My diameter parameter value**.

Example 13–1 Custom JSON AVP Request

```
"parameter": {
  "name": "myCustomDiameterParameter"
  "value": "My diameter parameter value"
}
```

Request Example

Example 13–2 shows an example of an apply QoS request body.

Example 13–2 Apply QoS Request Body

```
{
  "qoSFeatureProperties": {
    "duration": 3600,
    "applicationIdentifier": "test_app_id",
    "mediaComponentDescription": [
      {
        "mediaComponentNumber": 1,
        "mediaSubComponent": [
```

```
        {
          "flowNumber": 1,
          "flowDescription": [
            "test_flow"
          ],
          "flowStatus": "ENABLED-UPLINK",
          "flowUsage": "NO_INFORMATION",
          "maxRequestedBandwidthUL": 3300,
          "maxRequestedBandwidthDL": 2200,
        }
      ],
      "applicationIdentifier": "test",
      "mediaType": "AUDIO",
      "maxRequestedBandwidthUL": 1000,
      "maxRequestedBandwidthDL": 1000,
      "minRequestedBandwidthUL": 10,
      "minRequestedBandwidthDL": 10,
      "flowStatus": "ENABLED-UPLINK",
      "reservationPriority": 1,
      "rSBandwidth": 10,
      "rRBandwidth": 10,
      "codecData": [
        "codec"
      ]
    }
  ],
  "serviceInfoStatus": "FINAL_SERVICE_INFORMATION",
  "chargingIdentifier": "charging_id",
  "sIPForkingIndication": "SINGLE_DIALOGUE",
  "subscriptionId": [
    {
      "subscriptionIdType": "END_USER_E164",
      "subscriptionIdData": "861013388991111"
    }
  ],
  "supportedFeatures": [
    {
      "vendorId": 1,
      "featureListID": 1,
      "featureList": 333
    }
  ],
  "reservationPriority": 1,
  "framedIPAddress": "0A987898",
  "calledStationId": "adsf",
  "serviceURN": "counseling",
  "sponsoredConnectivityData": {
    "sponsorIdentity": "1adsf",
    "applicationServiceProviderIdentity": "1ss",
    "grantedServiceUnit": {
      "tariffTimeChange": 122,
      "cCTime": 444,
      "cCMoney": {
        "unitValue": 1,
        "currencyCode": 11
      },
      "cCTotalOctets": 1,
      "cCInputOctets": 1,
      "cCOutputOctets": 1,
      "cCServiceSpecificUnits": 1
```

```
      }
    },
    "mPSIdentifier": "mps_id"
  }
}
```

Response Header

In addition to the standard header fields, two additional fields are returned:

- **X-Plugin-Param-Keys**. Comma-separated keys that map to the values returned in X-Plugin-Param-Values. Two values are returned:
  - **AVP_LIST**. Key matching the Avp-List XML structure returned in X-Plugin-Param-Values.
  - **session-id**. Key matching the session ID returned in X-Plugin-Param-Values.
- **X-Plugin-Param-Values**. Comma-separated values that are mapped to their respective keys returned in X-Plugin-Param-Keys.

For the **Avp-List** XML structure, different Diameter servers may return different elements and values, the only required element being the **Result-Code**. For detailed information on the possible elements and values, see the *UMTS Policy and charging control over Rx reference point (ETSI TS 129 214 V10.6.0)* available at .

http://www.etsi.org/deliver/etsi_ts/129200_129299/129214/10.06.00_60/ts_129214v100600p.pdf

Likewise, the **session-id** format is dependent upon your Diameter server.

Example 13–3 shows a possible response header.

Example 13–3 Response Header

```
HTTP/1.1 201 Created
Date: Mon, 11 Mar 2013 03:29:11 GMT
Transfer-Encoding: chunked
Location:
http://localhost:8001/ApplicationQoSService/tel%3A88888888/qos/localhost%3B
1362972174%3B0-1362972554169
Content-Type: application/json
X-Plugin-Param-Keys: AVP_LIST,session-id
X-Plugin-Param-Values:
<Avp-List><Session-Id>localhost;1362972174;0</Session-Id><Origin-Host>MINFXU-CN</
Origin-Host><Origin-Realm>oracle.com</Origin-Realm><Result-Code>2001</Result-Code>
<IP-CAN-Type>0</IP-CAN-Type><RAT-Type>0</RAT-Type></Avp-List>,localhost;1362972174
;0
X-Powered-By: Servlet/2.5 JSP/2.1
```

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains an array of structures as the value for **applyQoSFeatureResponse**. Each element in the array contains values for the following parameters.

- **requestId**: String. The request ID. Used to uniquely identify the QoS session.
- **actualProperties**: Complex. Additional properties resulting from the request.

See "actualProperties" for details on each of the actualProperties parameters.

actualProperties

These are the values returned for the actualProperties parameter.

- **accessNetworkChargingIdentifier**: Complex. Contains a charging identifier within the accessNetworkChargingIdentifierValue AVP along with information about the flows transported within the corresponding bearer within the flows AVP. If no flows AVP is provided, the accessNetworkChargingIdentifierValue applies to all flows within the AF session.
- **accessNetworkChargingAddress**: String. Indicates the IP address of the network entity that handles charging within the access network.
- **acceptableServiceInfo**: Complex. Contains the maximum bandwidth for an AF session and/or for specific media components that will be authorized by the PCRF.
- **iPCANType**. Enumerated. Indicates the type of Connectivity Access Network (CAN) to which a user is connected. The CAN type is specified as one of the following:
  - **_3GPP-GPRS**
  - **DOCSIS**
  - **xDSL**
  - **WiMAX**
  - **_3GPP2**
  - **_3GPP-EPS**
  - **Non-3GPP-EPS**
- **rATType**. Identifies the Radio Access Technology (RAT) that is servicing the user equipment. The RAT type is specified as one of the following:
  - **WLAN**
  - **VIRTUAL**
  - **UTRAN**
  - **GERAN**
  - **GAN**
  - **HSPA_EVOLUTION**
  - **EUTRAN**
  - **CDMA2000_1X**
  - **HRPD**
  - **UMB**
  - **EHRPD**
- **flows**: Complex. Indicates IP flows using their flow identifiers.

  See "flows" for details on each of the flows parameters.
- **supportedFeatures**: Complex. See "supportedFeatures" for detailed information.

accessNetworkChargingIdentifier

These are the values returned for the accessNetworkChargingIdentifier parameter.

- **accessNetworkChargingIdentifierValue**: String. Includes the charging identifier.
- **flows**: Complex. Indicates IP flows using their flow identifiers. See "flows" for details on each of the flows parameters.

flows

These are the values returned for the flows parameter.

- **mediaComponentNumber**. Unsigned Integer. Ordinal number of the media component.
- **flowNumber**. Integer. Indicates the number of the flow. If no flowNumber AVPs are supplied, this refers to all flows matching the media component number.
- **finalUnitAction**. Enumerated. When reporting an out of credit condition, the finalUnitAction indicates the termination action applied to the impacted flows. Indicates to the credit-control client the action to be taken when a user's account cannot cover the service cost. The final unit action is specified as one of the following:
  - **TERMINATE**
  - **REDIRECT**
  - **RESTRICT_ACCESS**

acceptableServiceInfo

These are the values returned for the acceptableServiceInfo parameter.

- **mediaComponentDescription**: Complex. See "mediaComponentDescription" for detailed information.
- **maxRequestedBandwidthUL**. Unsigned Integer. The maximum requested bandwidth in bits per second for an uplink IP flow.
- **maxRequestedBandwidthDL**. Unsigned Integer. The maximum requested bandwidth in bits per second for a downlink IP flow

Response Body Example

Example 13–4 shows an example of an apply QoS response body.

Example 13–4 Apply QoS Response Body

```
{"applyQoSFeatureResponse": {
    "requestId": "localhost;1362972174;0-1362972554169",
    "actualProperties": {
      "iPCANType": "_3GPP-GPRS",
      "rATType": "WLAN"
    }
  }
}
```

Apply Template-Based QoS

An operation to apply a template-based QoS requires that a QoS plan based upon a template stored in Services Gatekeeper be applied to end user IDs. The QoS plan is specified as a Services Gatekeeper QoS plug-in regular expression matching rule.

QoS Templates

QoS templates must be formatted according to the XSD found in the **xsd** subdirectory in the **plugin_qos_diameter.jar** file, which itself is contained within the **wlng_nt_qos.ear** archive located in *Middleware_Home*/ocsg_*release*/applications directory, where release is the release version of Services Gatekeeper. Example 13–10 shows a reference template containing all of the possible elements and attributes for a request.

Following is a sample QoS template:

Example 13–5 QoS Template Example

```
<QoSTemplate xmlns="http://oracle/ocsg/rest/qos/template"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://oracle/ocsg/rest/qos/template ../xsd/qosTemplate.xsd"
  templateId="default_template">
  <applicationIdentifier parameterName="$APP_
ID">4d6f62696c655456</applicationIdentifier>
  <!--Zero or more repetitions: -->
  <mediaComponentDescription>
    <mediaComponentNumber>0</mediaComponentNumber>
    <!--Zero or more repetitions: -->
    <mediaSubComponent>
      <flowNumber parameterName="$FLOW_NUMBER_0">1</flowNumber>
      <!--0 to 2 repetitions: -->
      <flowDescription parameterName="$FLOW_DESCRIPTION_0">
 <![CDATA[permit out 8001 from assigned 34 to 24.2.1.6/18
8000]]></flowDescription>
    </mediaSubComponent>
                <mediaType parameterName="$MED_TYPE">VIDEO</mediaType>
    <flowStatus parameterName="$FLOW_STATUS">ENABLED</flowStatus>
  </mediaComponentDescription>
  <!--Optional: -->M
  <serviceInfoStatus parameterName="$SERV_INFO_STATUS">PRELIMINARY_SERVICE_
INFORMATION</serviceInfoStatus>
  <!--Optional: -->
  <chargingIdentifier parameterName="$CHG_ID">charging-id-555</chargingIdentifier>
  <!--Optional: -->
  <subscriptionId>
    <subscriptionIdType parameterName="$SUB_ID_TYPE">END_USER_
E164</subscriptionIdType>
    <subscriptionIdData parameterName="$SUB_ID_
DATA">14128771501</subscriptionIdData>
  </subscriptionId>
  <serviceURN parameterName="$SERV_URN">sos.police</serviceURN>
</QoSTemplate>
```

In Example 13–5, each element has a **parameterName** attribute whose value maps to the request's parameter name. The **parameterName** attribute identifier must be unique throughout the entire template. For example, if you have two instances of the element **applicationIdentifier** (one for the whole template and one for a sub-component), you can use the following names for each instance: *$APP_ID_0* and *$APP_ID_1*.

If a parameter is set dynamically in a request, its value replaces the default value configured in the template. For example, if a request sets the *$FLOW_DESCRIPTION_0* parameter value to "modified flow description", the PCRF will receive that value rather than the one defined in the template.

Custom AVPs in QoS Templates

In addition to the preset elements, QoS templates can accommodate custom AVP definitions, both simple and enumerated, as long as they are supported by the Diameter server. Any number of such custom AVP definitions can be added to the following elements:

- **mediaSubComponent**
- **supportedFeatures**
- **sponsoredConnectivityData**
- **grantedServiceUnit**
- **usedServiceUnit**

Example 13–6 shows a simple custom AVP template in which the type of the custom parameter is set to **String**, the parameterName is set to **$MY_CUSTOM_DIAMETER_PARAMETER,** and the value of the custom parameter is set to **My Diameter value**.

Example 13–6 Custom AVP Template Element

```
<avp name="myCustomDiameterParameter" description="This is a sample AVP"
code="234567" may-encrypt="true" mandatory-flag="optional" vendor-id="Oracle
Corporation" constrained="false">
   <type type-name="Integer32"/>
   <value parameterName="$MY_CUSTOM_DIAMETER_PARAMETER">"My Diameter
Value"</value>
</avp>
```

Example 13–7 shows a custom enumerated AVP element where the AVP name is **myCustomDiameterParameter**, and two possible enumerated values are defined:

- **ENUM_1**, the logical name associated with the enumerated type, **0**.
- **ENUM_2**, the logical name associated with the enumerated type, **1**.

The parameterName, **$ENUM_VAL**, can be replaced like any standard template parameter with a value of either **ENUM_1** or **ENUM_2**.

> **Note:** For custom enumerated AVPs, the type element's type-name attribute is always **Integer32**.

Example 13–7 Custom Enumerated AVP Template Element

```
<avp name="myCustomEnumDiameterParameter" description="This is a sample AVP"
code="12345" may-encrypt="true" mandatory-flag="required" vendor-id="Oracle
Corporation" constrained="false">
   <type type-name="Integer32"/>
   <enum name="ENUM_1" code="0"/>
   <enum name="ENUM_2" code="1"/>
   <value parameterName="$ENUM_VAL">ENUM_1</value>
</avp>
```

Managing QoS Templates in Services Gatekeeper

You use the Services Gatekeeper Administration Console or the Platform Test Environment to load, modify and query QoS templates using MBeans. Table 13–1 lists the available MBean operations and their descriptions:

Table 13–1 QoS Template Management MBean Operations

Operation	Description
listQoSRequestTemplateMatchRules	Lists all of the match rules that have been defined for the QoS plug-in.
loadQoSRequestTemplate	Loads a QoS template.
retrieveQoSRequestTemplate	Retrieves a QoS template associated with a particular subscriber ID or a range of subscriber IDs.
deleteQoSRequestTemplate	Deletes a QoS template associated with a particular subscriber ID or a range of subscriber IDs.

For more information on loading, retrieving, listing, and deleting QoS templates, see *Services Gatekeeper Communication Service Reference Guide*.

Authorization

Basic or OAuth 2.0

HTTP Method

`POST`

URI

`http://host:port/ApplicationQoSService/${endUserId}/qos/templatebased`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *endUserId* is a valid end user identifier.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the operation to apply template-based QoS accepts the following parameters:

- **duration**: Unsigned Long. Required. The duration of the applied QoS in seconds.
- **applicationIdentifier**: String. Optional. Identifies the service to which the application-facing service session belongs.
- **framedIPAddress**. Hexadecimal Binary. Optional. The valid routable IPv4 or IPv6 address that is applicable for the IP Flows toward the user equipment at the PCEF.
- **framedIPv6Prefix**. Hexadecimal Binary. Optional. A valid full IPv6 address that is applicable to an IP flow or IP flows toward the user equipment at the PCEF.

- **calledStationId**: String. Optional. If a private IP address is being used, the ID of the packet data network.
- **parameter**: Complex. Optional. An array of JSON objects that define which template parameters will be replaced and what the replacement values will be.

  The array of JSON objects are a collection of one or more AVPs labeled **name** and **value** that determine which parameters in the QoS template will be replaced and what the replacement values will be. In Example 13–8, the values for the template parameters **$CHG_ID** and **$MAX_REQ_BAND_DL** will be replaced with the values **"charging_id_test"** and **2048** respectively.

Request Body Example

Example 13–8 shows a request body associated with an example Template-based QoS request.

Example 13–8 Example Request Body when a Template-Based QoS is Applied

```
{
  "templateQoSFeatureProperties": {
    "duration": 3600,
    "applicationIdentifier": "app_id",
    "framedIPAddress": "0A0B9899",
    "calledStationId": "called_station_id",
    "parameter": [
      {
        "name": "$CHG_ID",
        "value": "charging_id_test"
      },
      {
        "name": "$MAX_REQ_BAND_DL",
        "value": 2048
      }
    ]
  }
}
```

Response Header

For details on the response header, see the Apply QoS "Response Header" section.

Response Body

The response body parameters for the request to apply template-based QoS are the same as those for the Apply QoS operation. See the Apply QoS "Response Body" section for details.

Response Body Example

Example 13–9 shows the response body associated with an example Template-based QoS request.

Example 13–9 Example Response Body for Template-Based QoS Request

```
{
  "applyQoSFeatureResponse": {
    "requestId": "localhost;1362972174;1-1362973261091",
    "actualProperties": {
      "iPCANType": "_3GPP-GPRS",
      "rATType": "WLAN"
```

```
    }
  }
}
```

Reference: Complete QoS Template

Example 13–10 is a QoS template containing all of the available elements and attributes from the reference XSD.

Example 13–10 A Complete QoS Template

```
<QoSTemplate xmlns="http://oracle/ocsg/rest/qos/template"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://oracle/ocsg/rest/qos/template ../xsd/qosTemplate.xsd"
  templateId="default_template">
  <applicationIdentifier parameterName="$APP_ID">test_
appId</applicationIdentifier>
  <!--Zero or more repetitions: -->
  <mediaComponentDescription>
    <mediaComponentNumber>0</mediaComponentNumber>
    <!--Zero or more repetitions: -->
    <mediaSubComponent>
      <flowNumber parameterName="$FLOW_NUMBER_0">0</flowNumber>
      <!--0 to 2 repetitions: -->
      <flowDescription parameterName="$FLOW_DESCRIPTION_0">flow0_
Description</flowDescription>
      <!--Optional: -->
      <flowStatus parameterName="$FLOW_STATUS_0">ENABLED</flowStatus>
      <!--Optional: -->
      <flowUsage parameterName="$FLOW_USAGE_0">NO_INFORMATION</flowUsage>
      <!--Optional: -->
      <maxRequestedBandwidthUL parameterName="$MAX_REQ_BAND_UL_
0">1000</maxRequestedBandwidthUL>
      <!--Optional: -->
      <maxRequestedBandwidthDL parameterName="$MAX_REQ_BAND_DL_
0">1000</maxRequestedBandwidthDL>
      <!--Optional: -->
    </mediaSubComponent>
    <!--Optional: -->
    <applicationIdentifier parameterName="$MED_DES_APP_ID">test_
appId</applicationIdentifier>
    <!--Optional: -->
    <mediaType parameterName="$MED_TYPE">AUDIO</mediaType>
    <!--Optional: -->
    <maxRequestedBandwidthUL parameterName="$MAX_REQ_BAND_
UL">1000</maxRequestedBandwidthUL>
    <!--Optional: -->
    <maxRequestedBandwidthDL parameterName="$MAX_REQ_BAND_
DL">1000</maxRequestedBandwidthDL>
    <!--Optional: -->
    <minRequestedBandwidthUL parameterName="$MIN_REQ_BAND_
UL">10</minRequestedBandwidthUL>
    <!--Optional: -->
    <minRequestedBandwidthDL parameterName="$MIN_REQ_BAND_
UL">10</minRequestedBandwidthDL>
    <!--Optional: -->
    <flowStatus parameterName="$FLOW_STATUS">ENABLED</flowStatus>
    <!--Optional: -->
    <reservationPriority parameterName="$RES_PRI">0</reservationPriority>
    <!--Optional: -->
```

```
    <rSBandwidth parameterName="$RS_BAND">1000</rSBandwidth>
    <!--Optional: -->
    <rRBandwidth parameterName="$RR_BAND">1000</rRBandwidth>
    <!--0 to 2 repetitions: -->
    <codecData parameterName="$CODEC_DATA">CODEC</codecData>
  </mediaComponentDescription>
  <!--Optional: -->
  <serviceInfoStatus parameterName="$SERV_INFO_STATUS">FINAL_SERVICE_
INFORMATION</serviceInfoStatus>
  <!--Optional: -->
  <chargingIdentifier parameterName="$CHG_ID">test_charging</chargingIdentifier>
  <!--Optional: -->
  <!--Zero or more repetitions: -->
  <subscriptionId>
    <subscriptionIdType parameterName="$SUB_ID_TYPE">END_USER_
E164</subscriptionIdType>
    <subscriptionIdData parameterName="$SUB_ID_
DATA">13693312888</subscriptionIdData>
  </subscriptionId>
  <!--Zero or more repetitions: -->
  <supportedFeatures>
    <vendorId parameterName="$VENDOR_ID">654321</vendorId>
    <featureListID parameterName="$FEATURE_LIST_ID">654320</featureListID>
    <featureList parameterName="$FEATURE_LIST">654322</featureList>
    <!--Zero or more repetitions: -->
    <avp name="test_supported_feature" description="test supported feature avp"
      code="688788" may-encrypt="true" mandatory-flag="required" vendor-id="87349"
      constrained="false">
      <grouped>
        <!--1 or more repetitions: -->
        <gavp name="grouped_avp" />
      </grouped>
      <avps name="avps_name" code="96785">
        <type type-name="String"/>
        <value parameterName="$customer_avps_name">xmf</value>
      </avps>
    </avp>
  </supportedFeatures>
  <!--Optional: -->
  <reservationPriority parameterName="$RESV_PRI">0</reservationPriority>
  <!--Optional: -->
  <framedIPAddress></framedIPAddress>
  <!--Optional: -->
  <framedIPv6Prefix></framedIPv6Prefix>
  <!--Optional: -->
  <calledStationId></calledStationId>
  <!--Optional: -->
  <serviceURN parameterName="$SERV_URN">sos.fire</serviceURN>
  <!--Optional: -->
  <sponsoredConnectivityData>
    <!--Optional: -->
    <sponsorIdentity parameterName="$SPON_ID">spon_id</sponsorIdentity>
    <!--Optional: -->
    <applicationServiceProviderIdentity
      parameterName="$SPON_APP_SERV_PROV_ID">spon_serv_prov_
id</applicationServiceProviderIdentity>
    <!--Optional: -->
    <grantedServiceUnit>
      <!--Optional: -->
      <tariffTimeChange parameterName="$TARIF_TIME_CHG">1</tariffTimeChange>
```

```
      <!--Optional: -->
      <cCTime parameterName="$CC_TIME">60</cCTime>
      <!--Optional: -->
      <cCMoney>
        <unitValue parameterName="$UNIT_VAL">6.28</unitValue>
        <!--Optional: -->
        <currencyCode parameterName="$CUR_CODE">80</currencyCode>
      </cCMoney>
      <!--Optional: -->
      <cCTotalOctets parameterName="$CC_TOTAL_OCTS">1000000</cCTotalOctets>
      <!--Optional: -->
      <cCInputOctets parameterName="$CC_INPUT_OCTS">500000</cCInputOctets>
      <!--Optional: -->
      <cCOutputOctets parameterName="$CC_OUTPUT_OCTS">500000</cCOutputOctets>
      <!--Optional: -->
      <cCServiceSpecificUnits parameterName="$CC_SERV_SPEC_
UNIT">1</cCServiceSpecificUnits>
    </grantedServiceUnit>
  </sponsoredConnectivityData>
  <!--Optional: -->
  <mPSIdentifier parameterName="$MPS_ID">mps_id</mPSIdentifier>
  <!--Zero or more repetitions: -->
  <avp name="myRandomAVP" description="This is a sample AVP" code="93222"
may-encrypt="true" mandatory-flag="required" vendor-id="Oracle Corporation"
constrained="false">
    <type type-name="Integer32"/>
    <enum name="ENUM_1" code="0"/>
    <enum name="ENUM_2" code="1"/>
    <value parameterName="$ENUM_VAL">ENUM_1</value>
  </avp>
</QoSTemplate>
```

Modify QoS

The Modify QoS operation lets you modify the parameters of an existing QoS plan.

Authorization

Basic or OAuth 2.0

HTTP Method

`PUT`

URI

`http://host:port/ApplicationQoSService/qos/${requestId}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *requestId* is a valid QoS request ID.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

To modify a QoS session, create a request body with the parameters you want changed. See the Apply QoS "Request Body" section for a complete listing of request body parameters.

Request Body Example

Example 13–11 shows an example of a modify QoS request body.

Example 13–11 Modify QoS Request Body

```
{
  "qoSFeatureProperties": {
    "applicationIdentifier": "654321"
  }
}
```

Response Header

For details on the response header, see the Apply QoS "Response Header" section.

Response Body

The response body contains an array of structures as the value for **actualProperties**. See "actualProperties" for complete details.

Response Body Example

Example 13–12 shows an example of a modify QoS response body.

Example 13–12 Modify QoS Response Body

```
{
  "actualProperties": {
    "iPCANType": "_3GPP-GPRS",
    "rATType": "WLAN"
  }
}
```

Template-Based Modify QoS

The Template-based Modify QoS operation requests a modification to an existing template-based QoS plan.

Authorization

Basic or OAuth 2.0

HTTP Method

`POST`

URI

`http://host:port/ApplicationQoSService/qos/${requestId}/templatebased`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *requestId* is a valid QoS request ID.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

To modify a template-based QoS session, create a request body with the parameters you want changed. See the Apply Template-based QoS "Request Body" section for a complete listing of request body parameters.

Request Body Example

Example 13–13 shows an example of a modify template-based QoS request body.

Example 13–13 Modify Template-Based QoS Request Body

```
{
  "templateQoSFeatureProperties": {
    "parameter": [
      {
        "name": "$FLOW_DESCRIPTION_0",
        "value": "permit out 8001 from assigned 34 to 24.2.1.6/18 8000"
      }
    ]
  }
}
```

Response Header

For details on the response header, see the Apply QoS "Response Header" section.

Response Body

See "actualProperties" for details on the response body parameters for a Modify Template-based QoS requests.

Response Body Example

Example 13–14 shows an example of a Modify Template-based QoS response body.

Example 13–14 Modify Template-Based QoS Response Body

```
{
  "actualProperties": {
    "iPCANType": "_3GPP-GPRS",
    "rATType": "WLAN"
  }
}
```

Get QoS Status

The Get QoS Status operation returns detailed information on the currently applied QoS plan.

Authorization

Basic or OAuth 2.0

HTTP Method

`GET`

URI

`http://host:port/ApplicationQoSService/qos/${requestId}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *requestId* is a valid QoS session identifier.

Request Header

Standard header fields.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body contains an array of structures as the value for **qoSStatus**. Each element in the array contains values for the following parameters.

- **userIsConnected**. Boolean. True indicates a user is connected. False indicates a user is disconnected.
- **actualProperties**: Complex. Additional properties resulting from the request.

  See "actualProperties" for details on each of the actualProperties parameters.
- **qosFeatureProperties**: Complex. Additional properties resulting from the request.

  See the "Request Body" section of the "Apply QoS" operation for details on each of the qosFeatureProperties parameters.

Response Body Example

Example 13–15 shows an example of a QoS status response body.

Example 13–15 QoS Status Response Body

```
{
  "qoSStatus": {
    "userIsConnected": true,
    "actualProperties": {
      "iPCANType": "_3GPP-GPRS",
      "rATType": "WLAN"
    },
    "qosFeatureProperties": {
      "duration": 3600,
      "applicationIdentifier": "654321",
      "mediaComponentDescription": [
        {
          "mediaComponentNumber": 0,
          "mediaSubComponent": [
            {
              "flowNumber": 1,
              "flowDescription": [
                "flow_1"
              ],
              "flowStatus": "ENABLED-UPLINK",
              "flowUsage": "NO_INFORMATION",
              "maxRequestedBandwidthUL": 100,
              "maxRequestedBandwidthDL": 500,
              "signallingProtocol": "NO_INFORMATION"
            }
          ],
          "applicationIdentifier": "test_app_id",
          "mediaType": "VIDEO",
          "maxRequestedBandwidthUL": 200,
          "maxRequestedBandwidthDL": 1000,
          "minRequestedBandwidthUL": 10,
          "minRequestedBandwidthDL": 100,
          "flowStatus": "ENABLED",
          "reservationPriority": 1
        }
      ],
      "serviceInfoStatus": "FINAL_SERVICE_INFORMATION",
      "chargingIdentifier": "charging_id",
      "subscriptionId": [
        {
          "subscriptionIdType": "END_USER_E164",
          "subscriptionIdData": "88888888"
        }
      ],
      "framedIPAddress": "0A999899",
      "calledStationId": "EE-AA-CD-AF-09"
    }
  }
}
```

Remove QoS

The Remove QoS operation removes a current QoS plan identified by a request ID.

Authorization

Basic or OAuth 2.0

HTTP Method

`DELETE`

URI

`http://host:port/ApplicationQoSService/qos/${requestId}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *requestId* is a valid QoS session identifier.

Request Header

The MIME-type for the Content-Type header field is **application/x-www-form-urlencoded**.

Request Body

There is no request body.

Response Header

For details on the response header, see the Apply QoS "Response Header" section.

Response Body

There is no response body.

Register for QoS Notifications

The Register for QoS Notifications operation lets an application to register to receive QoS events. A QoS event notification is generated by a PCRF when an event is triggered for which an application has registered to receive notifications.

Table 13–2 shows the events an application can register to receive.

Table 13–2 QoS Event Types

Event Type	Trigger Condition
QOS_FEATURE_RELEASED	Triggered when the AF session times out. Controlled by the duration parameter in the QoS session request.
CHARGING_CORRELATION_EXCHANGE	If different Access Network Charging Information is applicable to the IP-CAN session, the PCRF will notify the AF about the Access Network Charging Information that applies to each authorized flow.
INDICATION_OF_LOSS_OF_BEARER	The server reports a loss of a bearer to the AF.
INDICATION_OF_RECOVERY_OF_ BEARER	The server reports a recovery of a bearer to the AF.
INDICATION_OF_RELEASE_OF_ BEARER	The server reports the release of a bearer to the AF.
IP_CAN_CHANGE	The server indicates a change in the IP-CAN type or RAT type (if the IP-CAN type is GPRS).
INDICATION_OF_OUT_OF_CREDIT	The PCRF reports to the AF that the SDFs have run out of credit and that the termination action determined by the finalUnitAction AVP applies.
INDICATION_OF_SUCCESSFUL_ RESOURCES_ALLOCATION	The PCRF reports that the requested resources have been successfully allocated.
INDICATION_OF_FAILED_ RESOURCES_ALLOCATION	The PCRF reports that the requested resources could not be successfully allocated.
INDICATION_OF_LIMITED_PCC_ DEPLOYMENT	The server reports limited PCC deployment—dynamically allocated resources are not available.
USAGE_REPORT	The PCRF reports accumulated usage volume when the usage threshold provided by the AF has been reached.

Authorization

Basic or OAuth 2.0

HTTP Method

POST

URI

`http://`*host*`:`*port*`/ApplicationQoSNotification/registration`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Apply QoS operation accepts the following parameters:

- **reference**: Complex. Required. Defines the application endpoint, interfaceName and correlator used to notify the application.
- **endUserIdentities**. Array. Required. Network end users to be monitored for events. An array of one or more end user IDs.
- **eventCriteria**. Array. Required. One or more events to be monitored. Events are specified by their numeric event ID. Table 13–3 shows the mapping of numeric event IDs to logical event IDs. For definitions of these events, see Table 13–2.

Table 13–3 Numeric to Logical QoS Event Mapping

Numeric Event	Logical Event Name
1	CHARGING_CORRELATION_EXCHANGE
2	INDICATION_OF_LOSS_OF_BEARER
3	INDICATION_OF_RECOVERY_OF_BEARER
4	INDICATION_OF_RELEASE_OF_BEARER
6	IP-CAN_CHANGE
7	INDICATION_OF_OUT_OF_CREDIT
8	INDICATION_OF_SUCCESSFUL_RESOURCES_ALLOCATION
9	INDICATION_OF_FAILED_RESOURCES_ALLOCATION
10	INDICATION_OF_LIMITED_PCC_DEPLOYMENT
11	USAGE_REPORT
12	QOS_FEATURE_RELEASED

reference

These are the parameters for the reference parameter.

- **endpoint**: String. Required. The notification end point expected by the AF.
- **interfaceName**: String. Required. The interface name. Rarely used.
- **correlator**: String. Required. A unique ID for the message.

Request Body Example

Example 13–16 shows an example of a Register for QoS Notifications request body.

Example 13–16 Register for QoS Notifications Request Body

```
{
  "startQoSNotification": {
    "reference": {
      "endpoint": "http://endpt_host:port/jaxrs/QoSNotification",
      "interfaceName": "interfaceName",
      "correlator": "987654321"
    },
    "endUserIdentities": [
```

```
      "tel:88888888", "tel:123456", "tel:234567"
    ],
    "eventCriteria": [
      "1", "2", "6", "7"
    ]
  }
}
```

Response header

For details on the response header, see the Apply QoS "Response Header" section.

Response Body

There is no response body.

Unregister for QoS Notifications

The Unregister for QoS Notifications operation requests that the server cease sending an application QoS notifications.

Authorization

Basic or OAuth 2.0

HTTP Method

`DELETE`

URI

`http://host:port/ApplicationQoSNotification/registration/${correlator}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *correlator* is the unique identifier of the original notification request.

Request Header

The MIME-type for the Content-Type header field is **application/x-www-form-urlencoded**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

QoS Event Notification

A QoS event notification is generated by a PCRF when an event is triggered for which an application has registered to receive notifications.

Authorization

Basic or OAuth 2.0

HTTP Method

`POST`

URI

`http://host:port/QoSNotification`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the QoS Event Notification operation accepts the following parameters:

- **correlator**: String. Required. The duration of the applied QoS in seconds.
- **endUserIdentities**. URI. Required. Identifies the service to which the application-facing service session belongs.
- **eventType**: Complex. Required. The valid routable IPv4 address that is applicable for the IP Flows toward the user equipment at the PCEF.
- **subscriptionId**: Complex. Optional. A valid full IPv6 address that is applicable to an IP flow or IP flows toward the user equipment at the PCEF.
- **accessNetworkChargingIdentifier**: Complex. Optional. Contains a charging identifier within the accessNetworkChargingIdentifierValue AVP along with information about the flows transported within the corresponding bearer within the flows AVP. If no flows AVP is provided, the accessNetworkChargingIdentifierValue applies to all flows within the AF session.

  See "accessNetworkChargingIdentifier" for details on each of the accessNetworkChargingIdentifier parameters.
- **accessNetworkChargingAddress**: String. Optional. Indicates the IP address of the network entity that handles charging within the access network.
- **iPCANType**. Enumerated. Optional. Indicates the type of Connectivity Access Network (CAN) to which a user is connected. The CAN type is specified as one of the following:
  - **_3GPP-GPRS**
  - **DOCSIS**

  - **xDSL**
  - **WiMAX**
  - **_3GPP2**
  - **_3GPP-EPS**
  - **Non-3GPP-EPS**
- **rATType**. Enumerated. Optional. Identifies the Radio Access Technology (RAT) that is servicing the user equipment. The RAT type is specified as one of the following:
  - **WLAN**
  - **VIRTUAL**
  - **UTRAN**
  - **GERAN**
  - **GAN**
  - **HSPA_EVOLUTION**
  - **EUTRAN**
  - **CDMA2000_1X**
  - **HRPD**
  - **UMB**
  - **EHRPD**
- **flows**: Complex. Indicates IP flows using their flow identifiers.

  See "flows" for details on each of the flows parameters.
- **abortCause**. Enumerated. Optional. Determines the cause of an Abort Session Request (ASR) or of a Resource Access Restriction (RAR) indicating a bearer release. The abort cause is specified as one of the following:
  - **BEARER_RELEASED**
  - **INSUFFICIENT_SERVER_RESOURCES**
  - **INSUFFICIENT_BEARER_RESOURCES**
  - **PS_TO_CS_HANDOVER**
  - **SPONSORED_DATA_CONNECTIVITY_DISALLOWED**
- **sponsoredConnectivityData**: Complex. Optional. A set of AVPs that define which template parameters will be replaced and what the replacement values will be.

  See "sponsoredConnectivityData" for details on each of the sponsoredConnectivityData parameters.

Request Example

Example 13–17 shows an example of a QoS Event Notification request body.

Example 13–17 Notify QoS Event Request Body

```
{
  "notifyQoSEvent": {
    "correlator": "987654321",
    "eventType": 6,
```

```
    "accessNetworkChargingAddress": "127.0.0.1",
    "iPCANType": "WiMAX",
    "rATType": "WLAN"
  }
}
```

Response header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

List QoS Event Notifications

The List QoS Event Notifications operation returns a list of QoS events that are registered either to a single correlator, if a correlator is specified in the request, or a complete list of registered events for all correlators if no correlator is specified.

Authorization

Basic or OAuth 2.0

HTTP Method

`GET`

URI

`http://host:port/ApplicationQoSNotification/registration/[${correlator}]`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *correlator* is the optional unique identifier of the original notification request.

Request Header

Standard request headers.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body with No Correlator

The response body for the List QoS Event Notifications operation when an optional correlator parameter is specified, returns the following parameters:

- **correlators**. Array. An array of correlators that are registered to receive QoS events.

Response Body Example

Example 13–18 shows an example of a List QoS Notifications response body when no correlator is specified in the request.

Example 13–18 List QoS Notifications Response Body Without a Correlator

```
{
  "correlators": {
    "correlator": [
      "987654321", "12345676", "1272638"
    ]
  }
```

```
}
```

Response Body with Optional Correlator

The response body for the List QoS Event operation when an optional correlator parameter is specified, returns the following parameters:

- **reference**: Complex. Defines the application endpoint, interfaceName and correlator used to notify the application. See "reference" for detailed information on the reference parameters.
- **endUserIdentities**: Array. Network end users to be monitored for events. An array of one or more end user IDs.
- **eventCriteria**: Array. One or more events to be monitored. Events are specified by their numeric event ID. Table 13–3 shows the mapping of numeric event IDs to logical event IDs. Table 13–2 lists these event IDs.

Table 13–4 Numeric to Logical QoS Event Mapping

Numeric Event	Logical Event Name
1	CHARGING_CORRELATION_EXCHANGE
2	INDICATION_OF_LOSS_OF_BEARER
3	INDICATION_OF_RECOVERY_OF_BEARER
4	INDICATION_OF_RELEASE_OF_BEARER
6	IP-CAN_CHANGE
7	INDICATION_OF_OUT_OF_CREDIT
8	INDICATION_OF_SUCCESSFUL_RESOURCES_ALLOCATION
9	INDICATION_OF_FAILED_RESOURCES_ALLOCATION
10	INDICATION_OF_LIMITED_PCC_DEPLOYMENT
11	USAGE_REPORT
12	QOS_FEATURE_RELEASED

reference

These are the parameters for the reference parameter.

- **endpoint**: String. The notification end point expected by the AF.
- **interfaceName**: String. The interface name. Rarely used.
- **correlator**: String. A unique ID for the message.

Response Body Example

Example 13–19 shows an example of a List QoS Notifications response body when a correlator is specified in the request.

Example 13–19 List QoS Notifications Response Body with Correlator

```
{
  "startQoSNotification": {
    "reference": {
      "endpoint": "http://endpt_host:port/jaxrs/QoSNotification",
      "correlator": "987654321"
    },
    "endUserIdentities": [
```

```
      "tel:88888888"
    ],
    "eventCriteria": [
      "6", "7", "1", "2"
    ]
  }
}
```

14

Adding RESTful Device Capabilities Support

This chapter describes the operations in the Device Capabilities interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Device Capabilities Interface

Applications use the RESTful Device Capabilities interface to request and receive the a terminal's device ID (such as the IMEI) using **getDeviceID**, or receive the devices device ID type, name of the device/model, and a link to the User Agent Profile XML file using **getCapabilities**.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://host:port/rest/device_capabilities/index.html`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Get Capabilities

The Get Capabilities operation retrieves the unique ID for the device type, name of the device/model, and a link to the User Agent Profile XML file.

The request includes the device ID of the device, usually a phone number.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/device_capabilities/device_capabilities?capabilities=${capabilities}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${capabilities}* is a data object which contains the URI (usually, phone number) as the value of the **address** attribute.

  The following JSON data structure represents this data object in the URI. The value part of each name/value pair indicates its data type:

  ```
  {"address": "URI"}
  ```

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The requested device information is returned in a JSON object as the value for the **result** attribute. It contains the following name-value pairs.

- **deviceID**: String. The device/model number for the URI provided in the request.
- **name**: String. The name of the device.
- **userAgentProfileReference**: String. The link to the User Agent Profile XML file.

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": {
  "deviceId": "String",
  "name": "String",
  "userAgentProfileReference": "URI"
}}
```

Examples

Example 14–1 Get Capabilities Request

```
GET rest/device_capabilities/device_capabilities?capabilities=%7B%22address%22
%3A%22tel%3A%221234%22%7D HTTP/1.1
X-Session-ID: app:5198750923966743997
Authorization: Basic YXBwX2luc3RhbmNlXzE6d2VibG9naWM=
User-Agent: Jakarta Commons-HttpClient/3.0
Host: servgtkpr_host.port
```

Example 14–2 Get Capabilities Response

```
HTTP/1.1 200 OK
Date: Fri, Nov 05 2010 05:23:27 GMT
Content-Length=124
Content-Type=application/json
X-Powered-By: Servlet/2.5 JSP/2.1
Host: servgtkpr_host.port

{"result":
      {
        "name":"alice.smith.uiii",
        "deviceId":"0998",
        "userAgentProfileReference":"#1234897897987867gihuyuijlkjlkjllk"
       }
}
```

Example 14–3 Error Response

```
HTTP/1.1 500 Internal Server Error
Date: Fri, Nov 05 2010 05:32:27 GMT
Content-Length=131
Content-Type=application/json
X-Powered-By: Servlet/2.5 JSP/2.1

{"error":
      {
        "message":"Invalid input for message part Address",
        "type":"org.csapi.schema.parlayx.common.v3_1.ServiceException",
       }
}
```

Get Device Id

The Get Device Id operation retrieves the equipment identifier device/name (for example the IMEI number) for a given device. The request includes the device ID, usually a phone number.

The request includes the device ID of the device.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/device_capabilities/device_capabilities?deviceId=${deviceId}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${deviceId}* is a data object which contains the URI (usually, phone number) as the value of the **address** attribute.

  The following JSON data structure represents this data object in the URI. The value part of each name/value pair indicates its data type:

  ```
  {"result": "String"}
  ```

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The required equipment identifier device/name is returned as the value for the **result** attribute. The structure for the response body is

```
{"result": "String"}
```

Examples

Example 14–4 Get Device Id Request

```
GET rest/device_capabilities/device_capabilities?deviceId=%7B%22address%22%3A
%22tel%3A%221234%22%7D HTTP/1.1
X-Session-ID: app:5198750923966743997
Authorization: Basic YXBwX2luc3RhbmNlXzE6d2VibG9naWM=
User-Agent: Jakarta Commons-HttpClient/3.0
Host: servgtkpr_host.port
```

Example 14–5 Get Device Id Response

```
HTTP/1.1 200 OK
Date: Fri, Nov 05 2010 05:34:51 GMT
Content-Length=17
Content-Type=application/json
X-Powered-By: Servlet/2.5 JSP/2.1
Host: servgtkpr_host.port

{"result":"0998"}
```

Example 14–6 Error Response

```
HTTP/1.1 500 Internal Server Error
Date: Fri, Nov 05 2010 05:37:08 GMT
Content-Length=131
Content-Type=application/json
X-Powered-By: Servlet/2.5 JSP/2.1

{"error":
      {
        "message":"Invalid input for message part Address",
        "type":"org.csapi.schema.parlayx.common.v3_1.ServiceException"
       }
}
```

15

Adding RESTful Binary Short Messaging Support

This chapter describes the operations in the Binary Short Messaging (Binary SMS) interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Binary Short Messaging Interface

Applications use the RESTful Binary Short Messaging interface to send any generic binary object attachments to the network using SMS. The supported binary content is broader than the logos and ringtones specified by the Parlay X SMS Web service, extending to other types of binary content such as vCards (a file format standard for electronic business cards).

These interfaces also provide operations to start and stop notifications for SMS messages with binary content.

REST Service Descriptions Available at Runtime

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at:

`http://host:port/rest/binary_sms/index.html`

Where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

RESTful Binary SMS Interface Reference

The RESTful Third Party Call interface includes these operations:

- Send Binary Sms
- Start Binary Sms Notification
- Stop Binary Sms Notification

Send Binary Sms

The Send Binary Sms operation sends an SMS that includes content in binary format.

Authorization

Basic

HTTP Method

POST

URI

`http://host:port/rest/binary_sms/messages`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Send Binary Sms operation accepts the following parameters:

- **addresses**: String. Required. The set of end-user terminal addresses of the recipients as an array of URIs.
- **binaryMessage**. An array of JSON objects. Required. The message to be sent as an array of User Data Header (UDH) elements and message elements. Note that the entire array must be less than of equal to 141 bytes.

  Each element in the array contains:

  - **message**. String in base64Binary. Binary Message data formatted as TP-User Data (TP-UD) excluding the TP-User-Data-Indicator (TP UDHI).
  - **udh**. String in base64Binary. Specifies if the TP-User Data (TP-UD) field contains only the short message, or if it also contains the header formatted as the TP-User-Data-Indicator (TP UDHI).
- **dcs**: Byte. Required. The data-encoding scheme for the binaryMessage parameter.
- **protocolId**. Byte. Required. TP-Protocol-Identifier according to 3GPP 23.040 6.5.0
- **charging**. A JSON object. Optional. This object defines the cost charging properties for the operation. The entry `"charging": null` indicates no charge. If a charge is to be applied, provide values for the following in the **charging** object:
  - **description**: String. Required if the **charging** object is present in the body of the request. The text to be used for information and billing.
  - **amount**. Number (integer, or decimal). Optional. The amount to be charged.
  - **code**: String. Optional. The charging code, from an existing contractual description.
  - **currency**: String. Optional. The currency identifier as defined in ISO 4217 [9].

- **receiptRequest**. a JSON object. Optional. Used to notify the application that the message has been delivered to the terminal, or that delivery is impossible.

  If a delivery receipt is required, provide values for each of the following parameters which define this object:

  - **correlator**: String. Used to correlate the receipt with the initial message.
  - **endpoint**: String. The endpoint address (URI) to which the receipt must be delivered.
  - **interfaceName**: String. A description provided to identify the type of receipt.
- **senderName**: String. Optional. The sender's name.
- **validityPeriod**: String. The validity period of the short message, formatted as a validity-period parameter as described in SMPP v3.4.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "addresses":["URI"],
  "binaryMessage":[{
    "message":"base64Binary"
    "udh":"base64Binary"
  }],
  "dcs":"Byte",
  "charging": {
    "description": "String",
    "amount": "BigDecimal",
    "code": "String",
    "currency": "String"
  }
  "protocolId":"Byte",
  "receiptRequest": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "senderName":"String",
  "validityPeriod":"String",
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body is a JSON object which contains the **result** attribute. The identifier for the delivery request is the string value for **result**.

```
{"result":"String"}
```

Start Binary Sms Notification

The Start Binary SMS Notification operation starts a notification for short messages that contain binary content.

Authorization

Basic

HTTP Method

`PUT`

URI

`http://host:port/rest/binary_sms/notification`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

The request body for the Start Binary SMS Notification operation accepts the following parameters:

- **smsServiceActivationNumber**: String. Required. The destination address, as URI, of the short message.
- **reference**. JSON object. Required. Use this object to provide the following information about the endpoint that is to receive the notification:
  - **correlator**: String. Required. The correlator used to identify the notification.
  - **endpoint**: String. Required. The URI which represents the endpoint address to which the notification should be delivered. This string should be a Bayeux protocol channel name that begins with **`/bayeux/`***`appInstanceID`* where *appInstanceID* is the client application's application instance account ID.

    For more information on managing application instances, see *Services Gatekeeper Portal Developer's Guide*.
  - **interfaceName**: String. Required. A descriptive string to identify the type of notification.

The request body for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "reference": {
    "correlator": "String",
    "endpoint": "URI",
    "interfaceName": "String"
  },
  "smsServiceActivationNumber":"String",
}
```

Response Header

The Location header field contains the URI of the publish/subscribe server:

`http://host:port/rest/binary_sms/notifications`

If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body. The appropriate notifications (described below) are sent to the **endpoint** address provided by the application in the request body of this operation.

Notification of Binary SMS Reception

When a Binary SMS has been received for an application, its designated **endpoint** address on the publish/subscribe server receives a nested JSON object.

This nested JSON object contains the following as the value for the attribute **notifyBinarySmsReception**:

- **correlator**: String. The correlator used in the request body for this operation.
- **message**. a nested JSON object. It contains the data-encoding scheme for the set of messages in the notification, and the messages as an array of User Data Header (UDH) elements and message elements.
  - **dcs**. Byte. The data-encoding scheme for the binary messages included in this object.
  - **message**: Array of JSON objects. Table 15–1 lists the contents of each message object:

Table 15–1 Attributes of Message Object

Attribute	Description
message	Message data as a string base64Binary format
udh	Information specifying the message data format

- **senderAddress**: String. The sender's address, as a URI.
- **smsServiceActivationNumber**: String. The destination address for the binary message, as a URI.
- **dateTime**: String. The date and time the message was received in ISO 8601 extended format, `yyyy-mm-ddThh-mm-ss`.
- **protocolId**. Byte. The TP-Protocol-Identifier according to 3GPP 23.040 6.5.0.

The notification data object delivered to the **endpoint** address is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{"notifyBinarySmsReception": {
  "correlator":"String",
  "message":{
    "dcs":"Byte",
    "message":[{
        "message":"base64Binary"
        "udh":"base64Binary"
```

```
        }],
  "senderAddress":"URI",
  "smsServiceActivationNumber": "URI",
  "dateTime":"Calendar"
  "protocolId":"Byte"
  }
}}
```

Stop Binary Sms Notification

The Stop Binary Sms Notification operation terminates a previously-started notification for messages that contain binary content.

To stop a previously set up binary Sms notification, provide the correlator for the notification passed earlier in the Start Binary Sms Notification request.

There is no request or response body for Stop Binary Sms Notification. If the request fails, the body of the error response will contain the identifier for the notification and the type of exception.

Authorization

Basic

HTTP Method

`DELETE`

URI

The Request-URI used in the DELETE method for Stop Sms Notification is:

`http://`*`host`*`:`*`port`*`/rest/binary_sms/notification/`*`${correlator}`*

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *correlator* is the correlator for the notification provided in the **reference** object of the initial Start Binary Sms Notification request.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

There is no response body.

16

Adding RESTful Session Manager Support

This chapter describes the operations in the Session Manager interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Session Manager Interface

Applications use the RESTful Session Manager interface to get a unique session ID. Each application then adds this session ID to the header of all its requests. Services Gatekeeper uses this value to keep track of all the traffic that an application sends for the duration of the session, and to destroy a session.

The `GeneralException` error will be thrown when any operation in the RESTful Session Manager interface fails.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://`*`host`*`:`*`port`*`/rest/session_manager/index.html`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Get Session

The Get Session operation creates a session with an ID.

If the Get Session operation is successful, the response body will contain the session ID. This string value is used by the application in the X-Session-ID header of all subsequent traffic requests.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/session_manager/sessions`

where *host* and *port* are the hostname and port of the system on which Services Gatekeeper is installed.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body is a JSON object containing the **getSession Return** attribute whose value is the session Id, returned as a string.

```
{"getSessionReturn":"String"}
```

Get Session Remaining Lifetime

The Get Session Remaining Lifetime operation retrieves the time remaining in this session, in milliseconds.

The Request-URI for the GET method contains the session ID.

If the Get Session Remaining Lifetime operation is successful, the response body will contain the time remaining in this session, in milliseconds.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/session_manager/session/${sessionId}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${sessionID}* is the session ID obtained from the response to the Get Session request.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body is a JSON object containing the **getSessionRemainingLifeTimeReturn** attribute whose value is an integer representing the time remaining in this session, in milliseconds.

```
{"getSessionRemainingLifeTimeReturn":"Integer"}
```

Destroy Session

The Destroy Session operation destroys this session.

To destroy a session, provide the appropriate session in the Request-URI for this operation. This identifier should have been obtained by the initial setup for this session.

There is no request or response body for Destroy Session. If the request fails, the body of the error response will contain the call identifier and the type of exception.

Authorization

Basic

HTTP Method

`DELETE`

URI

`http://host:port/rest/session_manager/session/${sessionId}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *sessionId* is the session ID obtained from the response to the Get Session request.

Request Header

The MIME-type for the Content-Type header field is **application/json**.

Request Body

There is no request body.

Response Body

The response body is a JSON object containing the **destroySessionReturn** attribute whose value is a boolean.

```
{"destroySessionReturn":"Boolean"}
```

If the value for **destroySessionReturn** is **true**, the session was destroyed.

17

Adding RESTful Subscriber Profile Support

This chapter describes the operations in the Subscriber Profile interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the Subscriber Profile Interface

Applications use the RESTful Subscriber Profile interface to query an operator's database for individual subscriber profile attributes (such as a user's terminal type) or entire subscriber profiles.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://host:port/rest/subscriber_profile/index.html`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Get

The GET operation retrieves specific subscriber profile properties. The properties that can be accessed are defined in the service provider and application SLAs associated with the application.

To retrieve specific subscriber profile properties, provide the address the subscriber and the required subscriber profile properties within the query object in the Request-URI of the GET method.

If this operation is successful, the response body contains a JSON data object with the required pathnames and values for the required profile properties associated with the specified subscriber.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/subscriber_profile/profile?query=${query}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${query}* is a JSON object.

The parameters accepted in the *${query}* object are:

- **address**: String. Required. The address associated with the subscriber whose data is being accessed. The supported schemes are:
  - **tel id**
  - **imsi** (International Mobile Subscriber Identity)
  - **IPv4**
- **pathNames**: Array of String values. Required. The requested subscriber profile properties expressed as a relative UNIX path. For example, **serviceName/accessControlId/accessControlId**

The *${query}* object in the URI is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
   "address":"URI",
   "pathNames":["String"]
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body is a JSON object which contains the requested properties as an array value for the **properties** attribute. Each element in the array contains the following:

- **pathName**: String. The pathname for the requested property.
- **propertyValue**: String. The value associated with the requested property.

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"properties": [{
  "pathName": "String",
  "propertyValue": "String"
}]}
```

Get Profile

The GET Profile operation retrieves the entire profile for a specific subscriber.

To retrieve the entire profile for a specific subscriber, provide the associated subscriber Id and profile ID within the query object in the Request-URI of the GET method.

If Get Profile is successful, the response body contains a JSON data object with the required pathnames and values for all the profile properties associated with the specified subscriber.

Authorization

Basic

HTTP Method

`GET`

URI

`http://host:port/rest/subscriber_profile/profile?id=${id}`

where:

- *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.
- *${id}* is a JSON object.

The parameters accepted by *${id}* object are:

- **profileID**: String. Required. The ID of the profile which acts as a set of filters limiting the attributes that can be accessed based on the SLAs associated with the application. This entry may be ignored if Services Gatekeeper connects to the network using certain protocols.
- **subscriberID**: String. Required. The ID that uniquely identifies the subscriber whose profile is being accessed.

The *${id}* object in the URI is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
   "profileID":"a_profileId",}
   "subscriberID":"a_subsc_id"
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body is a JSON object which contains the requested properties as an array value for the **result** attribute. Each element in the array contains the following:

- **pathName**: String. The pathname for the requested property.
- **propertyValue**: String. The value associated with the requested property.

Here is the structure:

```
{"result": [{
  "pathName": "String",
  "propertyValue": "String"
}]}
```

18

Adding RESTful WAP Push Support

This chapter describes the operations in the WAP Push interface of the RESTful facade provided in Oracle Communications Services Gatekeeper.

About the WAP Push Interface

Applications use the RESTful WAP Push interface to send a Wireless Application Protocol (WAP) Push message. The content of the message is coded as a Password Authentication Protocol (PAP) message.

The message payload must adhere to the following:

- WAP Service Indication Specification, as specified in Service Indication Version 31-July-2001, Wireless Application Protocol WAP-167-ServiceInd-20010731-a.
- WAP Service Loading Specification, as specified in Service Loading Version 31-Jul-2001, Wireless Application Protocol WAP-168-ServiceLoad-20010731-a.
- WAP Cache Operation Specification, as specified in Cache Operation Version 31-Jul-2001, Wireless Application Protocol WAP-175-CacheOp-20010731-a.

For links to the specifications, see:

`http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html`.

The actual message is sent as an HTTP attachment. See "Headers for Multipart Messages with Attachments" for more information.

REST Service Descriptions Available at Run-time

When the Administration Server for your Services Gatekeeper domain is in the running state, the REST service descriptions of these operations can be found at

`http://`*`host`*`:`*`port`*`/rest/push_message/index.html`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Send Push Message

The Send Push Message operation sends a WAP Push message.

Authorization

Basic

HTTP Method

`POST`

URI

`http://host:port/rest/push_message/messages`

where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Request Header

The request headers depend on the type of message:

- If the message content is in the form of an attachment, the request will be multipart and the request header must contain the header fields that describe the parts of the message.
- If the message does not contain an attachment, the special headers associated with the multipart messaged are not used.

Message Part Content

The message part content for this operation accepts the following parameters:

- **destinationAddresses**: String. Required. An array of terminal addresses.

  Each of the addresses in this array should be formatted according to the Push Proxy Gateway Service Specification (WAP-249-PPGService-20010713-a).
- **pushId**: String. Required. A unique identifier provided by the application. Supported types are PLMN and USER.
- **replaceMethod**: String. Required. Valid entries are:
  - **all**
  - **pendingOnly**

  replaceMethod is used in conjunction with the **pushId** parameter and defines how to replace a previously sent message. This entry will be ignored if **replacePushId** is NULL.
- **requesterId**: String. Required. The application ID as given by the operator.
- **serviceCode**: String. Required. A code for charging purposes.
- **additionalProperties**. An array of JSON objects. Optional.

  Each element in the array is a JSON object used to add additional properties with the following parameters:

  - **name**: String. Required if the `additionalProperties` object is present. Valid entries include **pap.priority**, **pap.delivery-method**, **pap.network-required**, **pap.bearer**, and **pap.bearer-required**.
  - `value`: String. Required if the **additionalProperties** object is present. The value associated with the property.

- **deliverAfterTimeStamp**: String. Optional. The date and time after which the content should be delivered to the wireless device. This entry is in ISO 8601 extended format, as `yyyy-mm-ddThh-mm-ss`.

  If the network does not support this parameter, this entry will be rejected.

- **deliverBeforeTimeStamp**: String. Optional. The date and time by when the content should be delivered to the wireless device. This entry is in ISO 8601 extended format, as `yyyy-mm-ddThh-mm-ss`.

  If the network does not support this parameter, this entry will be rejected.

- **progressNoteRequested**. Boolean. Optional. If **true**, the application wishes to receive progress notes at the address specified by the **resultNotificationEndpoint**.

- **replacePushId**: String. Optional. The Push Id of the message that is to be replaced. If it is set to NULL, the message is treated as a new message. If it is set to a valid ID, message replacement will occur for all currently pending messages. Messages that have already been delivered cannot be canceled, and therefore cannot.

  Messages that have already been delivered cannot be canceled, and therefore cannot be replaced

- **resultNotificationEndpoint**: String. Optional. The URI which represents the endpoint address to which the notification should be delivered. This string should be a Bayeux protocol channel name that begins with **`/bayeux/`***`appInstanceID`* where *appInstanceID* is the client application's application instance account ID.

  If the application does not wish to receive notifications, this value should be NULL.

- **sourceReference**: String. Optional. The name of the service provider.

The message part content for this operation is represented by the following JSON data structure, where the value part of each name/value pair indicates its data type:

```
{
  "destinationAddresses": ["String"],
  "pushId": "String",
  "replaceMethod": "all|pending-only",
  "requesterID": "String",
  "serviceCode": "String",
  "additionalProperties": [{
    "name": "String",
    "value": "String"
  }],
  "deliverAfterTimestamp": "Calendar",
  "deliverBeforeTimestamp": "Calendar",
  "progressNotesRequested": "Boolean",
  "replacePushId": "String",
  "resultNotificationEndpoint": "URI",
  "sourceReference": "String"
}
```

Response Header

Standard header fields. If the request fails, the Status-Line header field will contain the status code and the reason for the failure. See "Understanding RESTful Errors and Exceptions".

Response Body

The response body is a nested JSON object. It contains the **result** attribute with the following attributes and data objects:

- **pushId**: String. Required. A unique identifier provided by the application. Supported types are PLMN and USER.
- **result**: JSON object. This object contains the outcome code. It contains the following two entries:
  - **code**: String. The outcome code generated by the network node. Table 18–1 lists the possible values.
  - **description**: String. The textual description for the code.
- **additionalProperties**: An array of JSON objects used to add additional properties.

  Each element in the array is a JSON object with the following parameters:
  - **name**: String. The name of an additional property. One of: **pap.stage**, **pap.note**, **pap.time**.
  - **value**: String. The value associated with the property.
- **replyTime**: String. The date and time for the reply in ISO 8601 extended format, as `yyyy-mm-ddThh-mm-ss`.
- **senderAddress**: String. Optional. The sender's address.
- **senderName**: String. Optional. The sender's name.

Table 18–1 Possible Outcome Codes

Code	Description
1000	OK
1001	Accepted for processing
2000	Bad request
2001	Forbidden
2002	Address error
2003	Address not found
2004	Push ID not found
2005	Capabilities mismatch
2006	Required capabilities not supported
2007	Duplicate Push ID
2008	Cancellation not possible
3000	Internal server error
3001	Not implemented
3002	Version not supported
3003	Not possible

Table 18–1 (Cont.) Possible Outcome Codes

Code	Description
3004	Capability matching not possible
3005	Multiple addresses not supported
3006	Transformation failure
3007	Specified delivery method not possible
3008	Capabilities not available
3009	Required network not available
3010	Required bearer not available
3011	Replacement not supported
4000	Service failure
4001	Service unavailable

The response body for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"result": {
  "pushId": "String",
  "result": {
    "code": "String",
    "description": "String"
  },
  "additionalProperties": [{
    "name": "String",
    "value": "String"
  }],
  "replyTime": "Calendar",
  "senderAddress": "String",
  "senderName": "String"
}}
```

Result Notification Message Object

The **resultNotificationMessage** object delivered to the **resultNotificationEndpoint** address is a nested JSON object containing the following entries:

- **address**: String. The address of the terminal to which this message is sent.
- **code**: String. The final state of the message, one of the values listed in Table 18–1.
- **messageState**: String. The state of the message. One of:
  - **rejected**
  - **pending**
  - **delivered**
  - **undelivered**
  - **expired**
  - **aborted**
  - **timeout**
  - **cancelled**

  - **unknown**

- **pushId**: String. The unique identifier defined in the initial request, used for correlation.
- **additionalProperties**. An array of JSON objects used to add additional properties.

  Each element in the array is a JSON object with the following parameters:

  - **name**: String. The name of an additional property, dependent on the network node. One of: **pap.priority**, **pap.delivery-method**, **pap.network-required**, **pap.bearer**, and **pap.bearer-required**.
  - **value**: String. The value associated with the property.
- **description**: String. A description of the notification provided by the network. (May not be provided).
- **eventTime**: String. The date and time the message reached the destination in ISO 8601 extended format, as `yyyy-mm-ddThh-mm-ss`.
- **receivedTime**: String. The date and time the message was received at the network node in ISO 8601 extended format, as `yyyy-mm-ddThh-mm-ss`.
- **senderAddress**: String. Optional. The sender's address.
- **senderName**: String. Optional. The sender's name.

The notification of the result for this operation is represented by the following JSON data structure, where the value part of the name/value pair indicates its data type:

```
{"resultNotificationMessage": {
  "address": "String",
  "code": "String",
  "messageState":
"rejected|pending|delivered|undeliverable|expired|aborted|timeout|cancelled|unknow
n",
  "pushId": "String",
  "additionalProperties": [{
    "name": "String",
    "value": "String"
  }],
  "description": "String",
  "eventTime": "Calendar",
  "receivedTime": "Calendar",
  "senderAddress": "String",
  "senderName": "String"
}}
```

Part III

Creating Applications Using the OneAPI RESTful Interfaces

Part III describes how to use the interfaces in the OneAPI facade to create applications that interact with Oracle Communications Services Gatekeeper.

Part III contains the following chapters:

- Using the OneAPI RESTful Interfaces

Also see these chapters in *Services Gatekeeper Communication Service Reference Guide* for details on using the OneAPI-based interfaces:

- OneAPI Multimedia Messaging/MM7
- OneAPI Payment/Diameter
- OneAPI Terminal Location/MLP
- OneAPI Short Messaging/SMPP

19

Using the OneAPI RESTful Interfaces

This chapter provides an overview of how the Oracle Communications Services Gatekeeper RESTful Web services work with OneAPI operations, and how you can use this functionality to interact with Services Gatekeeper.

About the OneAPI Facade Architecture

OneAPI operations are processed by the same network-facing interfaces that are used by the Services Gatekeeper RESTful facade for supported services.

For more information on communication services, see *Services Gatekeeper Communication Service Reference Guide*.

Services Gatekeeper supports OneAPI operations by complying with industry specifications published by the GSMA and the Parlay Group. Services Gatekeeper uses an implementation of Oracle's JSR 311 Jersey Java API for RESTful Web services (JAX-RS) to facilitate communication between applications and network nodes.

See the discussion on OneAPI service facades in *Services Gatekeeper Statement of Compliance* for the specifications supported.

For information about Oracle JAX-RS, see the Oracle Fusion Middleware documentation website:

`http://docs.oracle.com/middleware/1213/index.html`

For information about JSR 311, see the GlassFish website:

`http://jsr311.java.net`

The supported OneAPI services include:

- Multimedia messaging support
- Payment support
- Short Messaging support
- Terminal location support

Support for Anonymous Customer References

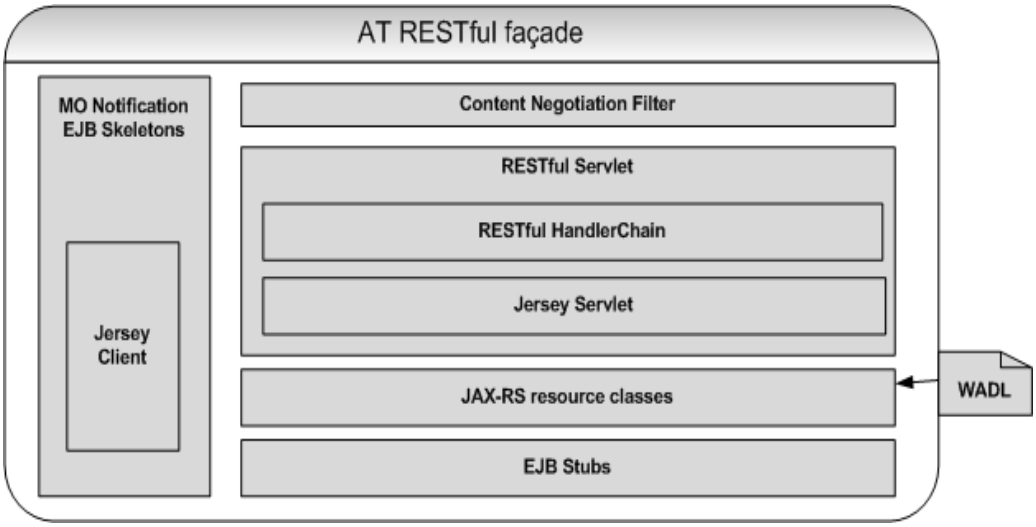
Services Gatekeeper supports the use of Anonymous Customer References (ACRs) for all of the supported OneAPI services it supports. See "Adding RESTful Anonymous Customer Reference Support" for more information on the RESTFUL APIs.

Also see *Services Gatekeeper Statement of Compliance* for a link to the supported specification.

Components of the RESTful Facade

Figure 19–1 shows the multiple components of the Services Gatekeeper Access Tier (AT) RESTful facade.

Figure 19–1 RESTful Facade



Network-facing (application-initiated) requests are received by the RESTful Servlet and then directed to the RESTful handler chain in preparation for delivery by the JAX-RS. The JAX-RS communicates the RESTful requests to the appropriate service enabler for execution at the network tier by the correct Service Enabler Enterprise JavaBean (EJB). Communication between the network-triggered traffic and applications is handled by the same modules in reverse.

See *Services Gatekeeper Communication Service Reference Guide* for information on service enablers.

A Web Application Description Language (WADL) file generates the JAX-RS resource classes used by a communication service to translate between the RESTful request for delivery and the appropriate EJB. For information on WADL, see:

`http://wadl.java.net`

For information on creating communication services from WADL files, see the discussion on Using Services Gatekeeper with REST Services in *Services Gatekeeper Extension Developer's Guide*.

Supported OneAPI Interfaces

Services Gatekeeper supports the following OneAPI Interfaces:

- SMS
- MMS
- Terminal Location
- Payment

There are a number of communication services for which OneAPI does not provide a standard interface. Services Gatekeeper supports these services using the proprietary RESTful interface. See "Creating Applications Using the RESTful Interfaces" for a complete list of supported services is available

The GSMA provides application developers OneAPI specifications for using these services when interacting with a OneAPI server. Services Gatekeeper, acting as a OneAPI server, provides response messages to operations as specified by the OneAPI specification.

SMS

Applications use the OneAPI RESTful Short Messaging interface (**wlng_nt_sms_px21.ear**) over SMPP to send and receive SMS messages, to fetch SMSs and delivery status reports, and to start and stop a notification.

When the request body for an SMS operation contains a request for a delivery receipt, the application provides a correlator for the message being sent and includes an endpoint address for returning the delivery notification.

See "OneAPI Short Messaging/SMPP" for a complete description of the OneAPI RESTful SMS interface.

MMS

Applications use the OneAPI RESTful Multimedia Messaging interface over MM7 (**wlng_nt_multimedia_messaging_px21.ear**) to send a multimedia message (MMS message) and to fetch information on MMS messages that have been received for the applications and stored by Services Gatekeeper.

Applications use the interface to fetch those messages, get delivery status on sent messages, and start and stop a notification.

See "OneAPI Multimedia Messaging/MLP" for a complete description of the OneAPI RESTful MMS interface, including example operations.

Terminal Location

Applications use the OneAPI RESTful Terminal Location interface over MLP (**wlng_nt_terminal_location_px21.ear**) to get a location for an individual terminal or a group of terminals.

See "OneAPI Terminal Location/MLP" for a complete description of the OneAPI RESTful Location interface, including example operations.

Payment

Applications use the OneAPI RESTful Payment interface over Diameter Ro (**wlng_nt_sms_px21.ear**) to charge an amount to an end-user's account, refund amounts to that account, query charge amount status and list charge amount transactions. Applications can also reserve amounts, reserve additional amounts, charge against the reservation and release the reservation.

See "OneAPI Payment/Diameter" for a complete description of the OneAPI RESTful Location interfaces.

About Configuring OneAPI Server Functionality

The Services Gatekeeper OneAPI server functionality is embedded in the Parlay X plug-in, which is installed and deployed by default. No additional OneAPI configuration is necessary.

General Format of an Operation

The following basic elements are present in the requests that an application makes to the OneAPI RESTful interfaces facade, to the responses it receives from the interfaces facade, or to both:

- Request-URI and HTTP Methods in requests and Status-Line in responses
- Headers
- Message Body
- Attachments

Request-URI and HTTP Methods

Applications use one of four methods, "GET", "POST", "PUT", or "DELETE", to request that a required action be performed on an abstract or physical resource, which is identified by a Uniform Resource Identifier (URI). The Request-URI is therefore the most important part of any request that an application makes to the RESTful interfaces.

Here is an example of the OneAPI POST method used to send an SMS message. The request URI is contained in the first line of the example.

```
POST http://example.com/oneapi/1/smsmessaging/outbound/tel%3A%2B5550100/requests
HTTP/1.1
Host: example.com:80
Content-Type: application/x-www-form-urlencoded
Accept: application/json

address=tel%3A%2B15415550100&
address=tel%3A %2B15415550101&
senderAddress=tel:%2B5550100&
message=Hello%20World&
clientCorrelator=123456&
notifyURL=http://application.example.com/notifications/DeliveryInfoNotification&
callbackData=some-data-useful-to-the-requester&
senderName=ACME%20Inc.
```

See “OneAPI Short Messaging/SMPP” for information on the parameters used in this example request.

General Format of a Request-URI

A fully qualified OneAPI Request-URI is made up of a sequence of sections, concatenated as:

`http://`*`host:port`*`/oneapi/`*`api_version/service/service_specific_sections`*

where,

- *host*:*port*: The hostname and port number of your Services Gatekeeper OneAPI installation; for example, **127.0.0.1** and **8001**.
- *api_version*: The version of the OneAPI interface deployed.

  In Services Gatekeeper, the current api version is always **1**.
- *service*: The communication service required by the method. For example, `smsmessaging`.

- *service_specific_sections*: One or more sections required by the OneAPI specification for the method called. See the examples provided in subsequent OneAPI chapters about specific services.

POST

The POST method accesses a resource factory to create a resource that does not yet have a URI. Multiple requests to a resource factory can create multiple new resources.

The following statement sets up a subscription to SMS delivery notifications:

```
POST http://example.com/oneapi/1/smsmessaging/outbound/tel%3A%2B5550100/
subscriptions HTTP/1.1
```

For the POST method:

- The URI in the request represents the factory resource that is accessed to create a resource. In the above example, **/smsmessaging/outbound/tel%3A%2B5550100/subscriptions** is the factory resource accessed to create a resource.
- The request body contains the information required to create the resource.
- If the resource is created, the response body will contain the identifier for the new resource. If the operation fails, the response body will contain the error response.

PUT

The PUT method creates a resource that has a predetermined URI. This method can be used to update a resource or to start a stateful process. For example, an application uses the following statement to release a payment reservation:

```
PUT http://example.com/oneapi/1/payment/tel%3A%2B5550100/transactions/
amountReservation/abc123
```

For the PUT method:

- The URI in the request represents the resource to update or to start a stateful process on. In the example, **/tel%3A%2B5550100/transactions/amountReservation/abc123** represents the resource accessed to release the payment reservation.
- The request body for this operation contains the required information. The JSON object will contain, for example, information on the description and amount of the reservation to release.
- If the operation fails, the response body contains the error response.

GET

The GET method retrieves the state of a specific resource that has been previously created. The specific resource is identified in the query string. For example, this statement can be used by an application to retrieve the location of a terminal whose address is "tel&3A%2B15415550100"::

```
GET http://example.com/oneapi/1/location/queries/location?&address=tel&
3A%2B15415550100&requestedAccuracy=1000
```

For the GET method:

- The URI in the request represents the query string that uniquely identifies the resource whose status the application wishes to retrieve. In the example, the value

for (**location?&address**) is the unique address of the terminal. (It is the address of the terminal, `{"address":"tel:15415550100",}` in JSON representation).

- The request body for this operation is empty.
- The response provides information on the location of the resource with an accuracy of 1000 meters. If the operation fails, it contain the error response.

  In order to complete the operation, the application must access the specified location and use the correlator to retrieve the notification.

DELETE

The DELETE method removes a specified resource. The application provides the correlator or the identifier for the resource to be removed in the Request-URI. An application stops SMS delivery notifications with the following request:

```
DELETE http://example.com/oneapi/1/smsmessaging/outbound/subscriptions/abc123
```

For the DELETE method:

- The URI in the request contains the correlator, which is a value that uniquely identifies the resource the application wishes to remove. In the example, **abc123** is the value which the application provided as the correlator when it requested notifications on a terminal's status.
- The request body for this operation is empty.
- The response body is either empty, or contains the error response if the operation fails.

Headers

The requests and responses for RESTful operations include the following header fields:

- Authorization: The Authorization header field is a required field and is found in all requests. It indicates the type of authentication and security. For example:

  ```
  Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
  ```

  See "Authentication and Security" for more information.
- Accept: OneAPI only supports JSON as the response format. For example:

  ```
  Accept: application/json
  ```

- Location: Certain requests include location headers. They identify a new resource or redirect the recipient to a location other than the Request-URI to complete the request.
- Content-Length: Specifies the length of the request (or response) body.
- Content-Type: The MIME-type value for the Content-Type header field may be **multipart/form-data**, **application/json**, **application/x-www-form-urlencoded** or **application/xml**.

  The **multipart/form-data** value for the Content-Type header field is described in the next section.
- X-Session-ID: A session ID is required if transaction sessions are used.

Headers for Multipart Messages with Attachments

The OneAPI RESTful-based communication services for Multimedia Messaging use HTTP attachments to transport their content. The OneAPI MMS interface supports multipart/form-data POST requests. When you use RESTful interfaces with the base Services Gatekeeper product, multiple attachments are supported in both application-initiated and network-triggered messages.

When a request message contains one or more messages embedded within it, a specified boundary is placed between the parts of the message and at the beginning and end of the message. For multipart message requests:

- The MIME-type value for the Content-Type header field may be **multipart/form-data** or **application/json**. If the MIME-type value for the Content-Type header field is **multipart/form-data**, the **boundary** entry is used to provide a value for the boundary between the message parts.
- Each message part contains the following:
  - Content-Disposition header field whose value is **form-data** and contains a **name** attribute with the appropriate value. For example, the message part name is **messagePart**. For Multimedia Messages the attribute is **attachments**. Multimedia messages contain an additional **filename** attribute and value.
  - Content-Type header field whose value describes the data format included in the message part.
  - Content-Transfer-Encoding field with the appropriate value, if needed.
- If the content of the message is pure ASCII, the response body contains the message. Otherwise, the response body contains an identifier that is used to fetch the message.

Example 19–1 Example of a Multipart Message Request

```
POST http://example.com/oneapi/1/messaging/outbound/ tel%3A%2B5550100/requests
HTTP/1.1
Content-Length: 12345
Content-Type: multipart/form-data;
boundary="===============123456==";

MIME-Version: 1.0
Host: www.example.com
Date: Thu, 04 Jun 2009 02:51:59 GMT

--===============123456==
Content-Disposition: form-data; name="root-fields"
Content-Type: application/x-www-form-urlencoded;
address=tel%3A%2B15415550100&
address=tel%3A%2B15415550101&
senderAddress=tel:%2B5550100&
&senderName=ExampleSender
Subject=My%20message&
notifyURL=http://example-application.com/notifications/DeliveryInfoNotification
/54311
&callbackData=
&clientCorrelator=123456
--===============123456==
Content-Disposition: form-data; name="attachments"; filename="picture.jpg"
Content-Type: image/gif

GIF89a...binary image data...
```

```
----------------123456==-
```

Status Line

The Status Line is the first line in any response that an application receives when it interacts with the OneAPI RESTful services interface in Services Gatekeeper. It takes the form:

```
HTTP/1.1 status_code reason_phrase
```

where:

- *status_code* is a three-digit number that indicates the success or failure to fulfill the request.
- *reason_phrase* is a brief description of the successful action performed; or the reason for the failure.

For example:

```
HTTP/1.1 201 Created
```

Table 19–1 lists some of the status codes and reason-phrases commonly encountered when interacting with the Services Gatekeeper OneAPI RESTful interfaces:

Table 19–1 A Sampling of Status Codes and Reason Phrases

Status Code	Reason Phrase	Description
200	OK	Success
201	Created	Success: The requested resource was created.
204	No Content	Success
400	Bad Request	The request cannot be processed. Check the error message for details.
401	Authentication failure	The authentication to the protected resource failed. Check your Services Gatekeeper OneAPI authentication requirements.
403	Forbidden	The authentication credentials provided in the request are not valid.
404	Not found	The resource specified in the URI cannot be found.
405	Method not supported	The method requested is not supported by the service.
501	Internal Server Error	Failure: An unexpected condition prevented the server from fulfilling the request.
503	Server busy and service unavailable. Please retry the request.	The server hosting Services Gatekeeper OneAPI RESTful interface is busy.

The Status-Codes used by OneAPI and supported by Services Gatekeeper conform to Internet Engineering Task Force (IETF) standards. For a complete listing of the HTTP status codes and their definitions, see RFC 2616 on the IETF website:

http://www.ietf.org/rfc/rfc2616.txt

Message Body

Request or response message bodies are only present if required. A message body is a JSON object.

Request Body

When present, the request body provides additional data required to complete the specific request. The following request body for an example Send SMS operation provides the addresses of the recipients and sending party, and the message text:

Example 19–2 Send SMS Request

```
address=tel%3A%2B15415550100&
address=tel%3A %2B15415550101&
senderAddress=tel:%2B5550100&
message=Hello%20World&
```

Response Body

When present, the response body provides data that the application needs for later action. Example 19–3 shows a response body for the Send SMS operation; it provides the application with the URI for the sent message.

Example 19–3 Response Body for Send SMS Request

```
{"resourceReference": {"resourceURL": "
http://example.com/1/smsmessaging/outbound/ tel%3A%2B5415550100/requests/abc123"}}
```

Example of a Request and Response

Example 19–4 shows an application's request to query the location of a mobile terminal in the Service Gatekeeper OneAPI RESTful interface.

Example 19–4 Request Associated with Get Terminal Location

```
GET
http://example.com/oneapi/1/location/queries/location?&address=tel&3A%2B1541555010
0&requestedAccuracy=1000 HTTP/1.1
Host: example.com:80
Accept: application/json
```

Example 19–5 shows the response which the application receives containing the terminal location information for the requested subscriber.

Example 19–5 Response Associated with a Get Terminal Location Operation

```
HTTP/1.1 200 OKContent-Type: application/json
Content-Type: application/json
Content-Length: 1234
Date: Thu, 04 Jun 2009 02:51:59 GMT

{"terminalLocationList": {"terminalLocation": {
    "address": "tel:15415550100",
    "currentLocation": {
        "accuracy": "100",
        "altitude": "1001.0",
        "latitude": "-80.86302",
        "longitude": "41.277306",
        "timestamp": "2009-06-03T00:27:23.000Z"
```

```
    },
    "locationRetrievalStatus": "Retrieved"
}}}
```

Authentication and Security

The OneAPI RESTful interfaces in Services Gatekeeper use HTTP basic authentication, using username and password. SSL is required. For example:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

For more information on HTTP basic authentication, see RFC 2617 on the IETF website:

http://www.ietf.org/rfc/rfc2617.txt

Services Gatekeeper supports sessionID security with use of the **X-SessionID** header.

Services Gatekeeper also supports subscriber authentication and authorization for access to protected resources by supporting OAuth 2.0. For information on using OAuth 2.0, see *Services Gatekeeper OAuth Guide*.

Notifications

When an application needs to receive a notification, for example about a message delivery receipt, the application posts a subscription request containing a **notificationURL** location where Services Gatekeeper delivers the notification.

Example 19–6 shows a POST operation subscribing to SMS delivery notifications for a **senderAddress**. The notificationURL is included in the message body.

Example 19–6 SMS Delivery Notification Subscription POST Request

```
POST http://example.com/oneapi/1/smsmessaging/outbound/
tel%3A%2B5550100/subscriptions HTTP/1.1
Host: example.com:80
Content-Type: application/x-www-form-urlencoded
Accept: application/json

notifyURL=http://www.yourURL.here&
criteria="SampleCriteria"&
callbackData=doSomething()
```

An application can only subscribe to its own notifications (that is, to the notifications associated with its start notification requests). Any attempt to subscribe to notifications for other applications is rejected.

Errors and Exceptions

The Status-Line in the response message indicates the protocol version, the three-digit status code, and the reason for the failure of the request.

In addition, service exception and policy exception objects are represented in the response body as JSON objects of the following form:

```
{"requestError":
           {
           "policyException":"class name of the error object"
              {
              "messageID": "System Generated ID:
```

```
            "text":"error message"
            "variables""relevant string"
            }
          }
 }
```

For service exceptions, the value for **type** is:

```
"type":"org.csapi.schema.parlayx.common.v2_1.ServiceException"
```

For Policy Exceptions, the value for **type** is:

```
"type":"org.csapi.schema.parlayx.common.v2_1.PolicyException"
```

Part IV

Creating Applications Using the SOAP Interfaces

Part IV describes how to use the interfaces in the SOAP facade and provides details on the interfaces that it supports to create applications that interact with Oracle Communications Services Gatekeeper.

Part IV contains the following chapters:

20

Using the SOAP Interfaces

This chapter presents a high-level description of the SOAP mechanisms and explains how to use this functionality to create services that interact with Oracle Communications Services Gatekeeper (Services Gatekeeper.)

Understanding the SOAP Interfaces

The SOAP-based interfaces are based on the Parlay X standards, and additional Extended Web Services interfaces that cover functionality which is not supported by Parlay X.

See *Services Gatekeeper Concepts* for a complete list of the SOAP-based interfaces supported by Services Gatekeeper. These interfaces are explained in detail in the chapters that follow.

Applications using SOAP-based interfaces must manipulate the SOAP messages that they use to make requests in certain specialized ways.They must add specific information to the SOAP header, and, if they are using for example Multimedia Messaging, they must send their message payload as a SOAP attachment. Applications using the native interfaces use the normal native interface mechanisms, which are not covered in this document. See "Understanding the Extended Web Services Common Definitions" for details.

The mechanisms for dealing with these requirements programmatically depend on the environment in which the application is being developed.

Note: Clients created using Axis 1.2 or older do not work with some communication services. Developers should use Axis 1.4 or newer if they wish to use Axis.

For examples using the Oracle WebLogic Server environment to accomplish these sorts of tasks, see "Managing SOAP Headers and Attachments Programmatically".

See "Adding a SOAP2SOAP Communication Services" for information on how to create and deploy a SOAP2SOAP communication service.

Requirements for Using the SOAP-Based Interfaces

If your application is using the SOAP-based interfaces to interact with Services Gatekeeper, there are four types of elements you may need to add to your application's SOAP messages to Services Gatekeeper.

Understanding SOAP-Based Authentication

In order to secure Services Gatekeeper and the telecom networks to which it provides access, applications are usually required to provide authentication information in every SOAP request which the application submits. Services Gatekeeper leverages the WebLogic Server Web services Security (WS-Security) framework to process this information.

> **Note:** WS-Security provides three modes of providing security between a Web service client application and the Web service itself for message level security: Authentication, Digital Signatures, and Encryption. See "Securing and Administering Web Services" in see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services* for an overview

Services Gatekeeper supports three authentication types:

- Username/Password Authentication (Username Token)
- Digital Signatures (X.509 Certificate Token)
- Encryption (SAML Token)

The type of token that the particular Services Gatekeeper operator requires is indicated in the Policy section of the WSDL files that the operator makes available for each application-facing interface it supports. In the following WSDL fragment, for example, the required form of authentication, indicated by the <wssp:Identity> element, is Username Token.

Example 20–1 WSDL fragment showing Policy

```
<s0:Policy s1:Id="Auth.xml">
<wssp:Identity>
<wssp:SupportedTokens>
<wssp:SecurityToken
TokenType="http://docs.example.com/wss/2004/01/oasis200401wssusernametoken
profile1.0#UsernameToken">
<wssp:UsePassword
Type="http://docs.oasisopen.org/wss/2004/01/oasis200401wssusernametoken
profile1.0#PasswordText"/>
</wssp:SecurityToken>
<wssp:SecurityToken
TokenType="http://docs.oasisopen.org/wss/2004/01/oasis200401wssx509token
profile1.0#X509v3"/>
</wssp:SupportedTokens>
</wssp:Identity>
</s0:Policy>
<wsp:UsingPolicy n1:Required="true"/>
```

> **Note:** If the WSDL also has a `<wssp: Integrity>` element, digital signing is required (WebLogic Server provides WS-Policy: sign.xml). If it has a `<wssp:Confidentiality>` element, encryption is required (WebLogic Server provides WS-Policy: encrypt.xml).

SOAP Header Element for Authentication

Below are examples of the three types of authentication that can be used with Services Gatekeeper.

Username/Password Authentication (Username Token) In the Username Token mechanism, which is specified by the use of the <wsse:UsernameToken> element in the header, authentication is based on a user name, specified in the <wsse:Username> element and a password, specified in the <wsse:Password> element.

Two types of passwords are possible, indicated by the Type attribute in the Password element:

- PasswordText indicates the password is in clear text format.
- PasswordDigest indicates that the sent value is a Base64-encoded, SHA-1 hash of the UTF8 encoded password.

There are two more optional elements in Username Token, introduced to provide a countermeasure for replay attacks:

- <wsse:Nonce>, a random value that the application creates.
- <wsu:Created>, a timestamp.

If either or both the Nonce and Created elements are present, the Password Digest is computed as: Password_Digest = Base64(SHA-1(nonce+created+password))

When the application sends a SOAP message using Username Token, the WSEE implementation in Services Gatekeeper evaluates the username using the associated authentication provider. The authentication provider connects to the Services Gatekeeper database and authenticates the username and the password. In the database, passwords are stored as MD5 hashed representations of the actual password.

Example 20–2 Example of a WSSE: Username Token SOAP header element

```
<wsse:UsernameToken wsu:Id="Example-1">
  <wsse:Username> myUsername </wsse:Username>
  <wsse:Password Type="PasswordText">myPassword</wsse:Password>
  <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
  <wsu:Created> ... </wsu:Created>
</wsse:UsernameToken>
```

The UserName is equivalent to the application instance ID. The Password part is the password associated with this UserName when the application credentials was provisioned in Services Gatekeeper.

For more information on Username Token, see:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

Digital Signatures (X.509 Certificate Token) In the X.509 Token mechanism, the application's identity is authenticated by the use of an X.509 digital certificate.

Typically a certificate binds the certificate holder's public key with a set of attributes linked to the holder's real world identity – for example the individual's name, organization and so on. The certificate also contains a validity period in the form of two date and time fields, specifying the beginning and end of the interval during which the certificate is recognized.

The entire certificate is (digitally) signed with the key of the issuing authority. Verifying this signature guarantees

- that the certificate was indeed issued by the authority in question
- that the contents of the certificate have not been forged, or tampered with in any way since it was issued

For more information on X.509 Token, see:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

The default identity assertion provider in Services Gatekeeper verifies the authenticity of X.509 tokens and maps them to valid Services Gatekeeper users.

Note: While it is possible to use the out-of-the-box keystore configuration in Services Gatekeeper for testing purposes, these should not be used for production systems. The digital certificates in these out-of-the-box keystores are only signed by a demonstration certificate authority For information on configuring keystores for production systems, see the discussion on configuring identity and trust in *Oracle Fusion Middleware Securing Oracle WebLogic Server* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13707/identity_trust.htm

---

The x.509 certificate common name (CN) for an application must be the same as the account user name, which is the string that was referred to as the **applicationInstanceGroupId** in previous versions of Services Gatekeeper. This is provided by the operator when the account is provisioned.

To authenticate using x.509 certificates:

1. Create a web service security configuration.

   For details, see "*Create a Web service security configuration*" in *Oracle WebLogic Server Administration Console Online Help*.

2. Generate key pair for the client :

   ```
   % keytool -genkey -alias client_cert_x509 -keyalg RSA -keysize 1024 -keypass
   ClientKey -keystore client_Identity.jks -storepass ClientKey
   ```

3. Export the self-signed certificate:

   ```
   % keytool -genkey -alias server_cert_x509 -keyalg RSA -keysize 1024 -keypass
   ServerKey -keystore ServerIdentity.jks -storepass ServerKey
   ```

4. Generate the key pair for the server:

   ```
   keytool -genkey -alias server_cert_x509 -keyalg RSA -keysize 1024 -keypass
   ServerKey -keystore ServerIdentity.jks -storepass ServerKey
   ```

5. Export the self-signed certificate:

   ```
   % keytool -export -rfc -alias server_cert_x509 -file ServerCert.cer -keystore
   ServerIdentity.jks -storepass ServerKey
   ```

6. Import the trust certificates:

   ```
   % keytool -import -v -trustcacerts -alias client_cert_x509 -file ClientCert.cer
   .
   -keystore C:\ocsg510ga\JDK160~1\jre\lib\security\cacerts -storepass changeit
   keytool -import -v -trustcacerts -alias server_cert_x509 -file ServerCert.cer .

   -keystore C:\ocsgversion\JDKversion\jre\lib\security\cacerts -storepass
   changeit
   ```

7. Configure the certificates in the Administration Console by pointing the server to the certificates you created in Step 5:

```
C:\ocsgversion\JDKversion\jre\lib\security\ServerCert.cer
```

8. You can use x.509 certificates to establish identity.

   For details, see "*Use X.509 certificates to establish identity*" in *Oracle WebLogic Server Administration Console Online Help*.

Example 20–3 Example of a WSSE: X.509 Certificate SOAP header element

```
<wsse:Security xmlns:wsse="..." xmlns:wsu="...">
  <wsse:BinarySecurityToken wsu:Id="binarytoken"
    ValueType="wsse:X509v3"
    EncodingType="wsse:Base64Binary">
      MIIEZzCCA9CgAwIBAgIQEmtJZc0…
  </wsse:BinarySecurityToken>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:Reference URI="#body">…</ds:Reference>
    <ds:Reference URI="#binarytoken">…</ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>HFLP…</ds:SignatureValue>
    <ds:KeyInfo>
      <wsse:SecurityTokenReference>
        <wsse:Reference URI="#binarytoken" />
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
  </ds:Signature>
</wsse:Security>
```

Encryption (SAML Token) Using WebLogic Server's WSSE implementation, Services Gatekeeper supports SAML versions 1.0, 1.1, and 2.0. The versions are similar.

See "Using SAML Assertions to Access Resources" in *Services Gatekeeper OAuth Guide* for more information on using SAML with OAuth.

For an overview of the differences between versions 1.0 and 1.1, see:

http://www.oasis-open.org/committees/download.php/3412/sstc-saml-diff-1.1-draft-01.pdf

In SAML, a third party, the Asserting Party, provides the identity information for a Subject that wishes to access the services of a Relying Party. This information is carried in an Assertion. In the SAML Token type of Authentication, the Assertion (or a reference to an Assertion) is provided inside the <WSSE:Security> header in the SOAP message. The Relying Party (which in this case is Services Gatekeeper, using the WebLogic Security framework) then evaluates the trustworthiness of the assertion, using one of two confirmation methods.

- Holder-of-Key
- Sender-Voucher

For more information on these confirmation methods, see the discussion on SAML token profile support in *Oracle Fusion Middleware Understanding Security for Oracle WebLogic Server* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13710/archtect.htm

Example 20–4 Example of a WSSE: SAML Token SOAP header element

```
<wsse:Security>
<saml:Assertion MajorVersion="1" MinorVersion="0"
```

```
    AssertionID="186CB370-5C81-4716-8F65-F0B4FC4B4A0B"
    Issuer="www.test.com" IssueInstant="2001-05-31T13:20:00-05:00">
  <saml:Conditions NotBefore="2001-05-31T13:20:00-05:00"
    NotAfter="2001-05-31T13:25:00-05:00"/>
  <saml:AuthenticationStatement AuthenticationMethod="password"
    AuthenticationInstant="2001-05-31T13:21:00-05:00">
    <saml:Subject>
      <saml:NameIdentifier>
        <SecurityDomain>"www.bea.com"</SecurityDomain>
        <Name>"cn=localhost,co=bea,ou=sales"</Name>
      </saml:NameIdentifier>
    </saml:Subject>
  </saml:AuthenticationStatement>
</saml:Assertion>
...
</wsse:Security>
```

Setting Callback Timeout Limits

By default Services Gatekeeper AT server waits 3 seconds to establish a connection with an application endpoint before giving up on the connection, and 30 seconds after establishing the connection before deciding that a reply will never come. The **-Dwlng.ws.callback.connect_timeout** java property setting controls the time limit for requesting a connection and **-Dwlng.ws.callback.read_timeout** controls the reply time limit. Change these settings if your Services Gatekeeper implementation requires different time limits by adding these Java arguments to your *Middleware_Home*`/user_projects/domains/`*domain_name*`/bin/startWebLogic.sh` script:

```
-Dwlng.ws.callback.connect_timeout=time_in_miliseconds
-Dwlng.ws.callback.read_timeout=time_in_miliseconds
```

Understanding How Service Correlation Orchestrates Services

In some cases the service that an application provides to its end-users may involve accessing multiple Services Gatekeeper communication services. For example, a mobile user might send an SMS to an application asking for the pizza restaurant nearest to his current location. The application then makes a Terminal Location request to find the user's current location, looks up the address of the closest pizza restaurant, and then sends the user an MMS with all the appropriate information. Three Services Gatekeeper communication services are involved in executing what for the application is a single service. In order to be able to correlate the three communication service requests, Services Gatekeeper uses a service correlation ID, or SCID. This is a string that is captured in all the CDRs and EDRs generated by Services Gatekeeper. The CDRs and EDRs can then be orchestrated in order to provide special treatment for a given chain of service invocations, by, for example, applying charging to the chain as a whole rather than to the individual invocations.

The SCID is not provided by Services Gatekeeper. When the chain of services is initiated by an application-initiated request, the application must provide, and ensure the uniqueness of, the SCID within the chain of service invocations.

> **Note:** In certain circumstances, it is also possible for a custom service correlation service to supply the SCID, in which case it is the custom service's responsibility to ensure the uniqueness of the SCID.

When the chain of services is initiated by a network-triggered request, Services Gatekeeper calls an external interface to get the SCID. This interface must be

implemented by an external system. No utility or integration is provided out-of-the box; this must be a part of a system integration project. It is the responsibility of the external system to provide, and ensure the uniqueness of, the SCID.

The SCID is passed between Services Gatekeeper and the application through an additional SOAP header element, the SCID element. Because not every application requires the service correlation facility, this is an optional element.

When the scid element is used, it should be on the same level as the session element in the SOAP header.

Example 20–5 Example of a SCID SOAP header element

```
<env:Header>
    <wsse:Security
    . . .
    </wsse:Security>
    <session
    . . .
    </session>
    <scid
    . . .
    </scid>
</env:Header>
```

Understanding Parameter Tunneling

Parameter tunneling is a feature that allows an application to send additional parameters to Services Gatekeeper and lets a plug-in use these parameters. This feature makes it possible for an application to tunnel parameters that are not defined in the application-facing interface and can be seen as an extension to it.

See the discussion on using SOAP parameter tunneling in *Services Gatekeeper Extension Developer's Guide* for more information parameter tunneling.

See the appropriate sections in *Services Gatekeeper Communication Service Reference Guide* for descriptions of the tunneled parameters that are applicable to your communication service.

Understanding SOAP Payload Attachments

In some communication services request payloads are sent as SOAP attachments. Example 20–6 below shows a Multimedia Messaging **sendMessage** operation that contains an attachment carrying a jpeg image.

Example 20–6 Example of a SOAP message with attachment (full content is not shown)

```
POST /parlayx21/multimedia_messaging/SendMessage HTTP/1.1
Content-Type: multipart/related; type="text/xml"; start="<1A07DC767BC3E4791AF25A04F17179EE>";
boundary="----=_Part_0_2633821.1170785251635"
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: localhost:8000
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 4652
Connection: close
------=_Part_0_2633821.1170785251635
Content-Type: text/xml; charset=UTF-8
```

```
Content-Transfer-Encoding: binary
Content-Id: <1A07DC767BC3E4791AF25A04F17179EE>
  <?xml version="1.0" encoding="UTF-8"?>
    <soapenv:Envelope       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Header>
        <ns1:Security ns1:Username="app:-4206293882665579772"
          ns1:Password="app:-4206293882665579772"
          soapenv:actor="wsse:PasswordToken"
          soapenv:mustUnderstand="1"           xmlns:ns1="/parlayx21/multimedia_
messaging/SendMessage">
        </ns1:Security>
      </soapenv:Header>
      <soapenv:Body>
        <sendMessage xmlns=
          "http://www.csapi.org/schema/parlayx/multimedia_messaging/send/v2_4/local">
          <addresses>tel:234</addresses>
          <senderAddress>tel:567</senderAddress>
          <subject>Default Subject Text</subject>
          <priority>Normal</priority>
          <charging>
            <description xmlns="">Default</description>
            <currency xmlns="">USD</currency>
            <amount xmlns="">1.99</amount>
            <code xmlns="">Example_Contract_Code_1234</code>
          </charging>
        </sendMessage>
      </soapenv:Body>
    </soapenv:Envelope>
------=_Part_0_2633821.1170785251635
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-Id:
  <9FFD47E472683C870ADE632711438CC3>???? JFIF      ?? C#%$""!&+7/&)4)!"0A149;>>>%.DIC<H7=>;??
C;("(;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;??  ? w" ??              ?? 7
!1AQ"aq2???#?BRr?3Cb?????             ?? '        !1"AQ2Raq???? ? ??{?????>?"7B?7!1???????Z
e{????ax??5??CC??-Du?
??X?)Y!??=R@??g?????T??c????f?Wc??eCi?l?????5s??\E???6I??(?x?^???=??d?#?itoi?{;? ??G.......
------=_Part_0_2633821.1170785251635--
```

Managing SOAP Headers and Attachments Programmatically

This section illustrates how to manage the Services Gatekeeper required SOAP headers and SOAP attachments when you are using WebLogic Server and WebLogic Server tools to generate stubs for your Web services clients. If you are using a different environment, these steps probably do not apply.

For an overview of using Oracle Fusion Middleware to create Web service clients, see the discussion on Oracle Fusion Middleware in *Oracle Fusion Middleware Introducing Web Services* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/e14294/toc.htm

These examples show how to use a single message handler to add both SOAP Headers and SOAP attachments.

The WebLogic Server environment relies heavily on using supplied Ant tasks. Example 20–7 shows the **clientgen** Ant task added to the standard **build.xml** file.

Example 20–7 Snippet from build.xml

```
<clientgen
  wsdl="${wsdl-file}"
  destDir="${class-dir}"
  handlerChainFile="SOAPHandlerConfig.xml"
  packageName="com.bea.wlcp.wlng.test"
  autoDetectWrapped="false"
  generatePolicyMethods="true"
/>
```

A handler configuration file, **SOAPHandlerConfig.xml** is added as the value for the `handlerChainFile` attribute. Example 20–8 show a **SOAPHandlerConfig.xml** file.

Example 20–8 SOAPHandlerConfig.xml

```
<weblogic-wsee-clientHandlerChain
  xmlns="http://www.example.com/ns/weblogic/90"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:j2ee="http://java.sun.com/xml/ns/j2ee">
  <handler>
    <j2ee:handler-name>clienthandler1</j2ee:handler-name>
    <j2ee:handler-class>
      com.bea.wlcp.wlng.client.TestClientHandler
    </j2ee:handler-class>
  </handler>
</weblogic-wsee-clientHandlerChain>
```

TestClientHandler provides the following functionality:

- Adds a Session ID to the SOAP header, see "Session Management for SOAP, RESTful, and OneAPI Interfaces". The session ID is hardcoded into the member variable `sessionId`.
- Adds a service correlation ID to the SOAP header. See "Understanding How Service Correlation Orchestrates Services" for more information.
- Adds a SOAP attachment in the form of a MIME message with content-type text/plain. See "Understanding SOAP Payload Attachments" for more information.

The configuration file for the message handler contains the handler-name and the associated handler-class. The handler class, **TestClientHandler**, is shown in Example 20–9.

Example 20–9 TestClientHandler

```
package com.example.wlcp.wlng.client;
import javax.xml.rpc.handler.Handler;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.*;
import javax.xml.namespace.QName;
public class TestClientHandler implements Handler{
  public String sessionId = "myID";
  public String SCID = "mySCId";
  public String contenttype = "text/plain";
  public String content = "The content";

  public boolean handleRequest(MessageContext ctx) {
    if (ctx instanceof SOAPMessageContext) {
```

```
      try {
        SOAPMessageContext soapCtx = (SOAPMessageContext) ctx;
        SOAPMessage soapmsg = soapCtx.getMessage();
        SOAPHeader header = soapCtx.getMessage().getSOAPHeader();
        SOAPEnvelope envelope =
          soapCtx.getMessage().getSOAPPart().getEnvelope();
        // Begin: Add session ID
        Name headerElementName = envelope.createName("session","",
          "http://schemas.xmlsoap.org/soap/envelope/");
        SOAPHeaderElement headerElement =
          header.addHeaderElement(headerElementName);
        headerElement.setMustUnderstand(false);
        headerElement.addNamespaceDeclaration("soap",
          "http://schemas.xmlsoap.org/soap/envelope/");
        SOAPElement sessionId = headerElement.addChildElement("SessionId");
        sessionId.addTextNode(sessionId);
        // End: Add session ID
        // Begin: Add Combined Services ID
        Name headerElementName = envelope.createName("SCID","",
          "http://schemas.xmlsoap.org/soap/envelope/");
        SOAPHeaderElement headerElement =
          header.addHeaderElement(headerElementName);
        headerElement.setMustUnderstand(false);
        headerElement.addNamespaceDeclaration("soap",
          "http://schemas.xmlsoap.org/soap/envelope/");
        SOAPElement sessionId = headerElement.addChildElement("SCID");
        sessionId.addTextNode(SCID);
        // End: Add Combined Services ID
        // Begin: Add SOAP attachment
        AttachmentPart part = soapmsg.createAttachmentPart();
        part.setContent(content, contenttype);
        soapmsg.addAttachmentPart(part);
        // End: Add SOAP attachment
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    return true;
  }
  public boolean handleResponse(MessageContext ctx) {
    return true;
  }
  public boolean handleFault(MessageContext ctx) {
    return true;
  }
  public void init(HandlerInfo config) {
  }
  public void destroy() {
  }
  public QName[] getHeaders() {
    return null;
  }
}
```

21

Adding a SOAP2SOAP Communication Services

This chapter explains information about creating Oracle Communications Services Gatekeeper SOAP2SOAP communication services.

About SOAP2SOAP Communication Services

You use the Partner and API Management Portal to create SOAP2SOAP APIs and services. See "PRM Portals and the Application Development Process" in *API Management Guide* for information.

See "Adding a SOAP2SOAP Communication Service" in *Extension Developer's Guide* for details on about the SOAP2SOAP plugin, and the artifacts and properties it contains.

22

Adding SOAP-Based Quality of Service Support

This chapter describes how to add Quality of Service (QoS) support to SOAP-based applications for use with Oracle Communications Services Gatekeeper.

About the SOAP-Based QoS Interface

An application can use the SOAP-based QoS interface to apply a QoS policy, to query, modify and remove that policy, and to register as well as unregister for QoS-related notifications. A Policy Control and Charging Rules Function provider (PCRF) can also return QoS events to registered applications.

SOAP-Based Service Descriptions Available at Run-time

The WADL file to for SOAP-based QoS services is located at **http://***at_host***:***at_port***/parlayx40/qos/ApplicationQoS?WSDL**.

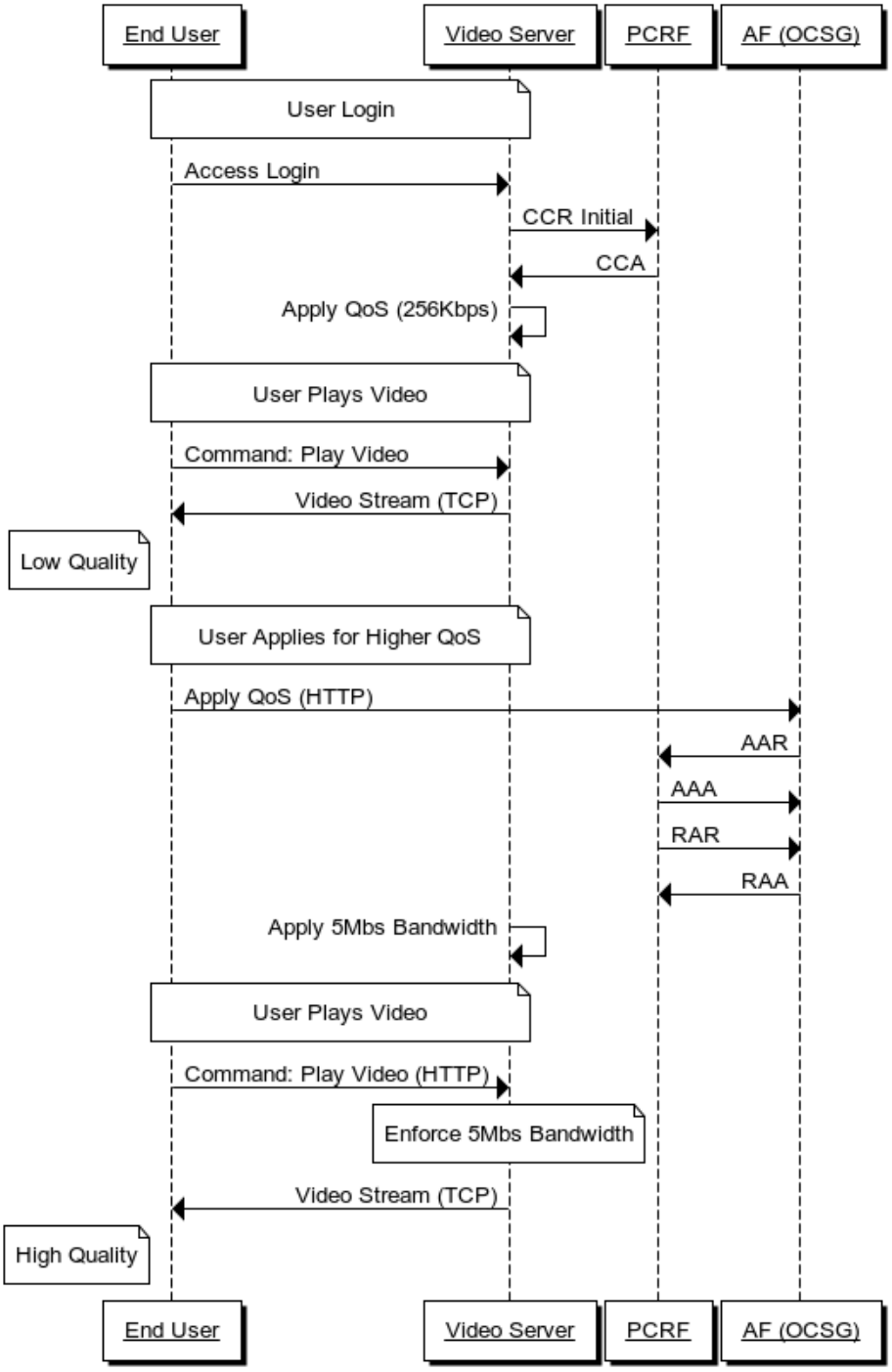
Example Parlay X 4.0 Application-Driven QoS/Diameter Scenario

A typical QoS scenario involves a subscriber using a handset to access a video feed using a video application installed on the handset. Initially, because the default QoS is set to a low bandwidth, the video stops and stutters frequently as it is buffered repeatedly over the low speed connection. The subscriber requests a faster QoS through the application, presumably with a corresponding billing charge. Services Gatekeeper forwards that request to a PCRF which then applies the upgraded QoS. The subscriber's video now streams at the upgraded speed, without stuttering.

Note: While QoS frequently refers to raw bandwidth speed, it can apply to any factors that affect network performance, for example, connection latency and time-out.

Figure 22–1 shows a detailed QoS call flow sequence.

Figure 22–1 Example Parlay X 4.0 QoS Call Sequence



Configuring Services Gatekeeper to Use the QoS Communication Services

Before you can implement QoS functionality, a QoS plug-in must be deployed and configured in Services Gatekeeper. For information on deploying and configuring QoS plug-ins, see the discussion on Parlay X 4.0 Application-driven Quality of Service (QoS) in *Services Gatekeeper Communication Service Reference Guide*.

23

About the Supported SOAP Parlay X 2.1 Facades

This chapter describes the Oracle Communications Services Gatekeeper interfaces in the supported Parlay X 2.1 facades and contains information specific to Services Gatekeeper. For detailed descriptions of the interfaces, methods, and parameters, see the ETSI OSA Parlay X website for links to the specifications:

http://www.etsi.org/deliver/etsi_es/202300_202399/20239102/01.02.01_60/es_20239102v010201p.pdf

Parlay X 2.1 Part 2: Third Party Call

This interface is compliant with ETSI ES 202 391-2 V1.2.1 (2006-12) Open Service Access (OSA); Parlay X Web services; Part 2: Third Party Call (Parlay X 2).

Interface: ThirdPartyCall

The endpoint for the **ThirdPartyCall** interface is:
`http://host:port/parlayx21/third_party_call/ThirdPartyCall`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

MakeCall

Sets up a call between two parties.

GetCallInformation

Displays information about a call.

EndCall

Ends a call.

CancelCall

Cancels a call setup procedure.

Error Codes

See General Error Codes.

Parlay X 2.1 Part 3: Call Notification

This set of interfaces is compliant with ETSI ES 202 391-3 V1.2.1 (2006-12) Open Service Access (OSA); Parlay X Web services; Part 3: Call Notification (Parlay X 2).

Interface: CallDirection

This interface is implemented by an application, and the consumer of this interface is Services Gatekeeper. The Web services Description Language (WSDL) file that defines the interface can be downloaded from:

`http://host:port/parlayx21/call_notification/wsdls/parlayx_call_direction_interface_2_2.wsdl`

`http://host:port/parlayx21/call_notification/wsdls/parlayx_call_direction_service_2_2.wsdl`

`http://host:port/parlayx21/call_notification/wsdls/parlayx_call_notification_types_2_2.xsd`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

HandleBusy

Services Gatekeeper calls this method, which is implemented by an application, when the called party is busy.

HandleNotReachable

Services Gatekeeper calls this method, which is implemented by an application, when the called party is not reachable.

HandleNoAnswer

Services Gatekeeper calls this method, which is implemented by an application, when the called party does not answer.

HandleCalledNumber

Services Gatekeeper calls this method, which is implemented by an application, before call setup.

Interface: CallNotification

The **CallNotification** interface is implemented by an application, and the consumer of this interface is Services Gatekeeper. The WSDL that defines the interface can be downloaded from:

`http://host:port/parlayx21/call_notification/wsdls/parlayx_call_notification_interface_2_2.wsdl`

`http://host:port/parlayx21/call_notification/wsdls/parlayx_call_notification_service_2_2.wsdl`

`http://host:port/parlayx21/call_notification/wsdls/parlayx_call_notification_types_2_2.xsd`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

NotifyBusy

Services Gatekeeper calls this method, which is implemented by an application, when the called party is busy.

NotifyNotReachable

Services Gatekeeper calls this method, which is implemented by an application, when the called party is not reachable.

NotifyNoAnswer

Services Gatekeeper calls this method, which is implemented by an application, when the called party does not answer.

NotifyCalledNumber

Services Gatekeeper calls this method, which is implemented by an application, before call setup.

Interface: CallNotificationManager

The endpoint for the **CallNotificationManager** interface is:
`http://host:port/parlayx21/call_notification/CallNotificationManager`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

StartCallNotification

Starts a subscription for call notifications.

StopCallNotification

Stops a subscription for call notifications.

Interface: CallDirectionManager

The endpoint for the **CallDirectionManager** interface is:
`http://host:port/parlayx21/call_notification/CallDirectionManager`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

StartCallDirectionNotification

Starts a subscription for call direction notifications.

StopCallDirectionNotification

Stops a subscription for call direction notifications.

Error Codes

See General Error Codes.

Parlay X 2.1 Part 4: Short Messaging

This set of interfaces is compliant with ETSI ES 202 391-4 V1.2.1 (2006-12) Open Service Access (OSA); Parlay X Web services; Part 4: Short Messaging (Parlay X 2).

Interface: SendSms

The endpoint for the **SendSms** interface is:
`http://host:port/parlayx21/sms/SendSms`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

If a backwards-compatible communication service is used:

- The **senderAddress** parameter is either of the format `tel:`*mailbox ID*`\`*mailbox password*`\`*Sender name* or just sender name depending on how the application was provisioned in Services Gatekeeper.
- The priority parameter is not supported.

SendSms

Sends an SMS to one or more destinations.

SendSmsLogo

Sends an SMS Logo to one or more destinations.

Logos in SmartMessaging and EMS are supported. The image is not scaled.

Logos in the following raw image formats are supported:

- bmp
- gif
- jpg
- png

The logos are in pure black and white (gray scale not supported). Animated images are not supported. Scaling is not supported.

If the logo is converted to SmartMessaging format, the image cannot be larger than 72x14 pixels.

If the logo is sent in EMS format, the following rules apply:

- If the image is 16x16 pixels, the logo is sent as an EMS small picture.
- If the image is 32x32 pixels, the logo is sent as an EMS large picture.
- If the image is of any other size, the logo is sent as an EMS variable picture.
- Images up to 1024 pixels are supported.

SendSmsRingtone

Sends an SMS Ringtone to one or more destinations.

Ringtones can be in any of these formats:

- RTX
- SmartMessaging
- EMS (iMelody)

GetSmsDeliveryStatus

Gets the delivery status of a previously sent SMS.

It is possible to query delivery status of an SMS only if a callback reference was not defined when the SMS was sent. If a callback reference was defined, NotifySmsDeliveryReceipt is invoked by Services Gatekeeper and the delivery status is not stored. If the delivery status is stored in Services Gatekeeper, it is stored for a configurable period.

Interface: SmsNotification

This **SmsNotification** interface is implemented by an application, and the consumer of this interface is Services Gatekeeper. The WSDL file that defines the interface can be downloaded from:

`http://host:port/parlayx21/sms/wsdls/parlayx_sms_notification_interface_2_2.wsdl`

`http://host:port/parlayx21/sms/wsdls/parlayx_sms_notification_service_2_2.wsdl`

`http://host:port/parlayx21/sms/wsdls/parlayx_sms_types_2_2.xsd`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

NotifySmsReception

Sends an SMS that is received by Services Gatekeeper to an application given that the SMS fulfills a set of criteria. The criteria is either defined by the application itself, using **startSmsNotification** or defined using a provisioning step in Services Gatekeeper.

Shortcode translation, if appropriate, is applied.

NotifySmsDeliveryReceipt

Sends a delivery receipt that a previously sent SMS has been received by its destination. The delivery receipt is propagated to the application given that the application provided a callback reference when sending the SMS.

Interface: ReceiveSms

The endpoint for the **ReceiveSms** interface is:
`http://host:port/parlayx21/sms/ReceiveSms`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

GetReceivedSms

Gets messages that have been received by Services Gatekeeper. The SMS messages are retrieved using a registrationIdentifier used when the notification was registered using a provisioning step in Services Gatekeeper.

The maximum number of messages returned for one operation is controlled by the **maxBatchSize** attrribute of the **SmsMBean**. The default is 20. If you attempt to exceed the **maxBatchSize** value for an operation, Services Gatekeeper throws a policy exception. For details see **SmsMBean** in the "All Classes" section of the *Services Gatekeeper OAM Java API Reference* documentation.

Interface: SmsNotificationManager

The endpoint for the **SmsNotificationManager** interface is:
`http://host:port/parlayx21/sms/SmsNotificationManager`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

StartSmsNotification

Initiates notifications to the application for a given service activation number and criteria.

Note: A Service activation number may be provisioned to handle a range of numbers using short code translations.

Note: The equivalent of this operation may have been performed as an off-line provisioning step by the Services Gatekeeper administrator.

StopSmsNotification

Ends a previously started notification.

Sending Custom Message Content for Split and Submit Messaging Requests

With Split and Submit Messaging enabled, short messages addressed to many recipients are split into multiple individually-addressed requests by Services Gatekeeper. For information on enabling Split and Submit Messaging, see the discussion on **supportBulkRequest** in *Services Gatekeeper System Administrator's Guide*. For an overview of Split and Submit Messaging, see *Services Gatekeeper Communication Service Reference Guide*.

Using DifferentContentForSingleAddressInBulk to Customize Split Messages

You can provide custom content to be sent to particular addresses. To do this, specify the per-address content in a `content` attribute for each message element with custom content. Include an xparam `DifferentContentForSingleAddressInBulk`, set to true, with the SOAP request.

Each address in the bulk SMS request must have a content attribute if `DifferentContentForSingleAddressInBulk` is set to true.

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<messages>
  <message address="1100" content="1100abc"/>
  <message address="1200" content="1200abc"/>
  <message address="2100" content="2100abc"/>
  <message address="3100" content="3100abc"/>
</messages>
```

The schema is as follows:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.oracle.com/ocsg/51"
xmlns:wlng="http://www.oracle.com/ocsg/51">
  <xsd:complexType name="messages">
```

```
    <xsd:sequence>
    <xsd:element name="message" type="wlng:message" minOccurs="1"
maxOccurs="unbounded"/>
  </xsd:complexType>
  <xsd:complexType name="message">
    <xsd:attribute name="address" type="xsd:string" use="required"/>
    <xsd:attribute name="address" type="xsd:string" use="required"/>
    <xsd:attribute name="content" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

Table 23–1 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0002	n/a	Invalid input value for message part %1 Log message: Parse bulk SMS XML message content failed: This log message indicates that xparam `DifferentContentForSingleAddressInBulk` in the request is set to true but the XML message content is not well-formed. Bulk SMS XML message cannot find content for address *dest_address*: This log message indicates that xparam `DifferentContentForSingleAddressInBulk` in the request is set to true and the XML is well-formed, but not every address has content provided for it. The first such address encountered is the one reported.

Error Codes

See General Error Codes.

Parlay X 2.1 Part 5: Multimedia Messaging

This set of interfaces is compliant with ETSI ES 202 391-5 V1.2.1 (2006-12) Open Service Access (OSA); Parlay X Web services; Part 5: Multimedia Messaging (Parlay X 2).

See "Sending Custom Message Content for Split and Submit Messaging Requests" for instructions on how to split messages into multiple individually-addressed requests

Interface: SendMessage

The endpoint for the **SendMessage** interface is:
`http://host:port/parlayx21/multimedia_messaging/SendMessage`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

SendMessage

Sends a multimedia message. The content of the message is sent as a SOAP attachment. Sending as email is not supported.

Table 23–2 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.

Table 23–2 (Cont.) Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	MMS-000001	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0001	MMS-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0001	MMS-000003	Address is utilizing an unsupported address type.
SVC0001	MMS-000005	Message could not be delivered to MMSC.

GetMessageDeliveryStatus

Gets the delivery status of a previously sent MMS.

It is possible to query delivery status of an MMS only if a callback reference was not defined when the message was sent. If a callback reference was defined, NotifyMessageDeliveryReceipt is invoked by Services Gatekeeper and the delivery status is not stored. If the delivery status is stored in Services Gatekeeper, it is stored for a configurable period.

Note: Storing delivery status for an MMS is configurable in Services Gatekeeper.

Table 23–3 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0002	*RequestIdentifier*	Message is not found.

Interface: ReceiveMessage

The endpoint for this interface is: `http://host:port/parlayx21/multimedia_messaging/ReceiveMessage`

Where the values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

GetReceivedMessages

Polls Services Gatekeeper for received messages.

The registrationIdentifier is required. Received message are stored in Services Gatekeeper only for a configurable period.

Table 23–4 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0002	MMS-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.

GetMessageURIs

Not supported.

GetMessage

Gets a specific message that was received by Services Gatekeeper and belongs to the application.

Table 23–5 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0001	MMS-000004	Correlator does not exist, no `notification` corresponds to the correlator.

Interface: MessageNotification

This interface is implemented by an application, and the consumer of this interface is Services Gatekeeper. The WSDL that defines the interface can be downloaded from:

`http://host:port/parlayx21/multimedia_messaging/wsdls/parlayx_mm_notification_interface_2_4.wsdl`

`http://host:port/parlayx21/multimedia_messaging/wsdls/parlayx_mm_notification_service_2_4.wsdl`

`http://host:port/parlayx21/multimedia_messaging/wsdls/parlayx_mm_types_2_4.xsd`

Where the values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

NotifyMessageReception

Sends a `notification` to an application that an MMS destined for the application is received by Services Gatekeeper.

When an application is notified about an incoming MMS message by an asynchronous call to **NotifyMessageReception** from Services Gatekeeper, the application receives a MessageReference structure containing information about the message.

If the MMS message contains only pure text, the `<message>` element of the MessageReference structure contains the entire text content as an ASCII string, and the message is not stored in Services Gatekeeper. If the message contains any content that is not pure text, such as an image, sound or video, the MessageReference structure does not include a `<message>` element, but instead includes a `<messageIdentifer>` element that contains a reference to the message stored in Services Gatekeeper. For more information about the MessageReference structure, see the Parlay X Web services Part 5: Multimedia Messaging specification.

NotifyMessageDeliveryReceipt

Sends a `notification` to an application that a previously sent MMS has been delivered to its destination.

Note: Supporting delivery notifications is optional for Services Gatekeeper.

Interface: MessageNotificationManager

The endpoint for the **MessageNotificationManager** interface is:
`http://host:port/parlayx21/multimedia_messaging/MessageNotificationManager`

Where the values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

StartMessageNotification

Initiates notifications to the application for a given service activation number and criteria.

Table 23–6 StartMessageNotification Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.

Note: A service activation number may be provisioned to handle a range of numbers using short code translations.

Note: The equivalent to this operation may have been performed as an off-line provisioning step by the Services Gatekeeper administrator.

Table 23–7 StartMessageNotification Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.

StopMessageNotification

Ends a previously started notification.

Table 23–8 StopMessageNotification Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0002	*correlator*	Correlator does not exist, no notification corresponds to the correlator.

Error Codes

See General Error Codes.

Parlay X 2.1 Part 9: Terminal Location

This set of interfaces is compliant with ETSI ES 202 391-9 V1.2.1 (2006-12), Open Service Access (OSA); Parlay X Web services; Part 9: Terminal Location (Parlay X 2).

Understanding Parlay X 2.1 Terminal Location Precision

The Parlay X 2.1 Terminal Location specification complies with the ISO 6709 [7] specification, specifying decimal values for longitude and latitude locations. The decimals representing longitude are floating point numbers in the range -90.0000 to +90.0000. Positive values for locations North of the equator and negative values for South of the equator. Longitude values are floating point numbers in the range -180.0000 to +180.0000. Positive values are East of the prime meridian, and negative values are West of the prime meridian.

It is important to understand that Services Gatekeeper rounds these floating point values to 8 significant digits. For example, if Services Gatekeeper receives messages with these longitude (X) and latitude (Y) values from the network:

```
"latitude":"12.34567890",
"longitude":"123.4567890",
```

Services Gatekeeper reports these rounded values to the application:

```
 <X> 12.345679 </X>
 <Y> 123.45679 </Y>
```

Interface: TerminalLocation

The endpoint for the **TerminalLocation** interface is:
`http://`*`host`*`:`*`port`*`/parlayx21/terminal_location/TerminalLocation`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

GetLocation

Gets the location of a terminal.

Table 23–9 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	TL-000007	There is a communication problems between Services Gatekeeper and the network node. Contact your Services Gatekeeper administrator.
SVC0001	TL-000010	There is a communication problem between Services Gatekeeper and the network node, and Services Gatekeeper was unable to interpret the response. Contact your Services Gatekeeper administrator.
SVC0001	TL-000009	No location data was received from network.
SVC0001	TL-000011	An unknown error was received from the network.
SVC0002	N/A	An invalid parameter was included in the terminal status request.
SVC0200	N/A	The location accuracy is invalid.
POL0001	N/A	General policy error.
POL0002	N/A	Privacy error.
POL0230	N/A	The requested accuracy is not supported.

GetTerminalDistance

Gets the distance from a certain point to the location of a terminal.

Table 23–10 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	TL-000007	There is a communication problems between Services Gatekeeper and the network node. Contact your Services Gatekeeper administrator.
SVC0001	TL-000010	There is a communication problem between Services Gatekeeper and the network node, and Services Gatekeeper was unable to interpret the response. Contact your Services Gatekeeper administrator.
SVC0001	TL-000009	No location data was received from the network.
SVC0001	TL-000011	An unknown error was received from the network.
SVC0002	N/A	An invalid parameter was included in the terminal status request.
SVC0200	N/A	The location accuracy is invalid.
POL0001	N/A	General policy error.
POL0002	N/A	Privacy error.
POL0230	N/A	The requested accuracy is not supported.

GetLocationForGroup

Gets the location for one or more terminals.

Table 23–11 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	TL-000007	There is a communication problems between Services Gatekeeper and the network node. Contact your Services Gatekeeper administrator.
SVC0001	TL-000010	There is a communication problem between Services Gatekeeper and the network node, and Services Gatekeeper was unable to interpret the response. Contact your Services Gatekeeper administrator.
SVC0001	TL-000009	No location data was received from the network.
SVC0001	TL-000011	An unknown error was received from the network.
SVC0002	N/A	An invalid parameter was included in the terminal status request.
SVC0004	N/A	No valid addresses were passed in on the request.
SVC0200	N/A	The location accuracy is invalid.
POL0001	N/A	General policy error.
POL0002	N/A	Privacy error.
POL0230	N/A	The requested accuracy is not supported.

Interface: TerminalLocationNotificationManager

The endpoint for the **TerminalLocationNotificationManager** interface is:
`http://host:port/parlayx21/terminal_location/TerminalLocationNotificationManager`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

StartGeographicalNotification

Initiates location notifications to the application when one or more terminals change their location according to a criteria.

Table 23–12 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	TL-000003	Unable to start the geographical notification due to a network error. Contact your Services Gatekeeper administrator.
SVC0001	TL-000004	Unable to start geographical notification due to an internal error. Contact your Services Gatekeeper administrator.
SVC0002	N/A	The request included an invalid parameter.
SVC0004	N/A	The request did not include any valid addresses.
SVC0005	N/A	The correlator used already exists.
POL0001	N/A	General policy error.

StartPeriodicNotification

Initiates location notifications to the application on a periodic basis.

Table 23–13 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	TL-000005	Unable to start periodic notification due to a network error. Contact your Services Gatekeeper administrator.
SVC0001	TL-000006	Unable to start periodic notification due to an internal error. Contact your Services Gatekeeper administrator.
SVC0002	N/A	The request included an invalid parameter.
SVC0004	N/A	The request did not include any valid addresses.
SVC0005	N/A	The correlator used already exists.
POL0001	N/A	General policy error.

EndNotification

Ends a previously started notification.

Table 23–14 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	TL-000001	Unable to start geographical notification due to a network error. Contact your Services Gatekeeper administrator.
SVC0001	TL-000002	Unable to start geographical notification due to an internal error. Contact your Services Gatekeeper administrator.
SVC0002	N/A	The request included an invalid parameter.
POL0001	N/A	General policy error.

Interface: TerminalLocationNotification

The **TerminalLocationNotification** interface is implemented by an application, and the consumer of this interface is Services Gatekeeper. The WSDL that defines the interface can be downloaded from:

`http://host:port/parlayx21/terminal_location/wsdls/parlayx_terminal_location_notification_interface_2_2.wsdl`

`http://host:port/parlayx21/terminal_location/wsdls/parlayx_terminal_location_notification_service_2_2.wsdl`

`http://host:port/parlayx21/terminal_location/wsdls/parlayx_terminal_location_types_2_2.xsdl`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

LocationNotification

Notifies an application about a change of location for a terminal.

LocationError

Notifies an application that the subscription for location notifications was cancelled by Services Gatekeeper.

LocationEnd

Notifies an application that no more location notifications are being sent to the application.

Error Codes

See "General Error Codes".

About Notifications

After an application starts notification, the notification persists. If an application has started a notification and destroys the session, the notification remains registered and matching notifications are sent to the application when it connects to Services Gatekeeper.

General Exceptions

This section describes the exception handling for the Parlay X 2.1 interfaces.

These exception types are defined:

- Service exceptions
- Policy exceptions

Service exceptions are related to the operation of the service itself. The following exceptions are general:

- SVC0001: Service error.
- SVC0002: Invalid input value
- SVC0003: Invalid input value with list of valid values

- SVC0004: No valid addresses
- SVC0005: Duplicate correlator
- SVC0006: Invalid group
- SVC0007: Invalid charging information
- SVC0008: Overlapping Criteria

Policy Exceptions are thrown when a policy has been violated, including violations of a service level agreements. The following general Policy Exceptions are defined:

- POL0001: Policy error.
- POL0002: Privacy error.
- POL0003: Too many addresses specified.
- POL0004: Unlimited notifications not supported.
- POL0005: Too many notifications requested.
- POL0006: Groups not allowed.
- POL0007: Nested groups not allowed.
- POL0008: Charging not supported.
- POL0009: Invalid frequency requested.

Within the exception, an error code is defined. The error code details why the exception was thrown. See "General Error Codes".

General Error Codes

The following are general error codes for SVC0001: Service error:

- Null sessionID (loginTicket) expired.
- CN-000001 Two requests for call direction overlap with each other.
- CN-000002 Internal error when accessing the subscription storage.
- CN-000003 Could not find the call-back plug-in.
- CN-000004 The call direction routing address is not valid.
- MAP-000001 The Terminal Status plug-in failed to encode the `anyTimeInterrorgation` MAP message.
- MMS-000001 Unable to perform action. Network error. Check that the incoming Parlay X address request is correct and that the `NetworkSelection` for the supplied address has the correct `MAPApplicationContext` and `MAPDialogueAS` values
- MMS-000002 Unable to retrieve configuration, internal error.
- MMS-000003 The used address type is not supported.
- MMS-000004 An error occurred when an attachment was put into the request context.
- MMS-000005 The MM7 Relay server responded with an error code, which includes the status code and the status text in the following format: MMS-000005:*StatusCode*:*StatusText*.

  For example, if the MMSC response were:

  ```
  <rel:Status>
  ```

```
    <rel:StatusCode>4000</rel:StatusCode>
    <rel:StatusText>Hello</rel:StatusText>
</rel:Status>
```

the SVC0001 ServiceException error code would be:

```
"MMS-000005:4000:Hello".
```

- OSA-000001 Invalid network state.
- OSA-000002 Invalid interface type.
- OSA-000003 Invalid event type.
- OSA-000004 Unsupported address plan.
- OSA-000005 Communication failure between OSA Gateway and Services Gatekeeper.
- PLC-000001 Internal policy engine error.
- PLG-000001 Could not find remote ejb home in access tier.
- PLG-000002 Could not create the ejb.
- PLG-000003 Could not access callback ejb.
- PLG-000004 Matching plug-in cannot be found because, for example, route has not been set up.
- SMS-000001 Unable to perform action. Network error.
- SMS-000002 Unable to retrieve configuration, internal error.
- SMS-000003 The used address type is not supported.
- SMS-000004 Unable to encode message segments.
- SMS-000005 GSM message format incorrect.
- SMS-000006 Could not send message. Message was not accepted by the network.

  The status code and the status text are in the following format: SMS-000006:*CommandStatus*:*Description*

  For example, if the SMSC commandstatus response is:

  ```
  0x0000000B(ESME_RINVDSTADR)
  ```

  the SVC0001 ServiceException error code is:

  ```
  "SMS-000006:11:Invalid Dest Addr".
  ```

  The description of the command status is taken from the SMPP Specification v3.4. If the command status in the response from the SMSC is not explicitly described in the SMPP Specification, the text of the error is **No description available for** *CommandStatus***.**

- SS7API-000001 The SS7 server failed to build and send the message to the SS7 stack. Make sure that the stack is running and that the `CpUserId` and SS7 host/port/instance are correct.
- SS7-000001 No answer or an incorrect answer received from the SS7 stack.
- TL-000001 Unable to end notification because of a network error.
- TL-000002 Unable to end notification because of an internal error.
- TL- 000003 Unable to start geographical notification because of a network error.

- TL-000004 Unable to start geographical notification because of an internal error.
- TL-000005 Unable to start periodic notification because of a network error.
- TL-000006 Unable to start periodic notification because of an internal error.
- TL-000007 Unable to get location because of a network error.
- TL-000008 Unable to get location because of an internal error.
- TL-000009 Unable to get location because no data was found.
- TL-000010 Failed to parse response because of an internal error.
- TL-000011 Failed to get location information because the MLP server returned an error.
- TPC-000001 Unable to retrieve configuration because of an internal error.
- TS-000001 Communication problems between Services Gatekeeper and the network node. Contact your Services Gatekeeper administrator. The `GroupRequestTimeout` attribute may be too low.
- TS-000002 Could not find a network service route to match the address. Validate the network selection routes. You may want to add a default route (expression=DEFAULT) to capture any addresses that do not matched any other expression.
- TS-000003 No result returned from a `getStatusForGroup` query. Update `getGroupRequestTimeout` to an appropriate value and check the status of network connection
- TS-000004 Did not find the correlator when sending an end-status notification. The value of correlator in the `endNotification` request is incorrect, or the notification has already been terminated by the network.
- TS-000005 Could not initialize the object. Check the `instancemap` configuration file.
- TS-000006 Failed to access storage. Check the status of the storage service and database.
- WNG-000000 No error.
- WNG-000001 Unknown error.
- WNG-000002 Storage service error.

Code Examples

The following code examples illustrate how to use the Parlay X 2.1 interfaces.

Example: sendSMS

This is an SMS sending example.

Example 23–1 SendSMS example

```
org.csapi.schema.parlayx.sms.send.v2_2.local.SendSms request =
new org.csapi.schema.parlayx.sms.send.v2_2.local.SendSms();
SimpleReference sr = new SimpleReference();
sr.setEndpoint(new
URI("http://localhost:8111/SmsNotificationService/services/SmsNotification?WSDL"));
sr.setCorrelator("cor188");
```

```
sr.setInterfaceName("InterfaceName");
ChargingInformation charging = new ChargingInformation();
charging.setAmount(new BigDecimal("1.1"));
charging.setCode("qwerty");
charging.setCurrency("USD");
charging.setDescription("some charging info");
sendInf.setCharging(charging);
URI[] uri = new URI[1];
uri[0] = new URI("1234");
request.setAddresses(uri);
request.setCharging(charging);
request.setReceiptRequest(sr);
request.setMessage("we are testing sms!");
request.setSenderName("6001");
org.csapi.schema.parlayx.sms.send.v2_2.local.SendSmsResponse response =
smport.sendSms(request);
String sendresult = response.getResult();
System.out.println("result: " + sendresult);
```

Example: startSmsNotification

This is an example of how to use startSmsNotification.

Example 23–2 startSmsNotification example

```
org.csapi.schema.parlayx.sms.notification_manager.v2_3.local.StartSmsNotification parameters =
new org.csapi.schema.parlayx.sms.notification_manager.v2_3.local.StartSmsNotification();
parameters.setCriteria("hello");
SimpleReference sr = new SimpleReference();
sr.setEndpoint(new
URI("http://localhost:8111/SmsNotificationService/services/SmsNotification?WSDL"));
sr.setCorrelator("cor189");
sr.setInterfaceName("interfaceName");
parameters.setReference(sr);
URI uri = new URI("tel:6001");
parameters.setSmsServiceActivationNumber(uri);
port.startSmsNotification(parameters);
```

Example: getReceivedSms

This example shows how to poll for SMS messages using getReceivedSms.

Example 23–3 getReceivedSms example

```
org.csapi.schema.parlayx.sms.receive.v2_2.local.GetReceivedSms parameters =
new org.csapi.schema.parlayx.sms.receive.v2_2.local.GetReceivedSms();
parameters.setRegistrationIdentifier("1");
org.csapi.schema.parlayx.sms.receive.v2_2.local.GetReceivedSmsResponse response =
port.getReceivedSms(parameters);
org.csapi.schema.parlayx.sms.v2_2.SmsMessage[] msgs =
response.getResult();
if(msgs != null) {
  for(org.csapi.schema.parlayx.sms.v2_2.SmsMessage msg : msgs) {
    System.out.println(msg.getMessage());
  }
}
```

Example: sendMessage

This is an MMS sending example.

Example 23–4 sendMessage example

```
org.csapi.schema.parlayx.multimedia_messaging.send.v2_4.local.SendMessage request =
new org.csapi.schema.parlayx.multimedia_messaging.send.v2_4.local.SendMessage();
ChargingInformation charging = new ChargingInformation();
charging.setAmount(new BigDecimal("1.1"));
charging.setCode("qwerty");
charging.setCurrency("USD");
charging.setDescription("some charging info");
sendInf.setCharging(charging);
SimpleReference sr = new SimpleReference();
if(getProperty("notification_mt").equalsIgnoreCase("true")) {
  sr.setEndpoint(new URI(getProperty(ClientConstants.NOTIFICATION_LISTENER_URL)));
  sr.setCorrelator(getProperty("correlator"));
  sr.setInterfaceName(getProperty("interfacename"));
}
URI[] uri = new URI[1];
uri[0] = new URI("1234");
request.setAddresses(uri);
request.setCharging(charging);
request.setPriority(MessagePriority.fromString("Default"));
request.setReceiptRequest(sr);
request.setSenderAddress("6001");
request.setSubject("subject");
org.csapi.schema.parlayx.multimedia_messaging.send.v2_4.local.SendMessageResponse response =
smport.sendMessage(request);
String sendresult = response.getResult();
System.out.println("sendresult: " + sendresult);
```

Example: getReceivedMessages and getMessage

This is shows polling for a received MMS.

Example 23–5 getReceivedMessages and getMessage example

```
org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetReceivedMessages parameters =
new org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetReceivedMessages();
parameters.setPriority(org.csapi.schema.parlayx.multimedia_messaging.v2_
4.MessagePriority.fromString("Default"));
parameters.setRegistrationIdentifier("2");
org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetReceivedMessagesResponse result
=
port.getReceivedMessages(parameters);
org.csapi.schema.parlayx.multimedia_messaging.v2_4.MessageReference[] refs =
result.getResult();
if(refs != null) {
  for(org.csapi.schema.parlayx.multimedia_messaging.v2_4.MessageReference ref : refs) {
    String id = ref.getMessageIdentifier();
    org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetMessage p2 =
    new org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetMessage();
    p2.setMessageRefIdentifier(id);
    port.getMessage(p2);
  }
}
```

Example: getLocation

This example shows how to get the location of a terminal.

Example 23–6 getLocation example

```
org.csapi.schema.parlayx.terminal_location.v2_2.local.GetLocation parameters =
new org.csapi.schema.parlayx.terminal_location.v2_2.local.GetLocation();
parameters.setAcceptableAccuracy(5);
parameters.setAddress(new URI("1234"));
parameters.setRequestedAccuracy(5);
TimeMetric maximumAge = new TimeMetric();
maximumAge.setMetric(TimeMetrics.fromString("Hour"));
maximumAge.setUnits(10);
parameters.setMaximumAge(maximumAge);
TimeMetric responseTime = new TimeMetric();
responseTime.setMetric(TimeMetrics.fromString("Hour"));
responseTime.setUnits(1);
parameters.setResponseTime(responseTime);
DelayTolerance tolerance = DelayTolerance.fromString("NoDelay");
parameters.setTolerance(tolerance);
org.csapi.schema.parlayx.terminal_location.v2_2.local.GetLocationResponse response =
port.getLocation(parameters);
org.csapi.schema.parlayx.terminal_location.v2_2.nfo result =
response.getResult();
System.out.println("accuracy : " + result.getAccuracy());
System.out.println("altitude : " + result.getAltitude().floatValue());
System.out.println("latitude : " + result.getLatitude());
System.out.println("longitude : " + result.getLongitude());
System.out.println("timestamp : " + result.getTimestamp());
```

24

About the Supported SOAP Parlay X 3.0 Facades

This chapter describes the Oracle Communications Services Gatekeeper interfaces in the supported Parlay X 3.0 facades and contains information specific to Services Gatekeeper not found in the specifications. For detailed descriptions of the interfaces, methods, and parameters, see the ETSI OSA Parlay X 3.0 specifications at:

http://www.etsi.org/deliver/etsi_es/202300_202399/20239111/01.02.01_60/es_20239111v010201p.pdf

Parlay X 3.0 Part 6: Payment

The Payment communication service interfaces follow Draft ETSI ES 202 504-6 v0.0.4 (2007-06), Open Service Access (OSA); Parlay X Web services; Part 6: Payment (Parlay X 3)

Interface: AmountCharging

The **AmountCharging** interface endpoint is:
`http://host:port/parlayx30/payment/AmountCharging`

where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

chargeAmount

Charges the account indicated by the end user identifier.

Table 24–1 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000002	A protocol-related error. This error is returned from the Diameter server. Make sure the server is running and reachable. Check the log files for more information.
SVC0001	PAYMENT000003	A transient error. This error is returned from the Diameter server, for example, in an authentication failure. Make sure the server is reachable and has adequate storage space. Check the log files for more information.
SVC0001	PLG-000004	General plug-in routing error.

Table 24–1 (Cont.) Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000004	A permanent failure. This error is returned from the Diameter server, for example, when there is incorrect data in the AVP. Check the log files for more information.

SVC0270	PAYMENT000004	A permanent failure. This error is returned from the Diameter server containing a Diameter error code and string. The Diameter charging server returns the error code and string to Services Gatekeeper, and the Services Gatekeeper charging plug-in returns it to the calling application in the `Diameter-Error-Message` Xparam using this syntax:
		`DIAMETER`*`Diameter_error_code`*
		`Diameter-Error-Message="`*`Diameter_error_string`*`"`
		Where *`Diameter_error_code`* and *`Diameter_error_string`* are the codes and strings listed in Section 7.1 of Diameter RFC3588. For example:
		`DIAMETER5001`
		`Diameter-Error-Message="DIAMETER_AVP_UNSUPPORTED"`
		For details on the error codes and strings see Diameter RFC3588 Section 7.1 (http://www.ietf.org/rfc/rfc3588.txt)

refundAmount

Refunds the account indicated by the end user identifier.

Table 24–2 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000002	A protocol-related error. This error is returned from the Diameter server. Make sure the server is running and reachable. Check the log files for more information.
SVC0001	PAYMENT000003	A transient error. This error is returned from the Diameter server, such as authentication failure. Make sure the server is reachable and has adequate storage space. Check the log files for more information.
SVC0001	PLG-000004	General plug-in routing error.
SVC0001	PAYMENT000004	A permanent failure. This error is returned from the Diameter server, such as incorrect data in the AVP. Check the log files for more information.

Table 24–2 (Cont.) Exceptions and Error Codes

Exception	Error Code	Explanation

SVC0270	PAYMENT000004	A permanent failure. This error is returned from the Diameter server containing a Diameter error code and string. The Diameter charging server returns the error code and string to Services Gatekeeper, and the Services Gatekeeper charging plug-in returns it to the calling application in the `Diameter-Error-Message` Xparam using this syntax:
		`DIAMETER`*`Diameter_error_code`*
		`Diameter-Error-Message="`*`Diameter_error_string`*`"`
		Where *`Diameter_error_code`* and *`Diameter_error_string`* are the codes and strings listed in Section 7.1 of Diameter RFC3588. For example:
		`DIAMETER5001`
		`Diameter-Error-Message="DIAMETER_AVP_UNSUPPORTED"`
		For details on the error codes and strings see Diameter RFC3588 Section 7.1 (http://www.ietf.org/rfc/rfc3588.txt)

chargeSplitAmount

Charges multiple end user accounts concurrently.

Table 24–3 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000002	A protocol-related error. This error is returned from the Diameter server. Make sure the server is running and reachable. Check the log files for more information.
SVC0001	PAYMENT000003	A transient error. This error is returned from the Diameter server, for example, in an authentication failure. Make sure the server is reachable and has adequate storage space. Check the log files for more information.
SVC0001	PLG-000004	General plug-in routing error.
SVC0001	PAYMENT000004	A permanent failure. This error is returned from the Diameter server, for example, such as incorrect data in the AVP. Check the log files for more information.

Table 24–3 (Cont.) Exceptions and Error Codes

Exception	Error Code	Explanation

SVC0270	PAYMENT000004	A permanent failure. This error is returned from the Diameter server containing a Diameter error code and string. The Diameter charging server returns the error code and string to Services Gatekeeper, and the Services Gatekeeper charging plug-in returns it to the calling application in the `Diameter-Error-Message` Xparam using this syntax:
		`DIAMETER`*`Diameter_error_code`*
		`Diameter-Error-Message="`*`Diameter_error_string`*`"`
		Where *`Diameter_error_code`* and *`Diameter_error_string`* are the codes and strings listed in Section 7.1 of Diameter RFC3588. For example:
		`DIAMETER5001`
		`Diameter-Error-Message="DIAMETER_AVP_UNSUPPORTED"`
		For details on the error codes and strings see Diameter RFC3588 Section 7.1 (http://www.ietf.org/rfc/rfc3588.txt)

Interface: VolumeCharging

The **VolumeCharging** interface endpoint is:
`http://`*`host`*`:`*`port`*`/parlayx30/payment/VolumeCharging`

where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

chargeVolume

Charges the specified volume to the end user account.

Table 24–4 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0002	*parameter*	Invalid parameter value. Check the log files for more information.
SVC0270	PAYMENT000004	Charge failed. Check the log files for more information.

refundVolume

Refunds the specified volume to the end user account.

Table 24–5 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0002	*parameter*	Invalid parameter value. Check the log files for more information.
SVC0270	PAYMENT000004	Refund failed Check the log files for more information.

chargeSplitVolume

Charges a volume amount to multiple end user accounts based on percentages defined in a policy.

Table 24–6 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0002	*parameter*	Invalid parameter value. Check the log files for more information.
SVC0270	PAYMENT000004	The charge failed. Check the log files for more information.
SVC0271		Invalid sum of percentage allocations. The sum of percentage allocations must be 100.
POL0250		The number of endUserIdentifiers exceeds policy-defined maximum.
POL0251		Split charging is not supported by the defined policy.

getAmount

Converts the given volume to a currency amount.

chargeReservation

Charge the reserved volume to an end user account.

Table 24–7 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0002	*parameter*	Invalid parameter value. Check the log files for more information.
SVC0270	PAYMENT000004	Charge reservation failed Check the log files for more information.

releaseReservation

Release the reserved volume on an end user account.

Table 24–8 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0002	*parameter*	Invalid parameter value. Check the log files for more information.

Interface: ReserveAmountCharging

The **ReserveAmountCharging** interface endpoint is:
`http://host:port/parlayx30/payment/ReserveAmountCharging`

where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

reserveAmount

Reserves a charge for an account indicated by the end user identifier.

Table 24–9 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000002	A protocol-related error. This error is returned from the Diameter server. Make sure the server is running and reachable. Check the log files for more information.
SVC0001	PAYMENT000003	A transient error. This error is returned from the Diameter server, such as authentication failure. Make sure the server is reachable and has adequate storage space. Check the log files for more information.
SVC0001	PLG-000004	General plug-in routing error.
SVC0001	PAYMENT000004	A permanent failure. This error is returned from the Diameter server, such as incorrect data in the AVP. Check the log files for more information.

SVC0270	PAYMENT000004	A permanent failure. This error is returned from the Diameter server containing a Diameter error code and string. The Diameter charging server returns the error code and string to Services Gatekeeper, and the Services Gatekeeper charging plug-in returns it to the calling application in the `Diameter-Error-Message` Xparam using this syntax:
		`DIAMETER`*`Diameter_error_code`*
		`Diameter-Error-Message="`*`Diameter_error_string`*`"`
		Where *`Diameter_error_code`* and *`Diameter_error_string`* are the codes and strings listed in Section 7.1 of Diameter RFC3588. For example:
		`DIAMETER5001`
		`Diameter-Error-Message="DIAMETER_AVP_UNSUPPORTED"`
		For details on the error codes and strings see Diameter RFC3588 Section 7.1 (http://www.ietf.org/rfc/rfc3588.txt)

reserveAdditionalAmount

Adds to or subtracts from a charge to or from an existing reservation.

Table 24–10 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000002	A protocol-related error. This error is returned from the Diameter server. Make sure the server is running and reachable. Check the log files for more information.
SVC0001	PAYMENT000003	A transient error. This error is returned from the Diameter server, such as authentication failure. Make sure the server is reachable and has adequate storage space. Check the log files for more information.
SVC0001	PLG-000004	General plug-in routing error.

Table 24–10 (Cont.) Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000004	A permanent failure. This error is returned from the Diameter server, such as incorrect data in the AVP. Check the log files for more information.

SVC0270	PAYMENT000004	A permanent failure. This error is returned from the Diameter server containing a Diameter error code and string. The Diameter charging server returns the error code and string to Services Gatekeeper, and the Services Gatekeeper charging plug-in returns it to the calling application in the `Diameter-Error-Message` Xparam using this syntax:
		`DIAMETER`*`Diameter_error_code`*
		`Diameter-Error-Message="`*`Diameter_error_string`*`"`
		Where *`Diameter_error_code`* and *`Diameter_error_string`* are the codes and strings listed in Section 7.1 of Diameter RFC3588. For example:
		`DIAMETER5001`
		`Diameter-Error-Message="DIAMETER_AVP_UNSUPPORTED"`
		For details on the error codes and strings see Diameter RFC3588 Section 7.1 (http://www.ietf.org/rfc/rfc3588.txt)

chargeReservation

Charges to a reservation indicated by the reservation ID.

Table 24–11 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000002	A protocol-related error. This error is returned from the Diameter server. Make sure the server is running and reachable. Check the log files for more information.
SVC0001	PAYMENT000003	A transient error. This error is returned from the Diameter server, such as authentication failure. Make sure the server is reachable and has adequate storage space. Check the log files for more information.
SVC0001	PLG-000004	General plug-in routing error.
SVC0001	PAYMENT000004	A permanent failure. This error is returned from the Diameter server, such as incorrect data in the AVP. Check the log files for more information.

Table 24–11 (Cont.) Exceptions and Error Codes

Exception	Error Code	Explanation

SVC0270	PAYMENT000004	A permanent failure. This error is returned from the Diameter server containing a Diameter error code and string. The Diameter charging server returns the error code and string to Services Gatekeeper, and the Services Gatekeeper charging plug-in returns it to the calling application in the `Diameter-Error-Message` Xparam using this syntax:
		`DIAMETER`*`Diameter_error_code`*
		`Diameter-Error-Message="`*`Diameter_error_string`*`"`
		Where *`Diameter_error_code`* and *`Diameter_error_string`* are the codes and strings listed in Section 7.1 of Diameter RFC3588. For example:
		`DIAMETER5001`
		`Diameter-Error-Message="DIAMETER_AVP_UNSUPPORTED"`
		For details on the error codes and strings see Diameter RFC3588 Section 7.1 (`http://www.ietf.org/rfc/rfc3588.txt`)

releaseReservation

Returns funds left in a reservation to the account from which this reservation was made.

Table 24–12 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000002	A protocol-related error. This error is returned from the Diameter server. Make sure the server is running and reachable. Check the log files for more information.
SVC0001	PAYMENT000003	A transient error. This error is returned from the Diameter server, such as authentication failure. Make sure the server is reachable and has adequate storage space. Check the log files for more information.
SVC0001	PLG-000004	General plug-in routing error.
SVC0001	PAYMENT000004	A permanent failure. This error is returned from the Diameter server, such as incorrect data in the AVP. Check the log files for more information.

Table 24–12 (Cont.) Exceptions and Error Codes

Exception	Error Code	Explanation

SVC0270	PAYMENT000004	A permanent failure. This error is returned from the Diameter server containing a Diameter error code and string. The Diameter charging server returns the error code and string to Services Gatekeeper, and the Services Gatekeeper charging plug-in returns it to the calling application in the `Diameter-Error-Message` Xparam using this syntax:
		`DIAMETER`*`Diameter_error_code`*
		`Diameter-Error-Message="`*`Diameter_error_string`*`"`
		Where *`Diameter_error_code`* and *`Diameter_error_string`* are the codes and strings listed in Section 7.1 of Diameter RFC3588. For example:
		`DIAMETER5001`
		`Diameter-Error-Message="DIAMETER_AVP_UNSUPPORTED"`
		For details on the error codes and strings see Diameter RFC3588 Section 7.1 (http://www.ietf.org/rfc/rfc3588.txt)

Interface: ReserveVolumeCharging

The **ReserveVolumeCharging** interface endpoint is:
`http://`*`host`*`:`*`port`*`/parlayx30/payment/ReserveVolumeCharging`

where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

reserveVolume

Reserves a volume amount for the end user account.

Table 24–13 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0001	PAYMENT000002	Service error. Check the log files for more information.
SVC0002	*parameter*	Invalid parameter value. Check the log files for more information.
POL0001		Policy error. Check the log files for more information.

reserveAdditionalVolume

Adds or reduces a volume amount to for an existing reservation on an end user account.

Table 24–14 Exceptions and Error Codes

Exception	Error Code	Explanation
SVC0002	*parameter*	Invalid parameter value. Check the log files for more information.

getAmountReserveCharging

Converts a reserved volume to a currency amount.

Parlay X 3.0 Part 13: Address List Management

The Address List Management communication service interfaces follow 3GPP TS 29.199-13 V7.0.2 (2007-06), Open Service Access (OSA); Parlay X Web services; Part 13: Address List Management (Parlay X 3).

Interface: GroupManagement

This section describes the **GroupManagement** interface.

The GroupManagement interface endpoint is:

`http://host:port/parlayx30/address_list/GroupManagement`

where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

createGroup

Creates an Address List Management group.

Table 24–15 Exceptions and Error Codes

Exception	Error Code	Explanation
POL0212	n/a	Group name too long.
POL0213	n/a	Group already exists.

queryGroups

Queries an Address List Management group to return details about a particular group attribute, which is specified by attributeName.

deleteGroup

Deletes the specified Address List Management group.

setAccess

Sets access permissions for a member of an Address List Management group.

queryAccess

Queries the access permissions set for the group member passed in the requester parameter.

General Exceptions

See "General Exceptions".

Interface: Group

This section describes the **Group** interface.

The Group interface endpoint is:

`http://host:port/parlayx30/address_list/Group`

where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

addMember

Adds a single member to an Address List Management group.

addMembers

Adds multiple members to an Address List Management group.

Table 24–16 Exceptions and Error Codes

Exception	Error Code	Explanation
POL0210	n/a	Too many members in group.
POL0210	n/a	Subgroups not allowed.

queryMembers

Deletes the specified Address List Management group.

deleteMember

Deletes a single member from an Address List Management group.

deleteMembers

Deletes multiple members from an Address List Management group.

addGroupAttribute

Adds an attribute to an Address List Management group.

queryGroupAttribute

Queries an Address List Management group for the value associated with the passed attribute name. The attribute's value and status are returned.

deleteGroupAttribute

Deletes an attribute from an Address List Management group.

addGroupMemberAttribute

Adds an attribute to a member of an Address List Management group.

queryGroupMemberAttributes

Queries a member of an Address List Management group for list of attributes attached to the member.

deleteGroupMemberAttribute

Deletes an attribute from a member of an Address List Management group.

Interface: Member

This section describes the **Member** interface.

The Member interface endpoint is:

`http://host:port/parlayx30/address_list/Member`

where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

addMemberAttribute

Adds an attribute to a member outside of the context of a particular Address List Management group.

queryMemberAttributes

Queries a list of attributes for a member and retrieves their values.

deleteMemberAttribute

Deletes an attribute from a member.

Parlay X 3.0 Part 18: Device Capabilities and Configuration

The Device Capabilities and Configuration communication service interfaces follow ETSI ES 202 504-18 v0.0.1(2007-06), Open Service Access (OSA); Parlay X Web services; Part 18: Device Capabilities and Configuration (Parlay X 3).

Interface: DeviceCapabilities

This section describes the **DeviceCapabilities** interface.

The Device Capabilities interface endpoint is:

`http://host:port/parlayx30/rest/device_capabilities`

where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

getCapabilities

Retrieves the following capability information for a device:

- The device's unique ID.
- The name and model of the device.
- A link to the `UAProf` file.

getDeviceId

Retrieves the device's equipment identifier.

General Exceptions

See "General Exceptions".

Interface: DeviceCapabilitiesNotificationManager

This section describes the `DeviceCapabilitiesNotificationManager` interface.

startNotification

Not supported.

endNotification

Not supported.

Interface: DeviceCapabilitiesNotification

This section describes the **DeviceCapabilitiesNotification** interface.

deviceNotification

Not supported.

deviceError

Not supported.

deviceEnd

Not supported.

Interface: DeviceConfiguration

This section describes the `DeviceConfiguration` interface.

pushConfiguration

Not supported.

getConfigurationList

Not supported.

getConfigurationHistory

Not supported.

General Exceptions

This section describes the exception handling for the Parlay X 3.0 interfaces.

The following exception types are defined:

- Service exceptions
- Policy exceptions

Service exceptions are related to the operation of the service itself. The following exceptions are general:

- SVC0001: Service error.
- SVC0002: Invalid input value
- SVC0003: Invalid input value with list of valid values
- SVC0004: No valid addresses
- SVC0005: Duplicate correlator
- SVC0006: Invalid group
- SVC0007: Invalid charging information
- SVC0008: Overlapping Criteria

Policy exceptions are thrown when a policy has been violated, including violations of a service level agreements. The following general policy exceptions are defined:

- POL0001: Policy error
- POL0002: Privacy error
- POL0003: Too many addresses specified
- POL0004: Unlimited `notification`s are not supported
- POL0005: Too many `notification`s requested
- POL0006: Groups not allowed
- POL0007: Nested groups not allowed
- POL0008: Charging not supported
- POL0009: Invalid frequency requested

Within the exception, an error code is defined. The error code details why the exception was thrown.

25

About the Supported SOAP Parlay X 4.0 Facades

This chapter describes the Oracle Communications Services Gatekeeper interfaces in the supported Parlay X 4.0 facades and contains information specific to Services Gatekeeper not found in the specifications. For detailed descriptions of the interfaces, methods, and parameters, refer to the specifications, see the ETSI OSA Parlay X 4.0 specifications at:

http://www.3gpp.org/DynaReport/29199-01.htm

Parlay X 4.0 Part 17 Application-Driven QoS

The application driven QoS communication service interfaces follow.ETSI ES 202 504-17 V1.1.1 (2008-05) Open Service Access (OSA); Parlay X Web services; Part 17: Application-driven Quality of Service (QoS); Parlay X 3.

See the specification at the ETSI website:

http://www.etsi.org/deliver/etsi_es/202500_202599/20250417/01.01.01_60/es_20250417v010101p.pdf

See the discussion on Parlay X 4.0 application-driven quality of service (QoS) in *Services Gatekeeper Communication Service Reference Guide* for details on the supported operations.

There are two interfaces available for this specification:

- Interface: Application-driven QoS
- Interface: ApplicationQoSNotificationManager

Interface: Application-driven QoS

The Application-driven Quality of Service interface endpoint is:

`http://host:port/parlayx40/qos/ApplicationQoS`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

These operations are available to applications to manage QoS requests. These operations are also available from the Platform Test Environment (PTE) graphical user interface:

- applyQoSFeature
- getQoSHistory

- getQoSStatus
- modifyQoSFeature
- notifyQoSEvent
- removeQoSFeature

applyQoSFeature

Applications use this operation to apply the connection settings in a temporary QoS feature profile to a subscriber connection.

Table 25–1 lists the **applyQoSFeature** request parameters.

Table 25–1 applyQoSFeature Request Parameters

Name	Data Type/Values	Mandatory?	Description
endUserIdentifier	xsd:anyURI	Yes	Identifies the network subscriber for which to apply the QoS feature profile for. Uses the standard URI format: **tel:[0-9]**
qoSFeatureIdentifier	xsd:string	Yes	The name of the QoS feature profile to apply to the subscriber connection. The QoS feature profile must exist before use, so use the Operation: loadQoSRequestTemplate MBean operation to create it first.
defaultQoSFeature	xsd:boolean	Yes	Always set to **false**.
modifyExistingSession	xsd:boolean	No	Always set to **false**.
qosFeatureProperties	QoSFeatureProperties	No	Specifies values for the configurable service attributes that apply to the temporary QoS feature profile.
qosFeatureProperties.Duration	common:TimeMetri	No	The duration of the QoS feature profile being applied. Supported units of measurement are **Second**, **Minute**, **Hour**, **Day**, **Week**, **Month**, **Year**. A **0** or negative value extends the duration for the entire session.
qosFeatureProperties.UpStreamSpeedRate	xsd:string	No	The upload speed rate of the QoS feature profile. The format is **[0-9]bps | Kbps | Mbps | Gbps | Tbps**. The maximum value is **4294967295bps**.
qosFeatureProperties.DownStreamSpeedRate	xsd:string	No	The download speed rate of the QoS feature profile. The format is **[0-9]bps | Kbps | Mbps | Gbps | Tbps**. The maximum value is **4294967295bps**..
qosFeatureProperties.OtherProperties	Property[0..unbounded	No	The attribute-value pairs that replace the default values defined in QoS feature profile. For example, you can define an application identifier variable in the QoS feature profile: **<applicationIdentifier parameterName="$APP_ID">test_appId</applicationIdentifier>** And then specify a value for the variable by setting **applicationIdentifier** to **app_id: "$APP_ID=app_id"**

Request Example This example shows a request that attempts to apply a QoS feature profile named **pxt** to the **tel:555012** subscriber, for a duration of 7200 seconds (120 minutes, or 2 hours). The upload rate is 1000 Mbps and the download rate is 1000 Kbps:

```
<S:Body>      <ns2:applyQoSFeature
          xmlns:ns2="http://www.csapi.org/schema/parlayx/adq/v4_0/local"
          xmlns:ns3="http://www.csapi.org/schema/parlayx/common/v4_0">
          <ns2:endUserIdentifier>tel:555012</ns2:endUserIdentifier>
          <ns2:qoSFeatureIdentifier>pxt</ns2:qoSFeatureIdentifier>
          <ns2:defaultQoSFeature>false</ns2:defaultQoSFeature>
          <ns2:modifyExistingSession>false</ns2:modifyExistingSession>
          <ns2:qosFeatureProperties>
          <duration>
               <metric>Second</metric>
               <units>7200</units>
          </duration>
          <upStreamSpeedRate>1000 Mbps</upStreamSpeedRate>
          <downStreamSpeedRate>1000 kbps</downStreamSpeedRate>
          </ns2:qosFeatureProperties>
     </ns2:applyQoSFeature>
 </S:Body>
```

Response Parameters Table 25–3 lists the **applyQoSFeature** response parameters.

Table 25–2 applyQoSFeature Response Parameters

Name	Data Type	Mandatory?	Description
result	QoSFeatureData	Yes	Details of the actions taken as a result of the QoS feature profile request.
result.requestID	XSD:string	Yes	A unique request identifier generated by this communication service and used by the application to identify this specific invocation of the **applyQoSFeature** operation.
result.qoSFeatureIdentifier	XSD:string	Yes	A name that uniquely identifies the QoS feature profile being applied temporarily to the subscriber's connection.
actualProperties	QoSFeatureProperties	No	The AVPs returned by the PCRF.

Response Example This example shows a response to an **applyQoSFeature** request. It identifies the **pxt** feature profile that the QoS applies to, and Diameter Rx session details.

```
<env:Body>
     <loc:applyQoSFeatureResponse
     xmlns:loc="http://www.csapi.org/schema/parlayx/adq/v4_0/local">
     <loc:result>
          <requestId>localhost;1385973500;0-1386001353054</requestId>
          <qoSFeatureIdentifier>pxt</qoSFeatureIdentifier>
          <actualProperties>
               <duration>
                    <metric>Second</metric>
                    <units>6000</units>
               </duration>
                    <upStreamSpeedRate>1000000000bps</upStreamSpeedRate>
                    <downStreamSpeedRate>1000000bps</downStreamSpeedRate>
                    <otherProperties>
                         <name>supportedFeatures[0].vendorId</name>
                         <value>10415</value>
                    </otherProperties>
                    <otherProperties>
```

```
                              <name>supportedFeatures[0].featureListID</name>
                              <value>1</value>
                         </otherProperties>
                         <otherProperties>
                              <name>ratType</name>
                              <value>WLAN</value>
                         </otherProperties>
                         <otherProperties>
                              <name>ipcanType</name>
                              <value>_3GPP-GPRS</value>
                         </otherProperties>
                         <otherProperties>
                              <name>supportedFeatures[0].featureList</name>
                              <value>2</value>
                         </otherProperties>
                    </actualProperties>
               </loc:result>
          </loc:applyQoSFeatureResponse>
</env:Body>
```

getQoSHistory

Applications use this operation to return a list of the QoS transactions (feature profile requests and responses) for a subscriber. You can filter the list by specifying:

- A QoS feature profile identifier
- A maximum number of transactions
- A date and time limit
- Additional custom criteria that you create

After a request/response, there is approximately a 20-second delay before its record is available to **getQoSHistory**. To avoid filling up the database, request/response records are maintained for a period of 30 days and then deleted.

Request Parameters Table 25–3 lists the **getQoSHistory** request parameters.

Table 25–3 getQoSHistory Request Parameters

Name	Data Type	Mandatory?	Description
endUserIdentifier	XSD:anyURI	Yes	Identifies the subscriber for which to get a history. Uses the standard URI format.
qosFeatureIdentifier	XSD:string	No	The name of the QoS feature profile this operation returns a history for. If not specified, a history is returned for all feature profiles for this subscriber.
date	XSD:datTime	No	Limits the history returned to the date and time specified. If not specified, a history is returned for this subscriber for the previous minute only.
maxEntries	XSD:Integer	No	Specifies the maximum number of QoS feature profile requests and responses to return. If not specified (or a value of **0** is specified) the maximum is set to **10**. The largest value allowed is **100**.
additionalCriteria	Property[0..unbounded]	No	Not implemented in this release.

Request Example This example shows a request for a QoS feature profile history of requests and responses for the subscriber **tel:555012**, for the **pxt** feature, that transpired on December 12th, 2013 at **01:33:53**.

```
 <S:Body>
     <ns2:getQoSHistory
          xmlns:ns2="http://www.csapi.org/schema/parlayx/adq/v4_0/local"
          xmlns:ns3="http://www.csapi.org/schema/parlayx/common/v4_0">
          <ns2:endUserIdentifier>tel:555012</ns2:endUserIdentifier>
          <ns2:qoSFeatureIdentifier>pxt</ns2:qoSFeatureIdentifier>
          <ns2:date>2013-12-05T01:33:53Z</ns2:date>
          <ns2:maxEntries>10</ns2:maxEntries>
     </ns2:getQoSHistory>
</S:Body>
```

Response Parameters Table 25–4 lists the **getQoSHistory** response parameters.

Table 25–4 getQoSHistory Response Parameters

Name	Data Type	Mandatory?	Description
transactionDateTime	XSD:dateTime	Yes	Specifies the transaction time.
transactionDetails	XSD:string	Yes	Specifies the request/response details. The details name the portion of the name/value pairs separated by the **#** symbol. The name portion of the name/value pairs include: ■ startTime ■ endTime ■ requstId ■ duration ■ featureName ■ A list of request/response records, including the transaction time and a list of the modified variables.

Response Example This example shows an example of a history returned to the application:

```
<env:Body>
     <loc:getQoSHistoryResponse
          xmlns:loc="http://www.csapi.org/schema/parlayx/adq/v4_0/local">
          <loc:result>

<transactionDateTime>2013-12-05T09:33:53.522+08:00</transactionDateTime>
 <transactionDetails>requestId=localhost;1386206978;0-1386207233485#duration=7200
seconds#endTime=2013-12-05 09:53:42#featureName=pxt#modification:time=
1386207684611#duration=9600seconds#$FLOW_DESCRIPTION_0=permit out ip from any to
any#</transactionDetails>
     </loc:result>
     </loc:getQoSHistoryResponse>
</env:Body>
```

getQoSStatus

Applications use this operation to retrieve the status and details of a subscriber's current QoS feature profile.

Request Parameters Table 25–5 lists the **getQoSStatus** request parameters.

Table 25–5 getQoSStatus Request Parameters

Name	Data Type	Mandatory?	Description
endUserIdentifier	XSD:any URI	Yes	Identifies a subscriber to obtain QoS details for.

Request Example This example requests QoS details for subscriber **tel:555012**.

```
<S:Body>
     <ns2:getQoSStatus xmlns:ns2="http://www.csapi.org/schema/parlayx/adq/v4_
0/local"
          xmlns:ns3="http://www.csapi.org/schema/parlayx/common/v4_0">
          <ns2:endUserIdentifier>tel:555012</ns2:endUserIdentifier>
     </ns2:getQoSStatus>
</S:Body>
```

Response Parameters Table 25–6 lists the **getQoSStatus** response parameters.

Table 25–6 getQosStatus Response Parameters

Name	Data Type	Mandatory?	Description
result	QosStatus	Yes	Returns the status of a subscriber connection, including information about the temporary QoS feature profile details that are currently activated.
result.userIsConnected	XSD:Boolean	Yes	Specifies whether the subscriber account is currently connected (**true** or **false**).
result.defaultQoSFeatureIdentifier	XSD:string	No	Not used because setting a default QoS feature profile is not supported.
trafficClasses	TrafficClass[0..unbounded]	No	Not used because setting a default QoS feature profile is not supported.
qosFeatureStatuses	QoSFeatureData[0..unbounded]	No	An array of the QoS feature profile details for the subscriber.

Response Example This example shows the QoS status details returned for a subscriber:

```
<env:Body>
     <loc:getQoSStatusResponse
          xmlns:loc="http://www.csapi.org/schema/parlayx/adq/v4_0/local">
          <loc:result>
               <userIsConnected>true</userIsConnected>
               <qosFeatureStatuses>
                    <requestId>localhost;1386206978;0-1386207233485</requestId>
                    <qoSFeatureIdentifier>pxt</qoSFeatureIdentifier>
                    <actualProperties>
                         <duration>
                              <metric>Second</metric>
                              <units>9600</units>
                         </duration>
                         <upStreamSpeedRate>1000000000bps</upStreamSpeedRate>
```

```
                    <downStreamSpeedRate>1000000bps</downStreamSpeedRate>
                    <otherProperties>
                         <name>supportedFeatures[0].vendorId</name>
                         <value>10415</value>
                    <otherProperties>
                    <otherProperties>
                         <name>supportedFeatures[0].featureListID</name>
                         <value>1</value>
                    </otherProperties>
                    <otherProperties>
                         <name>ratType</name>
                         <value>WLAN</value>
                    </otherProperties>
                    <otherProperties>
                         <name>ipcanType</name>
                         <value>_3GPP-GPRS</value>
                    </otherProperties>
                    <otherProperties>
                         <name>supportedFeatures[0].featureList</name>
                         <value>2</value>
                    </otherProperties>
               </actualProperties>
          </qosFeatureStatuses>
     </loc:result>
</loc:getQoSStatusResponse>
</env:Body>
```

modifyQoSFeature

Applications use this operation to alter a QoS feature profile that has already been applied using "applyQoSFeature". Specify the attributes you want to change in the QoS feature profile. Any other attributes remain as originally set. Remember that the new values only take effect for the time limit set by the **qosFeatureProperties.Duration** attribute of "applyQoSFeature"

This operation must take place inside an active application session, and you must send in the request ID generated by **applyQosFeature**.

Request Parameters Table 25–7 lists the **modifyQoSFeature** request parameters:

Table 25–7 modifyQoSFeature Request Parameters

Name	Data Type	Mandatory?	Description
requestID	XSD:string	Yes	Specifies the unique request identifier generated by this communication service from the original **"applyQoSFeature"** operation.
requestProperties	QoSFeatureProperties	Yes	Specifies new values of the attributes of the QoS feature profile.

Request Example This example shows a request that modifies the QoS feature profile identified as **localhost;1386206978;0-1386207233485**. It changes the values of the **upStreamSpeedRate**, **downStreamSpeedRate**, and flow description attributes.

```
<S:Body>
     <ns2:modifyQoSFeature
          xmlns:ns2="http://www.csapi.org/schema/parlayx/adq/v4_0/local"
          xmlns:ns3="http://www.csapi.org/schema/parlayx/common/v4_0">
```

```
          <ns2:requestId>localhost;1386206978;0-1386207233485</ns2:requestId>
          <ns2:requestProperties>
               <duration>
                    <metric>Second</metric>
                    <units>9600</units>
               </duration>
               <upStreamSpeedRate>1000 Mbps</upStreamSpeedRate>
               <downStreamSpeedRate>1000 kbps</downStreamSpeedRate>
               <otherProperties>
                    <name>$FLOW_DESCRIPTION_0</name>
                    <value>permit out ip from any to any</value>
                    <description></description>
               </otherProperties>
          </ns2:requestProperties>
     </ns2:modifyQoSFeature>
</S:Body>
```

Response Parameters Table 25–8 lists the **modifyQoSFeature** response parameters.

Table 25–8 modifyQoSFeature Response Parameters

Name	Data Type	Mandatory?	Description
result	QoSFeatureProperties	Yes	Specifies the attributes that have been modified and their values.

Response Example This example shows the attributes changed by **modifyQoSFeature**. The modified attributes are returned in the **result** array.

```
<env:Body>
          <loc:modifyQoSFeatureResponse
               xmlns:loc="http://www.csapi.org/schema/parlayx/adq/v4_0/local">
               <loc:result>
                    <duration>
                         <metric>Second</metric>
                         <units>9600</units>
                    </duration>
                    <upStreamSpeedRate>1000 Mbps</upStreamSpeedRate>
                    <downStreamSpeedRate>1000 kbps</downStreamSpeedRate>
                    <otherProperties>
                         <name>$FLOW_DESCRIPTION_0</name>
                         <value>permit out ip from any to any</value>
                         <description></description>
                    </otherProperties>
               </loc:result>
          </loc:modifyQoSFeatureResponse>
</env:Body>
```

notifyQoSEvent

Your PCRF uses this operation to report certain network events that occurred against one or more subscriber's active QoS features profiles.

Request Parameters Table 25–9 lists the **notifyQoSEvent** request parameters.

Table 25–9 notifyQoSEvent Request Parameters

Name	Data Type	Mandatory?	Description
correlator	xsd:string	yes	The correlator identifying the original notification registration.
endUserIdentities	xsd:anyURI[1..unbounded]	Yes	The network subscribers associated with the event.
eventType	QoSEvent	Yes	The event being reported.

Response Parameters There are no response parameters.

removeQoSFeature

Applications use this operation to remove an active temporary QoS feature profile, and return the QoS values to the state they were in before the temporary QoS feature profile was applied.

This operation must take place inside an active session.

Request Parameters Table 25–10 lists the **removeQoSFeature** request parameters.

Table 25–10 removeQoSFeature Request Parameters

Name	Data Type	Mandatory?	Description
requestID	XSD:String	Yes	Contains the unique request ID generated by the original **applyQoSFeature** call.

Request Example This example shows a **removeQoSFeature** request that removes the temporary QoS feature profile identified as **localhost;1386206978;0-1386207233485**:

```
<S:Body>
     <ns2:removeQoSFeature
          xmlns:ns2="http://www.csapi.org/schema/parlayx/adq/v4_0/local"
          xmlns:ns3="http://www.csapi.org/schema/parlayx/common/v4_0">
          <ns2:requestId>localhost;1386206978;0-1386207233485</ns2:requestId>
     </ns2:removeQoSFeature>
</S:Body>
```

Response Parameters Table 25–11 lists the **removeQoSFeature** response parameters.

Table 25–11 removeQoSFeature Response Parameters

Name	Data Type	Mandatory?	Description
result	XSD:boolean	Yes	Returns true if the QoS feature profile was successfully removed.

Response Example This example shows a response to a request to remove a temporary QoS feature profile in which the removal was successful:

```
<env:Body>
     <loc:removeQoSFeatureResponse
          xmlns:loc="http://www.csapi.org/schema/parlayx/adq/v4_0/local">
          <loc:result>true</loc:result>
     </loc:removeQoSFeatureResponse>
</env:Body>
```

Interface: ApplicationQoSNotificationManager

The Application-driven Quality of Service interface endpoint is:

`http://host:port/parlayx40/qos/ApplicationQoSNotificationManager`

Where values for *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

These operations are available to applications to manage start and stop QoS notification. These operations are also available from the Platform Test Environment (PTE) graphical user interface:

- startQoSNotification
- stopQoSNotification

startQoSNotification

Applications use this operation to register interest in receiving notifications of specific event types for a subscriber.

Request Parameters Table 25–12 lists the **startQoSNotification** request parameters.

Table 25–12 startQoSNotification Reqeust Parameters

Name	Data Type	Mandatory?	Description
reference	common:SimpleReference	Yes	The application end point to receive event notifications.
reference.Endpoint	xsd:anyURI	Yes	The end point address.
reference.InterfaceName	xsd:string	Yes	The interface name.
reference.Correlator	xsd:string	Yes	Correlation information.
endUserIdentities	xsd:anyURI[1..unbounded]	Yes	The subscriber for which to monitor events.
eventCriteria	QoSEvent[1..unbounded]	Yes	The events to be monitored.

Table 25–13 lists the event definitions that you use for the **events** request parameter to startQoSNotifiation.

Table 25–13 startQoSNotification event Definitions

Event Name	Description	Trigger Condition
AbnormalConnectionTermination	The subscriber connections were terminated abnormally by a network fault that released all temporary QoS feature profile requests.	This communication service can not detect this kind of event, so you must use this event definition for all abnormal terminations.
NormalConnectionTermination	The subscriber connections were terminated normally (for example, the subscriber logged off), which automatically released all of the temporary QoS feature profile requests.	This event is triggered by a Diameter Rx abort session request (ASR) message.
TemporaryQoSFeatureReleased	An active temporary QoS feature profile request on a subscriber connection was released, because it reached a threshold specified by one of the service attributes.	This event is triggered by this communication service when a QoS feature profile duration expires.

Request Example This example shows an application requesting notifications for the subscriber **tel:555012** for both normal network connection terminations and temporary QoS feature profiles released.

```
<S:Body>
     <ns2:startQoSNotification
          xmlns:ns2="http://www.csapi.org/schema/parlayx/adq/notification_
manager/v4_0/local"
          xmlns:ns3="http://www.csapi.org/schema/parlayx/common/v4_0">
          <ns2:reference>
             <endpoint>http://endpoint_host.port/jaxws/QoSNotification</endpoint>
             <interfaceName>interfaceName</interfaceName>
             <correlator>987654321</correlator>
          </ns2:reference>
          <ns2:endUserIdentities>tel:555012</ns2:endUserIdentities>
          <ns2:eventCriteria>NormalConnectionTermination</ns2:eventCriteria>
          <ns2:eventCriteria>TemporaryQoSFeatureReleased</ns2:eventCriteria>
     </ns2:startQoSNotification>
</S:Body>
```

Response Parameters There are no response parameters because the connections have been terminated.

stopQoSNotification

Applications use this operation to cancel notification registrations started by **startQoSNotification**.

Request Parameters Table 25–14 lists the **stopQoSNotification** request parameters.

Table 25–14 stopQoSNotification Request Parameter

Name	Data Type	Mandatory?	Description
correlator	XSD:string	Yes	The correlator identifying the original notification registration.

Response Parameters There are no response parameters.

26

About the Supported SOAP Native Facade

This chapter describes the Oracle Communications Services Gatekeeper interfaces in the supported SOAP Native facade and contains information specific to Services Gatekeeper not found in the specifications.

About the Native Interfaces

This chapter provides details on these supported native interfaces:

- MM7
- SMPP
- UCP

MM7

The MM7 specification is available from the 3GPP website:

`http://www.3gpp.org/ftp/Specs/html-info/23140.htm`

Messages are compliant with the schema defined by Rel-5-MM7-1-2.xsd. Because the network-facing interface supports Rel-5-MM7-1-5.xsd, Rel-5-MM7-1-2.xsd and a modified version of REL-5-MM7-1-0.xsd, some mapping may be done during processing.

The endpoint for this interface is:

`http://host:port/mm7/Mms`

where values for host and port are the host name and port of the system on which Services Gatekeeper is installed.

> **Note:** The MM7 interface uses HTTP basic authentication, username/password. The username is the application instance ID.

Supported MM7 Operations

Services Gatekeeper supports the following MM7 operations:

- MM7_submit
- MM7_deliver
- MM7_cancel
- MM7_replace

- MM7_delivery_report
- MM7_read_reply_report

MM7_submit

Sends an application-initiated multimedia message

Table 26–1 Error Codes

Error code	Reason/Action
4006	Service unavailable. Communication error within Services Gatekeeper or between Services Gatekeeper and the MMSC Transient error. The client should try again.
4007	Service denied. The request was not allowed by policy. Contact the Services Gatekeeper administrator.
<all MMSC fault codes>	Passed along transparently Contact the Services Gatekeeper administrator.

MM7_deliver

Services Gatekeeper delivers a network-triggered message to the application, at an endpoint implemented by the application.

MM7_cancel

Not supported.

MM7_replace

Not supported

MM7_delivery_report

Services Gatekeeper delivers a delivery report on a previously sent message to the application, at an endpoint implemented by the application.

MM7_read_reply_report

Services Gatekeeper delivers a read reply report on a previously sent message to the application, at an endpoint implemented by the application.

SMPP

The native SMPP communication service exposes SMPP version 3.4 to applications.

The specification is the Short Message Peer to Peer, Protocol Specification v3.4, Document Version:- 12-Oct-1999 Issue 1.2. It can be downloaded from

`http://smsforum.net/`

The native SMPP communication service supports all Protocol Data Units (PDUs) for SMPP version 3.4, and all header and body elements except when stated otherwise.

The native SMPP communication service also supports the billing identification parameter in the format defined by SMPP Specification 5.1, section 4.8.4.3. This parameter works with SMPP 5.1 SMSCs. Services Gatekeeper supports it as a tunneled parameter named **smpp_billing_id**. It also supports the ussd_service_operation parameter, which was expanded to support the deliver_sm opration in SMPP 5.1.

Services Gatekeeper supports it as a tunneled parameter named **ussd_service_operation**. For details about these tunneled parameters, see the discussion on the Tunneled Parameters for Parlay X 2.1 Short Messaging / SMPP in *Services Gatekeeper Communication Service Reference Guide*.

An application using this interface acts as an External Short Message Entity (ESME).

Bind PDUs and Sessions

An application must bind to the native SMPP communications service. It can bind using:

- bind_transmitter PDU
- bind_receiver PDU
- bind_transceiver PDU

As a result of a bind operation, Services Gatekeeper authenticates the application and establishes a session.

The following is valid for all bind operations:

- An application binds using host name or IP address and port that depends on the installation. The server to bind to is a network tier server.
- The system_id field must be the application instance group ID assigned to the application instance.
- The password field must be the same as the password for the application instance group.

A session is maintained until the application sends an "unbind PDU".

Services Gatekeeper can be configured to allow a limited number of sessions per application through the maxSession parameter of the addApplicationSpecificSettings operation. See *Services Gatekeeper Communication Service Reference Guide* for information about this operation.

Services Gatekeeper can be configured to terminate a session if:

- The session is inactive. See the **InactivityTimerValue** in *Services Gatekeeper Communication Service Reference Guide*.
- The application takes too long time to respond to a request. See the **RequestTimerValue** in *Services Gatekeeper Communication Service Reference Guide*.

Error Handling

All errors are reported in the command_status field of a response PDU.

Table 26–2 lists the error codes that are specific for Services Gatekeeper. Errors from the SMSC are transparently forwarded to the application.

Table 26–2 Error Codes for SMPP Communication Service

SMPP PDU	Error Code in Response (command_status)	Description
bind_transmitter	ESME_RBINDFAIL	Could not bind.
bind_receiver	ESME_RBINDFAIL	Could not bind.
bind_transceiver	ESME_RBINDFAIL	Could not bind.

Table 26–2 (Cont.) Error Codes for SMPP Communication Service

SMPP PDU	Error Code in Response (command_status)	Description
submit_sm	ESME_RTHROTTLED	Throttling limit or quota limit exceeded. The application has performed too many requests per time unit and has exceeded the Service Level Agreement.
N/A	ESME_RSUBMITFAIL	Could not submit the message. Possible reasons include time-out encountered when sending the message and configuration error.
submit_sm_multi	ESME_RTHROTTLED	Same as for submit_sm.
N/A	ESME_RSUBMITFAIL	Same as for submit_sm.

Supported Operations

The following operations are supported or not supported as indicated.

bind_transmitter PDU

The application binds to Services Gatekeeper as an SMPP transmitter.

bind_transmitter_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "bind_transmitter PDU".

bind_receiver PDU

The application binds as an SMPP receiver to Services Gatekeeper.

The address_range field must be the same as provisioned for the application instance group in the addressRange parameter to the addApplicationSpecificSettings operation. See *Services Gatekeeper Communication Service Reference Guide* for information about this operation.

bind_receiver_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "bind_receiver PDU".

bind_transceiver PDU

The application binds as an SMPP transceiver to Services Gatekeeper.

The address_range field must be the same as provisioned for the application instance group in the addressRange parameter to the addApplicationSpecificSettings operation. See *Services Gatekeeper Communication Service Reference Guide* for information about this operation.

bind_transceiver_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "bind_transceiver PDU".

outbind PDU

Not supported.

unbind PDU

The application unbinds from Services Gatekeeper.

unbind_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "unbind PDU".

generic_nack PDU

Services Gatekeeper sends this PDU as a negative acknowledgement of a PDU sent from the application if the PDU can not be recognized.

If this PDU is sent from the application, it is propagated to the SMPP SMSC.

submit_sm PDU

The application sends a short message to Services Gatekeeper, which forwards it to the destination address using an SMSC.

submit_sm_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "submit_sm PDU".

submit_multi PDU

The application sends a short message to Services Gatekeeper, which forwards it to a set of destination addresses using an SMSC.

submit_multi_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "submit_multi PDU".

deliver_sm PDU

Services Gatekeeper sends this PDU to an application upon receiving from an SMSC a network-triggered short message that matches the destination addresses that the application is interested in. The PDU contains the short message.

The application expresses interest by subscribing for notifications addressed to specific destination addresses.

deliver_sm_resp PDU

The application sends this PDU to acknowledge the reception of a "deliver_sm PDU".

data_sm PDU

Not supported.

data_sm_resp PDU

Not supported.

query_sm PDU

The application sends this PDU to query the status of a previously-sent short message.

The communication service can be configured to allow or block this operation through the subsequentOperationsAllowed parameter to the addApplicationSpecificSettings operation.

query_sm_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "query_sm PDU".

cancel_sm PDU

The application sends this PDU to cancel the sending of one more previously-sent short messages, if the message has not yet been delivered to the end-user terminal.

The communication service can be configured to allow or block this operation through the subsequentOperationsAllowed parameter to the addApplicationSpecificSettings operation.

cancel_sm_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "cancel_sm PDU".

replace_sm PDU

The application sends this PDU to replace a previously-sent short message with the short message provided in this PDU, if the message has not yet been delivered to the end-user terminal.

The communication service can be configured to allow or block this operation through the subsequentOperationsAllowed parameter to the addApplicationSpecificSettings operation. See *Services Gatekeeper Communication Service Reference Guide* for information about this operation.

replace_sm_resp PDU

Services Gatekeeper sends this PDU to an application as a response to "replace_sm PDU".

enquire_link PDU

The application or Services Gatekeeper sends this PDU to verify the connection between the application and Services Gatekeeper.

The communication service can be configured to send this PDU to the application on a regular interval. When an application receives this PDU it must respond with "enquire_link_resp PDU" within the configured time-interval. See the **EnquireLinkTimerValue** attribute for the native SMPP plug-in in *Services Gatekeeper Communication Service Reference Guide*.

enquire_link_resp PDU

Services Gatekeeper or an application sends this PDU as a response to "enquire_link PDU".

alert_notification PDU

Not supported.

UCP

The universal computer protocol (UCP) communication service complies with the Short Message Service Centre EMI-UCP Interface 5.1 specification.

Error Handling

The following errors are reported to the application or the SMSC in the UCP NACK PDU under the conditions described.

ERROR_CODE_OPERATION_NOT_ALLOWED

- The UCP service has received something in suspended mode.
- The UCP service has received an openSession request on a connection that has already received an openSession request and has processed an OK response to it. Further openSession requests are not allowed.
- The UCP service has received an openSession request on a connection where it is currently processing an openSession request.
- All SMSCs have responded with NACK to an openSession request.
- The UCP service has received a session management operation on a client-side connection.

ERROR_CODE_AUTH_FAILURE

- Authentication between Services Gatekeeper and the application has failed.

ERROR_CODE_OPERATION_NOT_SUPPORTED

- The UCP service has received a session management operation that is not of the openSession subtype.
- The UCP service has received an operation that it does not understand or support on a server-side connection.
- The UCP service has received an operation that it does not understand or support on a client-side connection.

ERROR_CODE_SYNTAX_ERROR

- The UCP service received an exception when trying to deliver a PDU that was received on a server-side connection to a plug-in.
- The UCP service received an exception when trying to deliver a PDU that was received on a client-side connection to a plug-in.

Any errors triggered in the SMSC are propagated to the application. See the Short Message Service Centre EMI-UCP Interface 4.6 specification for a list of those error codes.

Native UCP Operations: Application-Facing Interface

This section describes the native UCP operations in the NativeUCPPluginNorth interface.

submitSM

Sends a mobile-terminated SMS.

Signature:

```
submitSM(UcpPDU submitSMPDU, ServerPort sourceServerPort, String
sourceConnectionId)
```

openSession

Opens a new UCP session.

Signature:

```
openSession(UcpPDU openSessionPDU, ServerPort sourceServerPort, String
sourceConnectionId)
```

ack

Sends an ACK to the SMSC.

Signature:

```
ack(UcpPDU ack, String sourceConnectionId)
```

nack

Sends a NACK to the SMSC.

Signature:

```
nack(UcpPDU nack, String sourceConnectionId)
```

deliverSM

Delivers a mobile-originated SMS.

Signature:

```
deliverSM(UcpPDU deliverSMPDU, String connectionId)
```

deliveryNotification

Delivers a message delivery notification associated with a previously sent mobile-terminated SMS.

Signature:

```
deliveryNotification(UcpPDU deliveryNotificationPDU, String connectionId)
```

Native UCP Operations: Network-Facing Interface

This section describes the supported native UCP operations in the NativeUCPPluginSouth interface.

ack

Sends an ACK to the application.

Signature:

```
ack(UcpPDU ack, String connectionId)
```

nack

Sends a NACK to the application.

Signature:

```
nack(UcpPDU ack, String connectionId)
```

Part V

Creating Applications Using Extended Web Service Interfaces

Part V explains how to use the native telephony facade to create applications that interact with Oracle Communications Services Gatekeeper.

Part V contains the following chapters:

27

Understanding the Extended Web Services Common Definitions

This chapter describes the Oracle Communications Services Gatekeeper definitions that the Extended Web Services share.

Namespace

The namespace for the common data types is:

- `http://www.bea.com/wlcp/wlng/schema/ews/common`

The namespace for the common faults is:

- `http://www.bea.com/wlcp/wlng/wsdl/ews/common/faults`

XML Schema Datatype Definition

This section explains the XML schema datatype definition.

AdditionalProperty Structure

Defines a name-value pair.

Table 27–1 AdditionalProperty Structure

Element Name	Element type	Optional	Description
name	xsd:string	Y	Name part.
value	xsd:string	Y	Value part.

ChargingInformation structure

For services that include charging as an inline message part, the charging information is provided in this data structure.

Table 27–2 ChargingInformation Structure

Element Name	Element type	Optional	Description
description	xsd:string	N	Description text to be use for information and billing text.
currency	xsd:string	Y	Currency identifier as defined in ISO 4217.
amount	xsd:decimal	Y	Amount to be charged.

Table 27–2 (Cont.) ChargingInformation Structure

Element Name	Element type	Optional	Description
code	xsd:string	Y	Charging code, referencing a contract under which the charge is applied.

SimpleReference structure

For those services that require a reference to a Web service, the information required to create the endpoint information is provided in this data structure.

Table 27–3 SimpleReference Structure

Element Name	Element type	Optional	Description
endpoint	xsd:anyURI	N	Description text to be use for information and billing text.
interfaceName	xsd:string	Y	Name of interface.
correlator	xsd:decimal	Y	Correlation information.

Fault Definitions

This section explains the fault definitions.

ServiceException

Faults related to the operation of the service, not including policy related faults, result in the return of a ServiceException message.

Table 27–4 ServiceException

Element Name	Element type	Optional	Description
messageId	xsd:string	N	Message identifier, with prefix SVC.
text	xsd:string	N	Message text, with replacement variables marked with %#
variables	xsd:string [0...unbounded]	Y	Variables to substitute into text string.

Service exceptions are related to the operation of the service itself. The following exceptions are general:

- SVC0001: Service error
- SVC0002: Invalid input value
- SVC0003: Invalid input value with list of valid values
- SVC0004: No valid addresses
- SVC0005: Duplicate correlator
- SVC0006: Invalid group
- SVC0007: Invalid charging information
- SVC0008: Overlapping criteria

PolicyException

Faults related to policies associated with the service, result in the return of a PolicyException message.

Table 27–5 PolicyException

Element Name	Element type	Optional	Description
messageId	xsd:string	N	Message identifier, with prefix POL.
text	xsd:string	N	Message text, with replacement variables marked with %#
variables	xsd:string [0...unbounded]	Y	Variables to substitute into text string.

PolicyExceptions are thrown when a policy has been violated, including violations of a service level agreements. The following general PolicyExceptions are defined:

- POL0001: Policy error
- POL0002: Privacy error
- POL0003: Too many addresses specified
- POL0004: Unlimited notifications not supported
- POL0005: Too many notifications requested
- POL0006: Groups not allowed
- POL0007: Nested groups not allowed
- POL0008: Charging not supported
- POL0009: Invalid frequency requested

28

Adding Extended Web Service Binary SMSs Support

This chapter explains how to use the Oracle Communications Services Gatekeeper binary SMS extended web service interface to add binary SMS support to applications.

Understanding the Binary SMS Web Service

The Extended Web Services Binary SMS Web Service allows for the sending and receiving of any generic binary content through SMSs. Both application-initiated and network-triggered requests are supported. The binary content can include data beyond the logos and ringtones specified by Parlay X Short Messaging. Examples of supported binary content include vCards, calendar entries, and WAP Push messages.

The Extended Web Services Binary SMS Web Service supports the automatic chunking of oversized binary SMS messages to handle messages that exceed the maximum size of a single SMS request. Oversized unsegmented messages are automatically divided into size conforming individual messages and handled by Services Gatekeeper if the proper encoding is provided in the message header.

Namespaces

The BinarySMS interface and service use the namespaces:

- `http://www.bea.com/wlcp/wlng/wsdl/ews/binary_sms/interface`
- `http://www.bea.com/wlcp/wlng/wsdl/ews/binary_sms/service`

The BinarySmsNotificationManager interface and service use the namespaces:

- `http://www.bea.com/wlcp/wlng/wsdl/ews/binary_sms/notification/interface`
- `http://www.bea.com/wlcp/wlng/wsdl/ews/binary_sms/notification/service`

In addition, Extended Web Services Binary SMS uses common data type definitions common for all Extended Web Services interfaces, see "Understanding the Extended Web Services Common Definitions".

Fault definitions are according to ETSI ES 202 391-1 V1.2.1 (2006-10) Open Service Access (OSA); Parlay X Web Services; Part 1: Common (Parlay X 2).

Endpoints

The endpoint for the BinarySMS interface is: `http://<host:port>/ews/binary_sms/BinarySms`

The endpoint for the BinarySmsNotificationManager interface is:
`http://`*host*`:`*port*`/ews/binary_sms/BinarySmsNotificationManager`

Where the values for *host* and *port* depend on your specific Services Gatekeeper deployment.

Sequence Diagram

This section explains the sequence diagrams for sending and receiving an SMS.

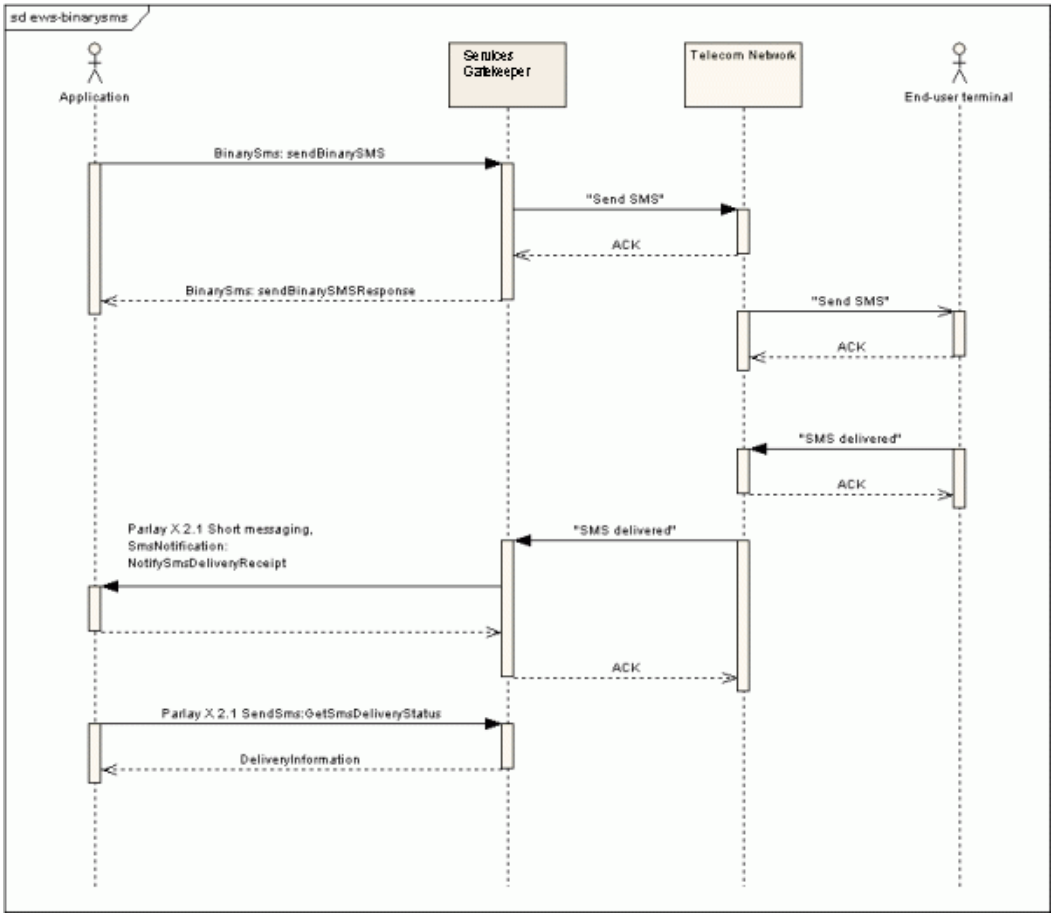
Send SMS

Figure 28–1 shows the general message sequence for sending a binary SMS message from an Extended Web Services Binary SMS application to the network. In this message sequence the application also receives a notification from the network indicating the delivery status of the SMS, that is that the message has reached its destination. It also displays how an application can query the delivery status of the message.

The interaction between the network and Services Gatekeeper is illustrated in a protocol-agnostic manner. The exact operations and sequences depend on which network protocol is being used.

Note: The delivery notifications are sent from the Parlay X 2.1 Short Messaging implementation.

Figure 28–1 Sequence diagram Application-initiated send Extended Web Services Binary SMS

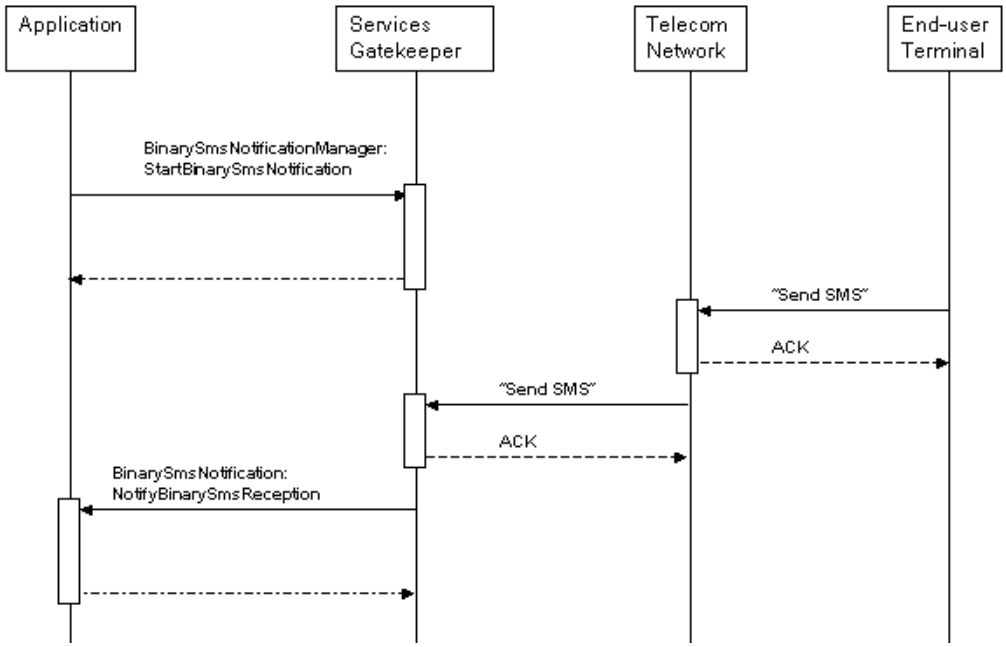


Receive SMS

Figure 28–2 shows the general message sequence for receiving a binary SMS message from the Network using Services Gatekeeper. In this message sequence the application also subscribes for a notifications on network triggered short messages.

The interaction between the network and Services Gatekeeper is illustrated in a protocol-agnostic manner. The exact operations and sequences depend on which network protocol is being used.

Figure 28–2 Sequence diagram receive Extended Web Services Binary SMS



XML Schema data type definition

The following data structures are used in the Extended Web Services Binary SMS Web Service.

BinaryMessage structure

Defines the binary payload of the SMS for application-initiated messages.

Defines the TP-User Data (TP-UD).

For a description of TP-User Data (TP-UD), TP-User-Data-Header-Indicator (TP UDHI), see 3GPP TS 23.040 V6.5.1, Technical realization of the Short Message Service (SMS) at:

http://www.3gpp.org/ftp/Specs/html-info/23040.htm

Table 28–1 BinaryMessage structure

Element Name	Element type	Optional	Description
udh	xsd:base64Binary	Y if message is set, otherwise N	Defines the User Data Header. See the description of TP-User Data (TP-UD) in the 3GPP specification for information about how to format the User Data Header.
message	xsd:base64Binary	Y if udh is set, otherwise N	Binary message data. Must be formatted according to TP-User Data (TP-UD), excluding the User Data Header.

BinarySmsMessage structure

Defines the binary payload of the SMS for network-triggered messages.

Table 28–2 BinarySmsMessage structure

Element Name	Element type	Optional	Description
message	ews_binary_sms_xsd:BinaryMessage[1..unbounded]	N	See "BinaryMessage structure".
dcs	xsd:byte	N	Data code schema, according to SMPP v3.4.
protocolId	xsd:byte	Y	TP-Protocol-Identifier according to 3GPP 23.040 6.5.1. Defines the TP-User Data (TP-UD). For a description of TP-User Data (TP-UD), TP-User-Data-Header-Indicator (TP UDHI), see 3GPP TS 23.040 V6.5.1, Technical realization of the Short Message Service (SMS) at: http://www.3gpp.org/ftp/Specs/html-info/23040.htm The protocol identifier is the information element by which the short message transport layer either refers to the higher layer protocol being used, or indicates interworking with a certain type of telematic device. Example: 123
senderAddress	xsd:anyURI	N	The address of the sender of the short message. Example: tel:1234556
smsServiceActivationNumber	xsd:anyURI	N	The destination address of the short message. Example: tel:1222
dateTime	xsd:dateTime	N	The timestamp of the message.

Interface: BinarySms

Operations to send SMSs with binary content.

Operation: sendBinarySMS

Sends an SMS with any binary data as content.

Input message: sendBinarySMS

Table 28–3 Input message: sendBinarySMS

Part name	Part type	Optional	Description
addresses	xsd:anyURI[1..unbounded]	N	An array of end-user terminal addresses. Example: tel:1234

Table 28–3 (Cont.) Input message: sendBinarySMS

Part name	Part type	Optional	Description
senderName	xsd:string	Y	The name of the sender. Alphanumeric. Example: tel:7485, Mycompany.
dcs	xsd:byte	N	Defines the data encoding scheme for the binaryMessage parameter. Formatted according to data_coding parameter in SMPP v3.4. See http://www.smsforum.net/
binaryMessage	binary_sms_xsd:BinaryMessage[1..unbounded]	N	Message payload. An array comprised of UDH elements and message elements, see "BinaryMessage structure". This array must be equal to or less than 140 bytes in size.
protocolId	xsd:byte	Y	TP-Protocol-Identifier (TP-PID) according to 3GPP TS 23.040 V6.5.1, Technical realization of the Short Message Service (SMS) at: http://www.3gpp.org/ftp/Specs/html-info/23040.htm Specifies the higher layer protocol being used, or indicates interworking with a certain type of telematic device.
validityPeriod	xsd:string	Y	Defines the validity period for the short message. Formatted according to validity_period parameter in SMPP v3.4. See http://www.smsforum.net/
charging	ews_common_xsd:ChargingInformation	Y	Charging information. See "ChargingInformation structure".
receiptRequest	ews_common_xsd:SimpleReference	Y	It defines the application endpoint, interfaceName and correlator that will be used to notify the application when the message has been delivered to the terminal or if delivery is impossible. See "SimpleReference structure"

Output message: sendBinarySMSResponse

Table 28–4 Output message: sendBinarySMSResponse

Part name	Part type	Optional	Description
result	xsd:string	N	Identifies a specific SMS delivery request.

Referenced faults

Table 28–5 exceptions and error codes

Exception	Error code	Reason/Action
SVC0001	BSMS-000001	Unable to perform action. Network error

Table 28–5 (Cont.) exceptions and error codes

Exception	Error code	Reason/Action
SVC0001	BSMS-000002	Unable to retrieve configuration, internal error.
SVC0001	BSMS-000003	The used address type is not supported
SVC0001	BSMS-000004	Unable to encode message segments. make sure the number of message segments is not 0.
SVC0001	BSMS-000005	GSM message format error.
SVC0001	BSMS-000006	Binary Message has too many segments.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	SenderName in non-alphanumeric format.
SVC0003	N/A	N/A
SVC0004	N/A	N/A
SVC0005	N/A	N/A
EPOL0001	N/A	N/A

Interface: BinarySmsNotificationManager

Operations to start and stop subscriptions for notifications for short messages with binary content.

Operation: StartBinarySmsNotification

Starts a subscription for notifications for short messages that have content in the form of binary data. A correlator is provided in the request. This correlator is used when stopping the subscription.

Input message: StartBinarySmsNotification

Table 28–6 Input message: StartBinarySmsNotification

Part name	Part type	Optional	Description
reference	ews_common_xsd:SimpleReference	N	Defines the application endpoint, interfaceName and correlator that will be used to forward a binary short message from the network. See "SimpleReference structure"
smsServiceActivationNumber	xsd:xsd:anyURI	Y	The destination address of the short message.

Output message: StartBinarySmsNotificationResponse

Table 28–7 Output message: StartBinarySmsNotificationResponse

Part name	Part type	Optional	Description
N/A	N/A	N/A	N/A

Referenced faults

Table 28–8 exceptions and error codes

Exception	Error code	Reason/Action
SVC0001	BSMS-000001	Unable to perform action. Network error
SVC0001	BSMS-000002	Unable to retrieve configuration, internal error.
SVC0001	BSMS-000003	The used address type is not supported
SVC0001	BSMS-000004	Unable to encode message segments. make sure the number of message segments is not 0.
SVC0001	BSMS-000005	GSM message format error.
SVC0001	BSMS-000006	Binary Message has too many segments.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	N/A
SVC0003	N/A	N/A
SVC0004	N/A	N/A
SVC0005	N/A	N/A
EPOL0001	N/A	N/A

Operation: StopBinarySmsNotification

Stops a previously started subscription for notifications for short messages that have content in the form of binary data. A correlator is provided in the request. This correlator was provided when the subscription was started, see "Operation: StartBinarySmsNotification".

Input message: StopBinarySmsNotification

Table 28–9 Input message: StopBinarySmsNotification

Part name	Part type	Optional	Description
correlator	xsd:String	N	The identifier for the subscription.

Output message: StopBinarySmsNotificationResponse

Table 28–10 Output message: StopBinarySmsNotificationResponse

Part name	Part type	Optional	Description
N/A	N/A	N/A	N/A

Referenced faults

Table 28–11 exceptions and error codes

Exception	Error code	Reason/Action
SVC0001	BSMS-000001	Unable to perform action. Network error
SVC0001	BSMS-000002	Unable to retrieve configuration, internal error.
SVC0001	BSMS-000003	The used address type is not supported

Table 28–11 (Cont.) exceptions and error codes

Exception	Error code	Reason/Action
SVC0001	BSMS-000004	Unable to encode message segments. make sure the number of message segments is not 0.
SVC0001	BSMS-000005	GSM message format error.
SVC0001	BSMS-000006	Binary Message has too many segments.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	N/A
SVC0003	N/A	N/A
SVC0004	N/A	N/A
SVC0005	N/A	N/A
EPOL0001	N/A	N/A

Interface: BinarySmsNotification

This interface is implemented by the application. It is used by Services Gatekeeper to deliver short messages with binary content to an application. Only messages that match a previously started subscription for notifications are delivered.

Note: Notifications on delivered short messages are delivered using the Parlay X 2.1 Short Messaging SmsNotification interface, using the method NotifySmsDeliveryReceipt.

Operation: NotifyBinarySmsReception

Services Gatekeeper calls this methods on

The notification is used to send a short message with binary content to the application. The notification occurs if the short message matched the criteria specified when starting the notification. See "Operation: StartBinarySmsNotification".

The method must be implemented by a Web Service at the application side. It is be invoked by Services Gatekeeper when it receives a short message with binary content form the network and the criteria is fulfilled.

Input message: NotifyBinarySmsReceptionRequest

Table 28–12 Input message: NotifyBinarySmsReceptionRequest

Part name	Part type	Description
correlator	xsd:String	The correlator for the subscription.
message	ews_binary_sms_xsd:BinarySmsMessage	The message in binary form. See "BinarySmsMessage structure".

Output message: NotifyBinarySmsReceptionResponse

Table 28–13 Output message: NotifyBinarySmsReceptionResponse

Part name	Part type	Optional	Description
N/A	N/A	N/A	N/A

Referenced faults

None.

Configuring Automatic Chunking of Binary SMSs

This interface includes a feature that automatically separates oversize SMS messages into smaller segments so that even switches that limit the size of SMSs will support them. This feature works for SMS messages that use either user data header (UDH) headers, or **Sar_** headers. Each chunk gets its own header segment, so your switches treat them a separate SMS messages.

You use the **wlng.smpp.concatenated_message_indicator** Services Gatekeeper system property to select the type of header you use. **0** is the default value. It indicates UDH headers. Change this setting to **1** if your implementation processes SMSs with **Sar_** headers.

This feature follows the 3GPP TS 23.038 and 3GPP TS 23.040 specifications for UDH and the SMPPv3.4 specification for Sar.

WSDLs

The document/literal WSDL representation of the interfaces can be retrieved from the Web Services endpoints, see "Endpoints".

The notification interface can be downloaded from:

```
http://host:port/ews/binary_sms/wsdls/ews_binary_sms_notification_service.wsdl
http://host:port/ews/binary_sms/wsdls/ews_binary_sms_notification_interface.wsdl
```

Where the values for *host* and *port* depend on the Services Gatekeeper deployment.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See "General Error Codes".
- Error codes defined for Parlay X 2.1 Short Messaging, see "Error Codes".
- 16133 Too many segments in message.

The following error codes are defined for EPOL0001: Policy error:

- See "Code Examples".
- Policy error codes defined for Parlay X 2.1 Short Messaging, see "Error Codes".

Sample Send Binary SMS

Example 28–1 Example Send Binary SMS

```
BinarySmsService service = new BinarySmsService_Impl("http://localhost:8001/ews/binary_
sms/BinarySms?WSDL");
```

```
BinarySms port = service.getBinarySms();
com.bea.wlcp.wlng.schema.ews.binary_sms.local.SendBinarySms parameters =
new com.bea.wlcp.wlng.schema.ews.binary_sms.local.SendBinarySms();
URI[] addresses = new URI[1];
addresses[0] = new URI("tel:1234");
parameters.setAddresses(addresses);
parameters.setDcs((byte)0);
parameters.setProtocolId((byte)0x7b);
parameters.setSenderName("tel:7878");
parameters.setValidityPeriod("020610233429000R");
com.bea.wlcp.wlng.schema.ews.binary_sms.BinaryMessage[] binaryMessages =
new com.bea.wlcp.wlng.schema.ews.binary_sms.BinaryMessage[1];
binaryMessages[0] = new com.bea.wlcp.wlng.schema.ews.binary_sms.BinaryMessage();
byte[] udh = {0};
byte[] message = {0x4d, 0x61, 0x64, 0x65, 0x20, 0x69, 0x6e, 0x20, 0x2e};
binaryMessages[0].setUdh(udh);
binaryMessages[0].setMessage(message);
parameters.setBinaryMessage(binaryMessages);
port.sendBinarySms(parameters);
```

29

Adding WAP Push Extended Web Service Message Support

This chapter explains how to use the Oracle Communications Services Gatekeeper WAP Push Extended Web Service interface to add WAP Push support to applications.

Understanding the WAP Push Extended Web Service Interface

The WAP Push Extended Web Services interface sends messages which are rendered as WAP Push messages by the addressee's terminal. The content of the message is coded as a PAP message. It also provides an asynchronous notification mechanism for delivery status.

The payload of a WAP Push message must adhere to the following specifications:

- WAP Service Indication Specification, as specified in Service Indication Version 31-July-2001, Wireless Application Protocol WAP-167-ServiceInd-20010731-a.
- WAP Service Loading Specification, as specified in Service Loading Version 31-Jul-2001, Wireless Application Protocol WAP-168-ServiceLoad-20010731-a.
- WAP Cache Operation Specification, as specified in Cache Operation Version 31-Jul-2001, Wireless Application Protocol WAP-175-CacheOp-20010731-a.

See the Open Mobile Alliance websitewebsite for links to the specifications:

`http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html`

The payload is sent as a SOAP attachment.

See "Sending Custom Message Content for Split and Submit Messaging Requests" for instructions on how to split messages into multiple individually-addressed requests

Namespaces

The PushMessage interface and service use the namespaces:

- `http://www.bea.com/wlcp/wlng/wsdl/ews/push_message/interface`
- `http://www.bea.com/wlcp/wlng/wsdl/ews/push_message/service`

The PushMessageNotification interface and service use the namespaces:

- `http://www.bea.com/wlcp/wlng/wsdl/ews/push_message/notification/interface`
- `http://www.bea.com/wlcp/wlng/wsdl/ews/push_message/notification/service`

The data types are defined in the namespace:

- `http://www.bea.com/wlcp/wlng/schema/ews/push_message`

In addition, WAP Push Extended Web Service uses definitions common for all Extended Web Services interfaces:

- The datatypes are defined in the namespace:
  - `http://www.bea.com/wlcp/wlng/schema/ews/common`
- The faults are defined in the namespace:
  - `targetNamespace="http://www.bea.com/wlcp/wlng/wsdl/ews/common/faults"`

Endpoint

The endpoint for the PushMessage interface is: `http://`*`host`*`:`*`port`*`/ews/push_message/PushMessage`

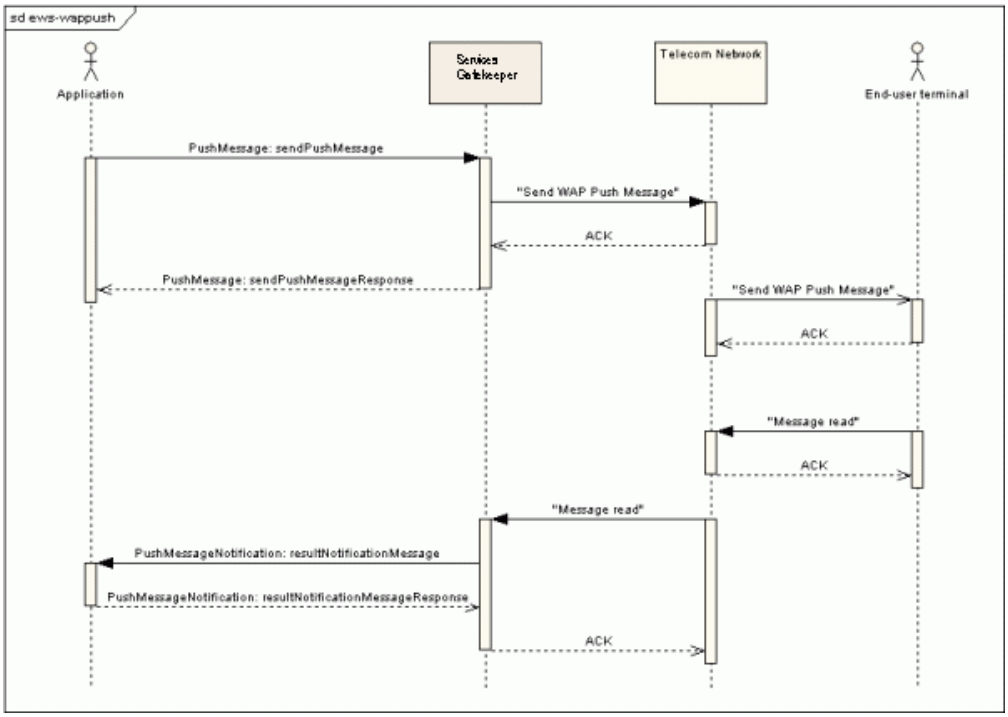
Where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Sequence Diagram

Figure 29–1 shows the general message sequence for sending a WAP Push message from an application that uses WAP Push Extended Web Service to the network. In this message sequence the application also receives a notification from the network indicating the delivery status of the WAP Push message, that is that the message has been read The interaction between the network and Services Gatekeeper is illustrated in a protocol-agnostic manner. The exact operations and sequences depend on which network protocol is being used.

Note: Zero or more `resultNotificationmesages` are sent to the application, depending on parameters provided in the initial `SendPushMessage` request.

Figure 29–1 Sequence Diagram of WAP Push Extended Web Services



XML Schema Data Type Definition

The following data structures are used in the WAP Push Extended Web Service.

PushResponse Structure

Defines the response that Services Gatekeeper returns from a sendPushMessage operation.

Table 29–1 PushResponse Structure

Element Name	Element type	Optional	Description
result	push_message_xsd:ResponseResult	No	The ResponseResult allows the server to specify a code for the outcome of sending the push message. See "ResponseResult structure"
pushId	xsd:string	No	The push ID provided in the request.
senderAddress	xsd:string	Yes	Contains the address to which the message was originally sent, for example the URL to the network node.
senderName	xsd:string	Yes	The descriptive name of the server.
replyTime	xsd:dateTime	Yes	The date and time associated with the creation of the response.
additionalProperties	ews_common_xsd:AdditionalProperty	Yes	Additional properties.The supported properties are: pap.stage, pap.note, pap.time

ResponseResult structure

Defines the result element in the PushResponse structure, which is used in the response returned from a sendPushMessage operation.

Table 29–2 ResponseResult Structure

Element Name	Element type	Optional	Description
code	xsd:string	No	A code representing the outcome when sending the push message. Generated by the network node. Possible status codes are listed in this section.
description	xsd:string	No	Textual description.

Table 29–3 Outcome Status Codes

Status code	Description
1000	OK.
1001	Accepted for processing.
2000	Bad request.
2001	Forbidden.
2002	Address error.
2003	Address not found.
2004	Push ID not found.
2005	Capabilities mismatch.
2006	Required capabilities not supported.
2007	Duplicate push ID.
2008	Cancellation not possible.
3000	Internal server error.
3001	Not implemented.
3002	Version not supported.
3003	Not possible.
3004	Capability matching not possible.
3005	Multiple addresses not supported.
3006	Transformation failure.
3007	Specified delivery method not possible.
3008	Capabilities not available.
3009	Required network not available.
3010	Required bearer not available.
3011	Replacement not supported.
4000	Service failure.
4001	Service unavailable.

ReplaceMethod enumeration

Defines the values for the replacePushId parameter in the sendPushMessage operation. This parameter is used to replace an existing message based on a given push ID. This parameter is ignored if it is set to NULL.

Table 29–4 ReplaceMethod Enumeration

Enumeration value	Description
all	Indicates that this push message must be treated as a new push submission for all recipients, whether or not a previously submitted push message with pushId equal to the replacePushId in this push message can be found.
pending-only	Indicates that this push message should be treated as a new push submission only for those recipients who have a pending push message that is possible to cancel. In this case, if no push message with pushId equal to the replacePushId in this push message can be found, the server responds with status code PUSH_ID_NOT_FOUND in the responseResult. Status code CANCELLATION_NOT_POSSIBLE may be returned in the responseResult if no message can be cancelled. Status code CANCELLATION_NOT_POSSIBLE may also be returned in a subsequent resultNotification to indicate a non-cancellable message for an individual recipient.

MessageState enumeration

Defines the values for the messageState parameter in a resultMessageNotification.

Table 29–5 MessageState Enumeration

Enumeration value	Description
rejected	Message was not accepted by the network.
pending	Message is being processed.
delivered	Message successfully delivered to the network.
undeliverable	The message could not be delivered.
expired	The message reached the maximum allowed age or could not be delivered by the time specified when the message was sent. Some network elements allows for defining policies on maximum age of messages.
aborted	The end-user terminal aborted the message.
timeout	The delivery process timed out.
cancelled	The message was cancelled.
unknown	The state of the message is unknown.

WAP Push Extended Web Service Interface Descriptions

The following describes the interfaces and operations that are available in the WAP Push Extended Web Service.

Interface: PushMessage

Operations to send, or to manipulate previously sent, WAP Push messages.

Operation: sendPushMessage

Sends a WAP Push message. The message Content Entity (the payload) is provided as a SOAP attachment in MIME format. The Content Entity is a MIME body part containing the content to be sent to the wireless device. The content type is not defined, and can be any type as long as it can be described by MIME. The Content Entity is included only in the push submission and is not included in any other operation request or response.

Input message: sendPushMessage

Table 29–6 Input Message: sendPushMessage

Part name	Part type	Optional	Description
pushId	xsd:string	N	Provided by the application. Serves as a message ID. The application is responsible for its uniqueness, for example, by using an address within its control (for example a URL) combined with an identifier for the push message as the value for pushId. Supported types are PLMN and USER. For example: "www.wapforum.org/123" or "123@wapforum.org"
destinationAddresses	xsd:string [1..unbounded]	N	An array of end-user terminal addresses. The addresses should be formatted according to the Push Proxy Gateway Service Specification (WAP-249-PPGService-20010713-a). Example addresses: ■ WAPPUSH=+155519990730 TYPE=PLMN@ppg.carrier.com ■ WAPPUSH=john.doe%40wapforum.org TYPE=USER@ppg.carrier.com
resultNotificationEndpoint	xsd:anyURI	Y	Specifies the URL the application uses to return result notifications. The presence of this parameter indicates that a notification is requested. If the application does not want a notification, this parameter must be set to NULL.

Table 29–6 (Cont.) Input Message: sendPushMessage

Part name	Part type	Optional	Description
replacePushId	xsd:string	Y	The pushId of the still pending message to replace. The presence of this parameter indicates that the client is requesting that this message replace one previously submitted, but still pending push message. The following rules apply: ■ Setting the replacePushId parameter to NULL indicates that it is a new message. It does not replace any previously submitted message. ■ The initial pending (pending delivery to the end-user terminal) message is cancelled, if possible, for all recipients of the message. This means that it is possible to replace a message for only a subset of the recipients of the original message. ■ Message replacement will occur only for the recipients for whom the pending message can be cancelled.
replaceMethod	push_ message_ xsd:Replace Method	N	Defines how to replace a previously sent message. Used in conjunction with the replacePushId parameter described above. Ignored if replacePushId is NULL.
deliverBeforeTimestamp	xsd:dateTime	Y	Defines the date and time by which the content must be delivered to the end-user terminal. The message is not delivered to the end-user terminal after this time and date. If the network node does not support this parameter, the message is rejected.
deliverAfterTimestamp	xsd:dateTime	Y	Specifies the date and time after which the content should be delivered to the wireless device. The message is delivered to the end-user terminal after this time and date. If the network node does not support this parameter, the message is be rejected.
sourceReference	xsd:string	Y	A textual name of the content provider.

Table 29–6 (Cont.) Input Message: sendPushMessage

Part name	Part type	Optional	Description
progressNotesRequested	xsd:boolean	Y	This parameter informs the network node if the client wants to receive progress notes. TRUE means that progress notes are requested. Progress notes are delivered using the PushMessageNotification interface. If not set, progress notes are not sent.
serviceCode	xsd:string	N	Used for charging purposes.
requesterID	xsd:string	N	The application ID as given by the operator.
additionalProperties	ews_common_xsd:AdditionalProperty [0...unbounded]	Y	Additional properties, defined as name/value pairs, can be sent using this parameter. The supported properties are: pap.priority, pap.delivery-method, pap.network, pap.network-required, pap.bearer, pap.bearer-required.

Output message: sendPushMessageResponse

Table 29–7 Output Message: sendPushMessageResponse

Part name	Part type	Optional	Description
result	push_message_xsd:PushResponse	N	The response that Services Gatekeeper returns for sendPushMessage operation

Referenced faults

Table 29–8 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000001	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact Services Gatekeeper administrator.
SVC0001	PUSHMSG-000002	Failed to create push message.
SVC0001	PUSHMSG-000003	Unable to retrieve configuration.
SVC0001	PUSHMSG-000001	Failed to submit push message to PPG.
SVC0001	PLG-000004	General plug-in routing error

Interface: PushMessageNotification

Operations resultNotificationMessage and resultNotificationMessageResponse.

Operation: resultNotificationMessage

Input message: resultNotificationMessage

Table 29–9 Input Message: resultNotificationMessage

Part name	Part type	Optional	Description
pushId	xsd:string	N	Defined by the application in the corresponding sendPushMessage operation. Used to match the notification to the message.
address	xsd:string	N	The address of the end-user terminal.
messageState	push_ message_ xsd:Message State	N	State of the message.
code	xsd:string	N	Final status of the message.
description	xsd:string	Y	Textual description of the notification. Supplied by the network. May or may not be present, depending on the network node used.
senderAddress	xsd:string	Y	Address of the network node. May or may not be present, depending on the network node used.
senderName	xsd:string	Y	Name of the network node. May or may not be present, depending on the network node used.
receivedTime	xsd:dateTime	Y	Time and date when the message was received at the network node.
eventTime	xsd:dateTime	Y	Time and date when the message reached the end-user terminal.
additionalProperties	ews_ common_ xsd:Addition alProperty	Y	Additional properties can be sent using this parameter in the form of name/value pairs. The supported properties are: ▪ pap.priority ▪ pap.delivery-method ▪ pap.network ▪ pap.network-required ▪ pap.bearer ▪ pap.bearer-required Which properties are sent, if any, is dependent on the network node.

Output message: resultNotificationMessageResponse

Table 29–10 Output Message: resultNotificationMessageResponse

Part name	Part type	Optional	Description
N/A	N/A	N/A	N/A

Referenced faults

Table 29–11 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	PUSHMSG-000004	Failed to send result notification to the application.

WSDLs

The document/literal WSDL representation of the PushMessage interface can be retrieved from the Web services endpoint.

The document/literal WSDL representation of the PushMessageNotification interface can be downloaded from

`http://`*host*`:`*port*`/ews/push_message/wsdls/ews_common_types.xsd`

`http://`*host*`:`*port*`/ews/push_message/wsdls/ews_push_message_notification_interface.wsdl`

`http://`*host*`:`*port*`/ews/push_message/wsdls/ews_push_message_notification_service.wsdl`

`http://`*host*`:`*port*`/ews/push_message/wsdls/ews_push_message_types.xsd`

Where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Sample Send WAP Push Message

Example 29–1 Example Send WAP Push Message

```
// Add handlers for MIME types needed for WAP MIME-types
MailcapCommandMap mc = (MailcapCommandMap) CommandMap.getDefaultCommandMap();
mc.addMailcap("text/vnd.wap.si;;x-java-content-handler=com.sun.mail.handlers.text_xml");
CommandMap.setDefaultCommandMap(mc);
// Create a MIME-message where with the actual content of the WAP Push message.
InternetHeaders headers = new InternetHeaders();
headers.addHeader("Content-type", "text/plain; charset=UTF-8");
headers.addHeader("Content-Id", "mytext");
byte[] bytes = "Test message".getBytes();
MimeBodyPart mimeMessage = new MimeBodyPart(headers, bytes);

// Create PushMessage with only the manadatory parameters

// SendPushMessage is provided in the stubs generated from the WSDL.
SendPushMessage sendPushMessage = new SendPushMessage();
String [] destinationAddresses = {"wappush=461/type=user@ppg.o.se"};
sendPushMessage.setDestinationAddresses(destinationAddresses);
// Create "unique" pushId, using a combination of timestamp and domain.
sendPushMessage.setPushId(System.currentTimeMillis() + "@wlng.bea.com");
// ReplaceMethod is provided by the stubs generated from the WSDL.
sendPushMessage.setReplaceMethod(ReplaceMethod.pendingOnly);
// Defined by the operator/service provider contractual agreement
sendPushMessage.setServiceCode("Service Code xxx");
// Defined by the operator/service provider contractual agreement
sendPushMessage.setRequesterID("Requester ID xxx");
// Endpoint to send notifications to. Implemented on the application side.
String notificationEndpoint = "http://localhost:80/services/PushMessageNotification";
sendPushMessage.setResultNotificationEndpoint(new URI(notificationEndpoint));
```

```
// Send the WAP Push message
PushMessageService pushMessageService = null;
// Define the endpoint of the WAP Push Web service
String endpoint = "http://localhost:8001/ews/push_message/PushMessage?WSDL";
try {
  // Instantiate an representation of the Web service from the generated stubs.
  pushMessageService = new PushMessageService_Impl(endpoint);
} catch (ServiceException e) {
  e.printStackTrace();
  throw e;
}
PushMessage pushMessage = null;
try {
  // Get the Web service interface to operate on.
  pushMessage = pushMessageService.getPushMessage();
} catch (ServiceException e) {
    e.printStackTrace();
    throw e;
  }
SendPushMessageResponse sendPushMessageResponse = null;
try {
  // Send the WAP Push message.
  sendPushMessageResponse = pushMessage.sendPushMessage(sendPushMessage);
} catch (RemoteException e) {
  e.printStackTrace();
  throw e;
} catch (PolicyException e) {
  e.printStackTrace();
  throw e;
} catch (com.bea.wlcp.wlng.schema.ews.common.ServiceException e) {
  e.printStackTrace();
  throw e;
}
// Assign the pushId provided in the in the response to a local variable.
String pushId = sendPushMessageResponse.getPushId();
```

30

Adding Subscriber Profile Extended Web Service Support

This chapter explains how to use the Oracle Communications Services Gatekeeper subscriber profile extended web service interface to add subscriber profile information applications.

Understanding the Subscriber Profile Extended Web Service Interface

The Subscriber Profile Extended Web Service interface allows an application to get subscriber-specific data from data sources within the network operator's domain.

Examples of data sources are subscriber databases containing information about terminal types in use, preferred language, and currency types. This information can be used by applications in order to control rendering options for rich media, charging information, and the language to be used in voice and text interaction with the end-user.

The interface is built around a model where the data can be retrieved in two different ways:

- Individual attributes, identified using a path.
- A collection of attributes.

The attributes are keyed on a subscriber ID that uniquely identifies the subscriber for whom the attributes are valid or by an address that uniquely identifies the terminal for which the attributes are valid. An attribute is identified by a path name, which corresponds to a specific property. The following is an example of a path name:

serviceName/accessControlId/accessControlId

The syntax for the path is similar to relative file system paths in UNIX.

A collection of attributes is specified in a subscriber profile filter for the application or the service provider. Only allowed attributes, as specified in the filter, are returned.

The returned attributes are returned in the form of name-value pairs, or property tuples, where the name is expressed as a path name with a associated property value.

The interface is based on a proposal for a Parlay X Subscriber Profile Web service interface.

Namespaces

The SubscriberProfile interface and service use these namespaces:

- `http://www.bea.com/wlcp/wlng/wsdl/ews/subscriber_profile/interface`

- `http://www.bea.com/wlcp/wlng/wsdl/ews/subscriber_profile/service`

The data types are defined in the namespace:

- `http://www.bea.com/wlcp/wlng/schema/ews/subscriber_profile`

In addition, the Subscriber Profile Extended Web Service interface uses definitions common for all Extended Web Services interfaces:

- The datatypes are defined in the namespace:
  - `http://www.bea.com/wlcp/wlng/schema/ews/common`
- The faults are defined in the namespace:
  - `http://www.bea.com/wlcp/wlng/wsdl/ews/common/faults`

Endpoint

The endpoint for the PushMessage interface is: `http://`*host*`:`*port*`/ews/subscriber_profile/SubscriberProfile`

Where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.

Address schemes

Table 30–1 Supported Address Schemes

Address scheme	Valid for Communication service
tel	Subscriber Profile Extended Web Service profile for LDAPv3
id	Subscriber Profile Extended Web Service profile for LDAPv3
imsi	Subscriber Profile Extended Web Service profile for LDAPv3
ipv4	Subscriber Profile Extended Web Service profile for LDAPv3

XML Schema data type definition

The following data structures are used in the Subscriber Profile Extended Web Service.

PropertyTuple Structure

Defines the response that Services Gatekeeper returns from "Operation: get" and "Operation: getProfile".

Table 30–2 PropertyTuple Structure

Element Name	Element type	Optional	Description
pathName	xsd:string	N	The key of the name-value pair. Expressed as a relative UNIX path. Example: serviceName/accessControlId/accessControlId
propertyValue	xsd:string	N	The value associated with the key.

WAP Push Extended Web Service Interface Descriptions

The following describes the interfaces and operations that are available in the Subscriber Profile Extended Web Service.

Interface: SubscriberProfile

Operations to obtain specific subscriber profile attributes and operations to obtain a set of profile properties grouped together in a profile.

Operation: get

Gets specific subscriber profile attributes. The requested attributes are identified by the pathNames parameter, and the possible values are restricted by the configured capabilities of the underlying data source. The allowed path name values are also restricted individually per service provider and application in the SLA.

Input message: get

Table 30–3 Input Message: get

Part name	Part type	Optional	Description
address	xsd:anyURI	N	Identity to get profile attributes for.
pathNames	xsd:string [1..unbounded]	N	Requested subscriber properties. Expressed as a relative UNIX path. Example: serviceName/accessControlId/accessControlId

Output message: getResponse

Table 30–4 Output Message: getResponse

Part name	Part type	Optional	Description
properties	PropertyTuple [1..unbounded]	N	All retrieved subscription property name and value pairs which are requested by application and allowed by the usage policies as specified in a filter. See "PropertyTuple Structure".

Referenced faults

Table 30–5 Exceptions and Error Codes

Exception	Error code	Reason/Action
ESVC0001	WNG000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
ESVC0001	SP000001	Internal problem in Services Gatekeeper. The LDAP connection is not working. There might be a configuration error for with the underlying LDAP server or a network error. Contact your Services Gatekeeper administrator.

Table 30–5 (Cont.) Exceptions and Error Codes

Exception	Error code	Reason/Action
ESVC0001	SP000002	Internal problem in Services Gatekeeper. LDAP operation failed. Contact your Services Gatekeeper administrator.
ESVC0001	SP000003	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
ESVC0001	SP000004	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.

Operation: getProfile

Gets a set of profile properties grouped together in a profile identified by a certain profile ID.

Input message: getProfile

Table 30–6 Input Message: getProfile

Part name	Part type	Optional	Description
subscriberID	xsd:string	N	Identity to get profile attributes for.
profileID	xsd:string	N	Identity of the profile to get.

Profile ID is ignored when connecting the to the network using the LDAPv3 network protocol plug-in. The collection of attributes that identifies the profile are provisioned as filters.

Output message: getProfileResponse

Table 30–7 Output Message: getProfileResponse

Part name	Part type	Optional	Description
properties	PropertyTuple [1..unbounded]	N	All retrieved subscription property name and value pairs which are requested by application and allowed by the usage policies as specified in a filter. See "PropertyTuple Structure".

Referenced faults

Table 30–8 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.

Table 30–8 (Cont.) Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	SP-000001	Internal problem in Services Gatekeeper. The LDAP connection is not working. There might be a configuration error for with the underlying LDAP server or a network error. Contact your Services Gatekeeper administrator.
SVC0001	SP-000002	Internal problem in Services Gatekeeper. LDAP operation failed. Contact your Services Gatekeeper administrator.
SVC0001	SP-000003	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0001	SP-000004	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
SVC0001	PLG-000004	General plug-in routing error

WSDLs

The document/literal WSDL representation of the SubscriberProfile interface can be retrieved from the Web services endpoint, see "Endpoint", or:

- http://*host*:*port*/ews/subscriber_profile/SubscriberProfile?WSDL
- http://*host*:*port*/ews/subscriber_profile/SubscriberProfile?WSDL/ews_subscriber_profile_interface.wsdl
- http://*host*:*port*/ews/subscriber_profile/SubscriberProfile?WSDL/ews_common_types.xsd

Where *host* and *port* are the host name and port of the system on which Services Gatekeeper is installed.