Oracle® Communications Services Gatekeeper

Application Developer's Guide

Release 5.1

E37542-01

June 2013

ORACLE®

Oracle Communications Services Gatekeeper Application Developer's Guide, Release 5.1

E37542-01

Contents

9 Parlay X 3.0 Interfaces

10 Native Interfaces

Preface

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

This document covers the SOAP-based interface and the native interface for Services Gatekeeper. A separate document in this set, *Oracle Communications Services Gatekeeper RESTful Application Developer's Guide*, describes how to use the RESTful interface.

Audience

This book if intended for software developers who wish to integrate functionality provided by telecom networks into their programs using the SOAP-based or 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 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

For more information, see the following documents in the Services Gatekeeper set:

- *Oracle Communications Services Gatekeeper Accounts and SLAs Guide*
- *Oracle Communications Services Gatekeeper Alarm Handling Guide*
- *Oracle Communications Services Gatekeeper Communication Service Guide*
- *Oracle Communications Services Gatekeeper Concepts Guide*
- *Oracle Communications Services Gatekeeper Licensing Guide*
- *Oracle Communications Services Gatekeeper Partner Relationship Management Guide*

- *Oracle Communications Services Gatekeeper Platform Development Studio Developer's Guide*
- *Oracle Communications Services Gatekeeper Platform Test Environment Guide*
- *Oracle Communications Services Gatekeeper RESTful Application Developer's Guide*
- *Oracle Communications Services Gatekeeper SDK User's Guide*
- *Oracle Communications Services Gatekeeper Statement of Compliance*

1

Creating Applications for Oracle Communications Services Gatekeeper

As the worlds of Internet applications and of telephony-based functionality continue to converge, many application developers have become frustrated by the unfamiliar and often complex telecom interfaces that they need to master to add even simple telephony-based features to their programs. By using Oracle Communications Services Gatekeeper, telecom operators can instead offer developers a secure, easy-to-develop-for single point of contact with their networks, made up of simple Web Service interfaces that can easily be accessed from the Internet using a wide range of tools and languages.

The following chapter presents an overview of Services Gatekeeper's functionality, and the ways that application developers can use this functionality to simplify their development projects.

Basic Concepts

There are a few basic concepts you need to understand to create applications that can interact with Services Gatekeeper:

- Communication Services
- Traffic Types
  - Application-initiated Traffic
  - Network-triggered Traffic
- Management Structures

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, etc.), an application-facing interface (also called a "north" interface), and a network-facing interface (also called a "south" interface). A request for service enters through one interface, is subjected to internal processing, including evaluation for policy and protocol translation, and is then sent on using the other interface.

Note: Because a single application-facing interface may be connected to multiple protocols and hardware types in the underlying telecom network, it's important to understand that an application is communicating, finally, with a specific communication service, and not just the north interface. So in some cases it is possible that an application request sent to two different carriers, with different underlying network structures, might behave in slightly different ways, even though the initial request uses exactly the same north interface.

Traffic Types

In some Services Gatekeeper communication services, request traffic can travel in two directions - from the application to the underlying network and from the underlying network to the application - and in others traffic flows in one direction only.

Application-initiated Traffic

In application-initiated traffic, the application sends a request to Services Gatekeeper, the request is processed, and a response of some kind is returned synchronously. So, for example, an application could use the Third Party Call interface to set up a call. The initial operation, MakeCall, is sent to Services Gatekeeper (which sends it on to the network) and 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 the CallIdentifier to identify the specific call, and then receives the requested information back from Services Gatekeeper synchronously.

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 or not the message has been delivered, using GetSmsDeliveryStatus, which functions much like GetCallInformation described above. But given the possibly extended period of time involved, it would be convenient simply to have the network notify the application once delivery to the mobile phone has been accomplished. To do this, two things must happen:

- The application must inform Services Gatekeeper that it wishes to receive information that originates from the network. It does this by subscribing or registering for notifications via an application-initiated request. (In certain cases, this can also be accomplished by the operator, using OAM procedures.) Often this subscription includes filtering criteria that describes exactly what kinds of traffic it wishes 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, as described above, 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 for only limited amounts of time.

Management Structures

In order 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 which is tied to an Application Account. All the Application Accounts that belong to a single entity are assigned to a Service Provider Account. Application Accounts with similar requirements are put into Application Groups and Service Providers with similar requirements are put into Service Provider Groups. Each Application Group is associated with an Application Group Service Level Agreement (SLA) and each Service Provider Group are associated with Service Provider Group SLAs. These SLAs define and regulate the contractual agreements between the telecom operator and the application service provider, and cover such things as which services the application may access and the maximum bandwidth available for use.

Functional Overview

Services Gatekeeper allows operators to provide client application developers with a choice of interface types, based on the needs of their applications. Services Gatekeeper provides SOAP-based Web Services interfaces, RESTful interfaces (see *Oracle Communications Services Gatekeeper RESTful Application Development Guide*), and even two native telephony interfaces (MM7 and SMPP).

Note: The exact mix of interfaces depends on the specific Services Gatekeeper installation.

The SOAP-based Web Services APIs are based on the Parlay X 2.1 and 3.0 standards and also include three additional Extended Web Services ones to cover Binary SMS, Subscriber Profile, and WAP Push functionality, which are not supported by Parlay X. The functionality supported by these communication services includes:

- Third Party Call (Parlay X 2.1 and 3.0)

  Using this communication service, an application can set up a call between two parties (the caller and the callee), poll for the status of the call, and end the call. In addition, using the 3.0 communication only, an application can set up a call among multiple participants and add, delete, or transfer those participants. The application can also 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.

- Audio Call (Parlay X 2.1 and 3.0)

  Using this communication service, an application can play audio to one or more call participants in an existing call session set up by PX 3.0 Third Party call, 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 in conjunction with a notification set up using Call Notification PX 3. 0, 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 this communication service, an application can set up and end notifications on call events, such as a callee in a third party call attempt is busy. In addition, in some cases the application can then reroute the call to another party. In addition, using the PX 3.0 communication service, an application can interact with PX 3.0 Audio Call to return digits collected from a call participant back to the application and to end calls.

- Device Capabilities (Parlay X 3.0)

  Using this communication service, an application can request a device's unique device ID, device/model name, and a link to the User Agent Profile XML file, or device's equipment identifier.

- Short Messaging (Parlay X 2.1)

  Using this communication service, 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 SMSs meeting particular criteria from the network.

- Multimedia Messaging (Parlay X 2.1)

  Using this communication service, 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 MMSs meeting particular criteria from the network or to poll for such messages.

- Terminal Location (Parlay X 2.1)

  Using this communication service, an application can request the position of one or more terminals or the distance between a given position and a terminal. It 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)

  Using this communication service, an application can be a watcher for presence information or a presentity (an end user who has agreed to have certain data, such as current activity, available communication means, and contact addresses made available to others). So a presentity might say that at this moment he is in the office and prefers to be contacted by SMS at this number. Before the watcher can receive this information, it must subscribe and be approved by the presentity. Once this is done, the watcher can either poll for specific presentity information, or set up status notifications based on a wide range of criteria published by the presentity. The presentity can control the kinds of information, or attributes, that he makes available to watchers.

- Payment (Parlay X 3.0)

  Using the communication service based on this interface, an application can charge an end user's account a specific amount, refund an amount, and split costs among multiple end user accounts. An application can also reserve an amount in

an account, extend the amount associated with that reservation, make a charge against that reservation, and release the reservation.

- Binary SMS (EWS)

  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 Oracle Extended Web Services (EWS) set.

- WAP Push (EWS)

  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. 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.

- Subscriber Profile (EWS)

  The application-facing interface of this communication service is based on a subset of that in a proposed Parlay X version. Using this communication service, 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.

- Session Manager (EWS)

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

There are three native telephony APIs supported by Services Gatekeeper.

- Native MM7

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

- Native SMPP

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

- Native UCP

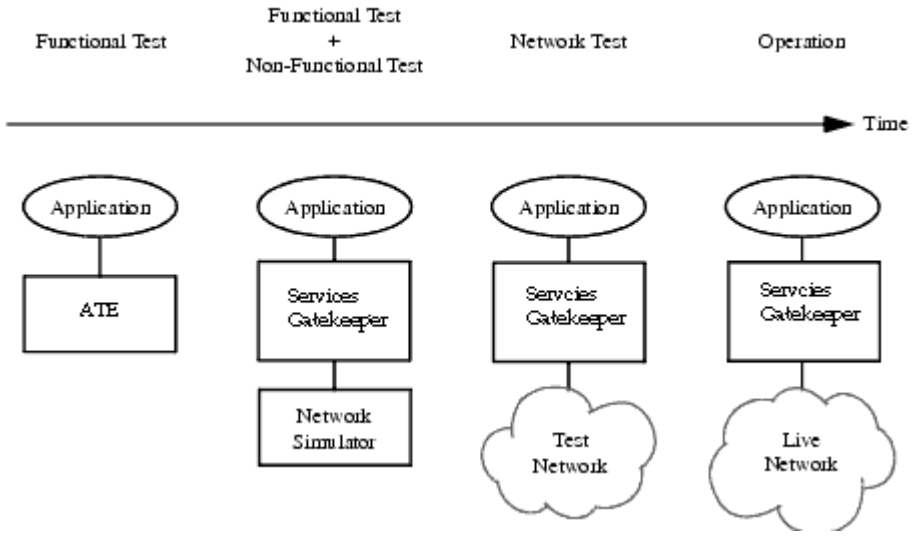
  The application-facing interface of this communication service is based on the Short Message Service Centre EMI-UCP Interface 5.1 specification. Using the communication service based on this interface, an application can send and receive SMSs and receive status notifications about previously sent messages.

Application Testing Workflow

Application testing in a telecom environment is usually conducted in a stepwise manner. For the first step, applications are run against simulators like the optional Services Gatekeeper Simulator. The Services Gatekeeper Simulator emulates both the Services Gatekeeper and the underlying network, and allows developers to sort out basic functional issues without having to be connected to a network or network simulator. Once basic functional issues are sorted through, the application is connected to an instance of the Services Gatekeeper attached to a network simulator for

non-functional 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 complete application test flow, from the developer's functional tests to deployment in a live network. While Services Gatekeeper Simulator-based tests may be performed in-house by an Application Service Provider, the other tests require the cooperation of the target network operator.

Figure 1–1 Application Testing Cycle



2

Interacting with Oracle Communications Services Gatekeeper

This chapter starts by explaining how Services Gatekeeper manages SOAP, REST, and OneAPI sessions. It then presents a high-level description of the SOAP mechanisms and how they function to manage the interaction between Services Gatekeeper and the application.

In order to interact with Oracle Communications Services Gatekeeper (Services Gatekeeper), applications use SOAP-based, RESTful, OneAPI, or 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 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.

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 SOAP attachments programmatically".

Session Management for SOAP, REST, and OneAPI Interfaces

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 it is 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 of the traffic for a session.

An application establishes a session in Services Gatekeeper by invoking the `getSession()` operation in the Session Manager Web Service. `getSession()` is the only request that does not require a session ID to execute. This operation returns a string

representing the session ID to the client, and Services Gatekeeper establishes a session identified by the ID string. Sessions last until they time out, based on an operator-set interval, or until the application closes the session. 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.

You have these options for session checking:

- **Required** - The default value. Requires all applications authorize themselves with credentials before requesting a session ID. Services Gatekeeper validates the session IDs and rejects the communication attempt if it is invalid. The session ID is a requirement 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 simply uses whichever 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.

Example 2–1 Example of a SessionId header element

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

Changing the Session Mode

To change the Services Gatekeeper session mode:

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

   The JConsole and PTE are supplied with Services Gatekeeper for this purpose.

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

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

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

   See "Session Management for SOAP, REST, and OneAPI Interfaces" for details on the different options.

Requirements for Using the SOAP-based Facades

If your application is using the a SOAP-based facade (set of 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.

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 framework to process this information.

> **Note:** WS Security provides three separate modes of providing security between a Web Service client application and the Web Service itself for message level security - Authentication, Digital Signatures, and Encryption. For an overview of securing web services, see "Securing and Administering WebLogic Web Services" in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services* at:
>
> http://download.oracle.com/docs/cd/E15523_01/web.1111/b32511/weblogic.htm

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 2–2 WSDL fragment showing Policy

```
<s0:Policy s1:Id="Auth.xml">
<wssp:Identity>
<wssp:SupportedTokens>
<wssp:SecurityToken
TokenType="http://docs.oasisopen.org/wss/2004/01/oasis200401wssusernametokenprofil
e1.0#UsernameToken">
<wssp:UsePassword
Type="http://docs.oasisopen.org/wss/2004/01/oasis200401wssusernametokenprofile1.0#
PasswordText"/>
</wssp:SecurityToken>
<wssp:SecurityToken
TokenType="http://docs.oasisopen.org/wss/2004/01/oasis200401wssx509tokenprofile1.0
#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 username, 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 2–3 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 "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 UserName, 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.

Example 2–4 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 and 1.1. The versions are similar. For an overview of the differences between the versions, 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 2–5 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
```

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 the pizza place 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 place, 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 2–6 Example of a SCID SOAP header element

```
<env:Header>
    <wsse:Security
    . . .
    </wsse:Security>
    <session
    . . .
    </session>
    <scid
    . . .
    </scid>
</env:Header>
```

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 the it.

See the appropriate sections in *Oracle Communications Services Gatekeeper Communication Service Guide* for descriptions of the tunneled parameters that are applicable to your communication service.

The application sends the tunneled parameters in the SOAP header of a Web Services request.

The parameters are defined using key-value pairs encapsulated by the tag `<xparams>`. The xparams tag can include one or more `<param>` tags. Each `<param>` tag has a **key** attribute that identifies the parameter and a `value` attribute that defines the value of the parameter. In the example below, the application tunnels the parameter aParameterName and assigns it the value aParameterValue.

Example 2–7 SOAP header with a tunneled parameter.

```
<soapenv:Header>
...
  <xparams>
    <param key="aParameterName" value="aParameterValue" />
  </xparams>
...
</soapenv:Header>
```

Depending on the plug-in the request reaches, the parameter is fetched and used in the request towards the network node.

The operator can block requests that contain tunneled parameters that have not been configured as allowed. Filtering is on a global, not application, level. See *Oracle Communications Services Gatekeeper System Administrator's Guide* for more information.

SOAP attachments

In some communication services, the request payload are sent as SOAP attachments. Example 2–8 below shows a Multimedia Messaging sendMessage operation that contains an attachment carrying a jpeg image.

Example 2–8 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 SOAP 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, the steps you need to take to accomplish these tasks will be different.

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

The following examples show particularly the use of a SOAP message handler.

These examples show the use of a single message handler to add both SOAP Headers and SOAP attachments.

The WebLogic Server environment relies heavily on using supplied Ant tasks. In Example 2–9 a supplied Ant task, clientgen, is added to the standard build.xml file. A handler configuration file, SOAPHandlerConfig.xml is added as the value for the `handlerChainFile` attribute. SOAPHandlerConfig.xml is shown in Example 2–10.

Example 2–9 Snippet from build.xml

```
<clientgen
  wsdl="${wsdl-file}"
  destDir="${class-dir}"
  handlerChainFile="SOAPHandlerConfig.xml"
  packageName="com.bea.wlcp.wlng.test"
  autoDetectWrapped="false"
  generatePolicyMethods="true"
/>
```

The configuration file for the message handler contains the handler-name and the associated handler-class. The handler class, TestClientHandler, is described in Example 2–11.

Example 2–10 SOAPHandlerConfig.xml

```
<weblogic-wsee-clientHandlerChain
  xmlns="http://www.bea.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, REST, and OneAPI Interfaces". The session ID is hardcoded into the member variable `sessionId`.
- Adds a service correlation ID to the SOAP header. See "Service Correlation" for more information.
- Adds a SOAP attachment in the form of a MIME message with content-type text/plain. See "SOAP attachments" for more information.

Example 2–11 TestClientHandler

```
package com.bea.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;
  }
}
```

3

Session Manager Web Service

The Session Manager Web Service contains operations for establishing a session with Oracle Communications Services Gatekeeper, 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 on 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 towards Services Gatekeeper.

Endpoint

The WSDL for the Session Manager can be found at

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

where *host* and *port* depend on the Services Gatekeeper deployment.

Interface: SessionManager

Operations to establish a session, change a password, get the remaining lifetime of a session, refresh a session and destroy a session.

Operation: getSession

Establishes a session using Web Services Security. Authentication information must be provided according to WS-Security. See "Authentication".

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

-

Operation: getSessionRemainingLifeTime

Gets the remaining lifetime of an established session. The default lifetime is configured in Services Gatekeeper.

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

-

Operation: refreshSession

Refreshes the lifetime of an session. The session can be refreshed during a time interval after the a 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

-

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

-

Examples

The code below illustrates how to get the Session Manager and how to prepare the generated stub with Web Service security information. The stub is generated from the Session Manager Web Service.

Example 3–1 Get hold of the Session Manager

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

Below illustrates how to prepare the Session Manager stub with Username Token information according to WS-Policy.

Example 3–2 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);
  }

}
```

4

Extended Web Services Binary SMS

The Extended Web Services Binary SMS Web Service allows for the sending and receiving of any generic binary content via SMS. Both application-initiated and network-triggered requests are supported. The binary content can be other than the Logos and Ringtones as specified by Parlay X Short Messaging. An example would be a vCard.

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 "Extended Web Services Common".

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_notification/BinarySmsNotificationManager

The values for host and port depend on the specific Services Gatekeeper deployment.

Sequence Diagram

This section explains the sequence diagrams for sending and receiving an SMS.

Send SMS

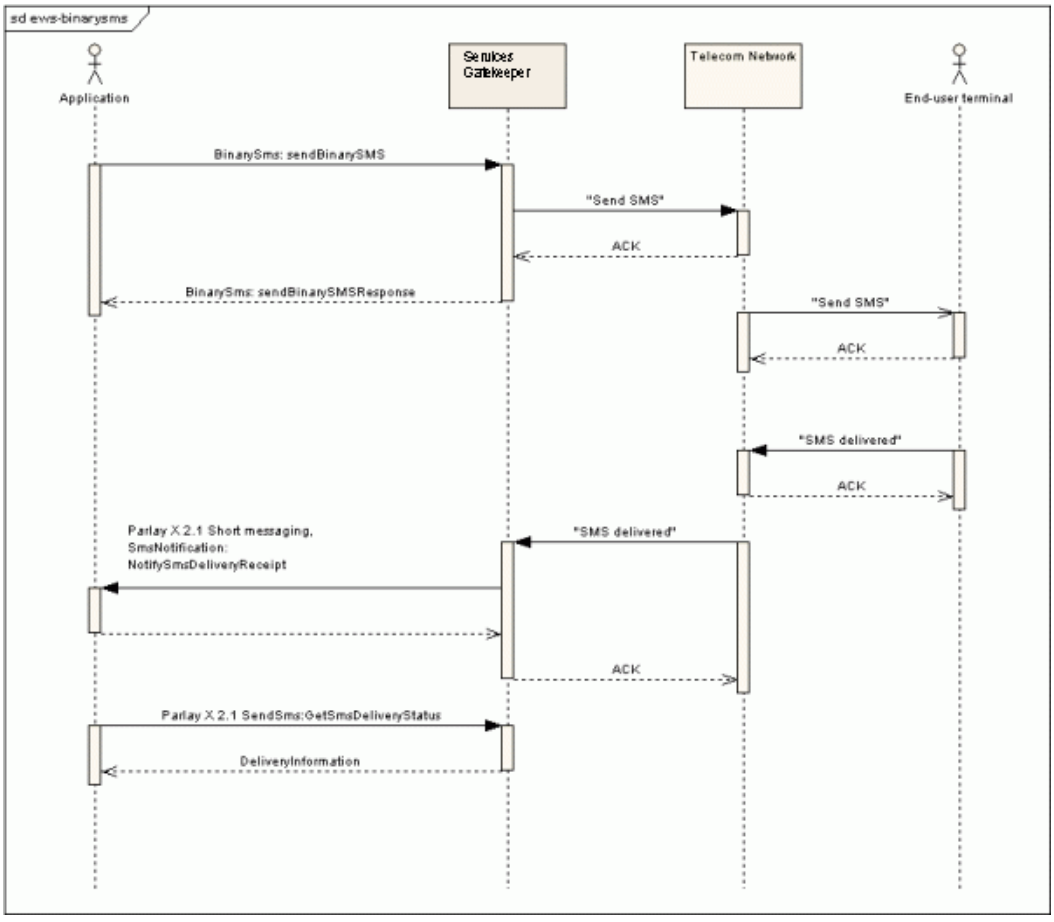
The following diagram shows the general message sequence for sending a binary SMS message from an Extended Web Services Binary SMS application to the network. In

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 4–1 Sequence diagram Application-initiated send Extended Web Services Binary SMS

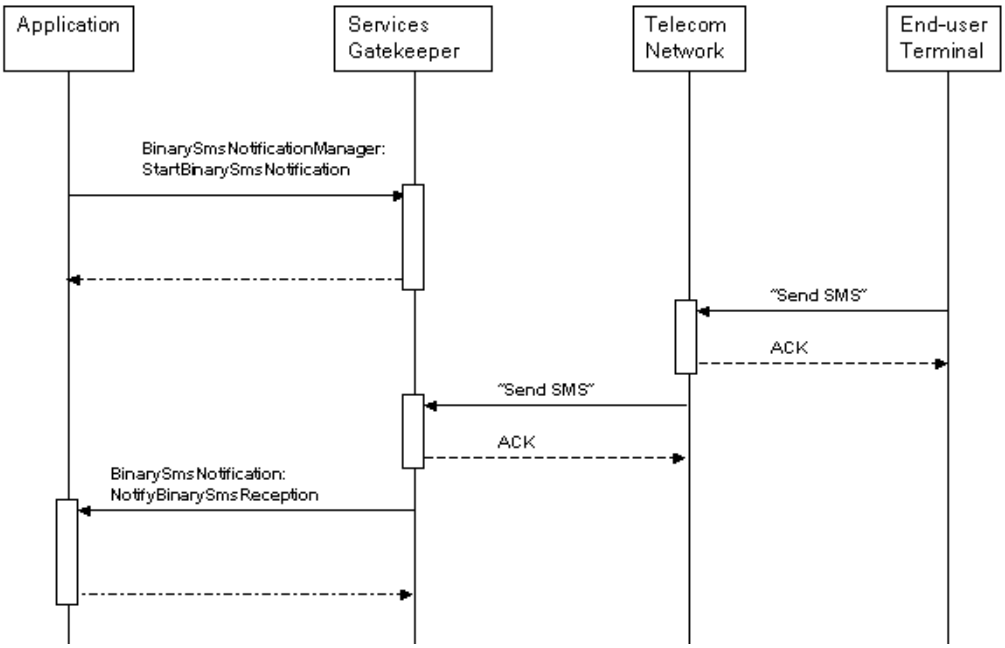


Receive SMS

The following diagram 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 4–2 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 4–1 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 4–2 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 4–3 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 4–3 (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 4–4 Output message: sendBinarySMSResponse

Part name	Part type	Optional	Description
result	xsd:string	N	Identifies a specific SMS delivery request.

Referenced faults

Table 4–5 exceptions and error codes

Exception	Error code	Reason/Action
SVC0001	BSMS-000001	Unable to perform action. Network error

Table 4–5 (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 4–6 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 4–7 Output message: StartBinarySmsNotificationResponse

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

Referenced faults

Table 4–8 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 4–9 Input message: StopBinarySmsNotification

Part name	Part type	Optional	Description
correlator	xsd:String	N	The identifier for the subscription.

Output message: StopBinarySmsNotificationResponse

Table 4–10 Output message: StopBinarySmsNotificationResponse

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

Referenced faults

Table 4–11 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 4–11 (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 4–12 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 4–13 Output message: NotifyBinarySmsReceptionResponse

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

Referenced faults

None.

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_notification/wsdls/ews_binary_sms_notification_service.wsdl`
`http://host:port/ews/binary_sms_notification/wsdls/ews_binary_sms_notification_interface.wsdl`
Where host and port are depending 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 4–1 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);
```

5

Extended Web Services WAP Push

The Extended Web Services WAP Push Web Service allows for the sending of 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:

- 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

http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

for links to the specifications.

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, Extended Web Services WAP Push 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

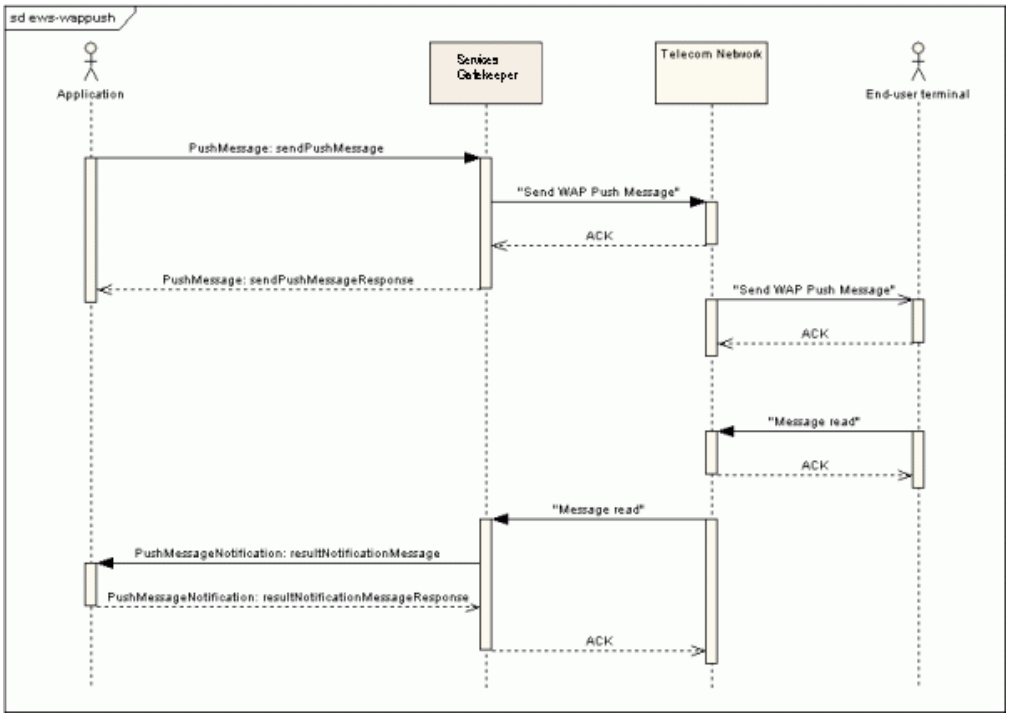
Where host and port depend on the Oracle Communications Services Gatekeeper deployment.

Sequence Diagram

The following diagram shows the general message sequence for sending a WAP Push message from an Extended Web Services WAP Push application 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 5–1 Sequence diagram Extended Web Services WAP Push



XML Schema data type definition

The following data structures are used in the Extended Web Services WAP Push Web Service.

PushResponse structure

Defines the response that the Services Gatekeeper returns from a sendPushMessage operation.

Table 5–1 PushResponse structure

Element Name	Element type	Optional	Description
result	push_message_xsd:ResponseResult	N	The ResponseResult allows the server to specify a code for the outcome of sending the push message. See "ResponseResult structure"
pushId	xsd:string	N	The push ID provided in the request.
senderAddress	xsd:string	Y	Contains the address to which the message was originally sent, for example the URL to the network node.
senderName	xsd:string	Y	The descriptive name of the server.
replyTime	xsd:dateTime	Y	The date and time associated with the creation of the response.
additionalProperties	ews_common_xsd:AdditionalProperty	Y	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 5–2 ResponseResult structure

Element Name	Element type	Optional	Description
code	xsd:string	N	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	N	Textual description.

Table 5–3 Outcome status codes

Status code	Description
1000	OK.
1001	Accepted for processing.
2000	Bad request.
2001	Forbidden.
2002	Address error.
2003	Address not found.

Table 5–3 (Cont.) Outcome status codes

Status code	Description
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 5–4 ReplaceMethod enumeration

Enumeration value	Description
all	Indicates that this push message MUST be treated as a new push submission for all recipients, no matter if a previously submitted push message with pushId equal to the replacePushId in this push message can be found or not.
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 5–5 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.

Web Service interface description

The following describes the interfaces and operations that are available in the Extended Web Services WAP Push 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 5–6 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"

Table 5–6 (Cont.) Input message: sendPushMessage

Part name	Part type	Optional	Description
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.
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:ReplaceMethod	N	Defines how to replace a previously sent message. Used in conjunction with the replacePushId parameter described above. Ignored if replacePushId is NULL.

Table 5–6 (Cont.) Input message: sendPushMessage

Part name	Part type	Optional	Description
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.
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 via 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:Additional Property [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 5–7 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 5–8 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 5–9 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.

Table 5–9 (Cont.) Input message: resultNotificationMessage

Part name	Part type	Optional	Description
additionalProperties	ews_common_xsd:AdditionalProperty	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 5–10 Output message: resultNotificationMessageResponse

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

Referenced faults

Table 5–11 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 depending on the Services Gatekeeper deployment.

Sample Send WAP Push Message

Example 5–1 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();
```

6

Extended Web Services Subscriber Profile

The Extended Web Services Subscriber Profile Web Service 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 the 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, Extended Web Services Subscriber Profile 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 depend on the Oracle Communications Services Gatekeeper deployment.

Address schemes

Table 6–1 Supported address schemes

Address scheme	Valid for Communication service
tel	Extended Web Services Subscriber profile for LDAPv3
id	Extended Web Services Subscriber profile for LDAPv3
imsi	Extended Web Services Subscriber profile for LDAPv3
ipv4	Extended Web Services Subscriber profile for LDAPv3

XML Schema data type definition

The following data structures are used in the Extended Web Services Subscriber Profile Web Service.

PropertyTuple structure

Defines the response that the Services Gatekeeper returns from "Operation: get" and "Operation: getProfile".

Table 6–2

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/access ControlId
propertyValue	xsd:string	N	The value associated with the key.

Web Service interface description

The following describes the interfaces and operations that are available in the Extended Web Subscriber Profile 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 6–3 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 6–4 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 6–5 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.
ESVC0001	SP000002	Internal problem in Services Gatekeeper. LDAP operation failed. Contact your Services Gatekeeper administrator.

Table 6–5 (Cont.) Exceptions and error codes

Exception	Error code	Reason/Action
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 6–6 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 6–7 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 6–8 exceptions and error codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
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.

Table 6–8 (Cont.) exceptions and error codes

Exception	Error code	Reason/Action
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 depend on the Services Gatekeeper deployment.

7

Extended Web Services Common

This chapter describes the Extended Web Services set of Web Services share common definitions.

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 7–1 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 7–2 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 7–2 (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 contained in this type.

Table 7–3 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 7–4 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 Exception 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 7–5 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

8

Parlay X 2.1 Interfaces

This chapter describes the supported Parlay X 2.1 interfaces and contains information specific for Services Gatekeeper and not found in the specifications. For detailed descriptions of the interfaces, methods, and parameters, refer to the specifications.

See the ETSI OSA Parlay X web site:

http://www.etsi.org/deliver/etsi_es/202300_202399/20239102/01.02.01_60/es_20239102v010201p.pdf

for links to the specifications.

Parlay X 2.1 Part 2: Third Party Call

This set of interfaces 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* depend on the Services Gatekeeper deployment.

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* depend on the Services Gatekeeper deployment.

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, prior to 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* depend on the Services Gatekeeper deployment.

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, prior to call setup.

Interface: CallNotificationManager

The endpoint for the **CallNotificationManager** interface is:
`http://host:port/parlayx21/call_notification/CallNotificationManager`

Where values for *host* and *port* depend on the Services Gatekeeper deployment.

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* depend on the Services Gatekeeper deployment.

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* depend on the Services Gatekeeper deployment.

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 of time.

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* depend on the Services Gatekeeper deployment.

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* depend on the Services Gatekeeper deployment.

GetReceivedSms

Gets messages that have been received by Services Gatekeeper. The SMSs are retrieved using a registrationIdentifier used when the notification was registered using a provisioning step in Services Gatekeeper.

Interface: SmsNotificationManager

The endpoint for the **SmsNotificationManager** interface is:
`http://host:port/parlayx21/sms/SmsNotificationManager`

Where values for *host* and *port* depend on the Services Gatekeeper deployment.

StartSmsNotification

Initiates notifications to the application for a given service activation number and criteria.

Note: A Service activation number may be provisioned to cater to 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 *System Administrator's Guide*. For an overview of Split and Submit Messaging, see *Communication Service 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 8–1 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* depend on the Services Gatekeeper deployment.

SendMessage

Sends a multimedia message. The content of the message is sent as a SOAP attachment. Sending as email is not supported.

Table 8–2 Exceptions and Error Codes

Exception	Error code	Reason/Action
SVC0001	WNG-000002	Internal problem in Services Gatekeeper. Contact your Services Gatekeeper administrator.
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 of time.

Note: Storing delivery status for an MMS is configurable in Services Gatekeeper.

Table 8–3 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* depend on the Services Gatekeeper deployment.

GetReceivedMessages

Polls Services Gatekeeper for received messages.

The registrationIdentifier is required. Received message are stored in Services Gatekeeper only for a configurable period of time.

Table 8–4 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 8–5 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* depend on the Services Gatekeeper deployment.

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* depend on the Services Gatekeeper deployment.

StartMessageNotification

Initiates notifications to the application for a given service activation number and criteria.

Table 8–6 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 cater to 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 8–7 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 8–8 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 8: Terminal Status

This set of interfaces is compliant with ETSI ES 202 391-8 V1.2.1, Open Service Access (OSA); Parlay X Web Services; Part 8: Terminal Status (Parlay X 2).

Interface: TerminalStatus

The endpoint for the **TerminalStatus** interface is:

`http://host:port/parlayx21/terminal_status/TerminalStatus`

Where values for *host* and *port* depend on your Services Gatekeeper deployment.

getStatus

Gets the status for a single terminal.

getStatusForGroup

Gets the status for multiple terminals.

Interface: TerminalStatusNotificationManager

The endpoint for the **TerminalStatusNotificationManager** interface is:

`http://host:port/parlayx21/terminal_status/TerminalStatusNotificationManager`

Where values for *host* and *port* depend on the Services Gatekeeper deployment.

startNotification

Starts the terminal status change subscription.

endNotification

Notifies the Terminal Status communication service that it can cancel the terminal status change subscription.

Interface: TerminalNotification

The endpoint for the **TerminalNotification** interface is provided by the calling application.

statusNotification

Supplies the terminal status to the application. The status is sent by the Terminal Status communication service to the application.

statusError

Informs the application that the Terminal Status could not be obtained and contains an error message indicating why. Sent by the Terminal Status plug-in to the application.

statusEnd

Informs the application that the status notification subscription has ended. Sent by the Terminal Status plug-in to the application.

Error Codes

See the General Exceptions and General error codes sections for most error information. Table 8–9 includes additional diagnostic information specific to the Terminal Status/MAP plug-in.

Table 8–9 Terminal Status Exception and Error Code Information

Exception	Error code	Reason/Action
POL0002	N/A	A Terminal Status request was sent using a criteria of `busy`, in violation of a `BusyAvailable = false` is setting in an SLA.
POL0003	N/A	The number of Terminal Server addresses sent exceeded the `MaximumNotificationAddresses` setting in an SLA.
POL0004	N/A	An unlimited number of Terminal Status requests were sent, but the SLA `UnlimitedCountAllowed` was set to `False`.
POL0005	N/A	The number of Terminal Status requests exceeded the `MaximumCount` setting in an SLA.
POL0009	N/A	The Terminal Status request frequency violated the `MaximumNotificationFrequency` setting in an SLA.
POL0200	N/A	Busy criteria not supported.

Table 8–9 (Cont.) Terminal Status Exception and Error Code Information

Exception	Error code	Reason/Action
SVC0001	MAP-000001	The Terminal Status communication service could not encode the `anyTimeInterrogation` MAP message. 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.
SVC0001	SS7API-000001	The SS7 API failed to build and send a message to the SS7 stack. This usually means that the Terminal Status plug-in could not bind with the stack, or the bind was lost. Make sure that the stack is running and that the `CpUserId` and SS7 host/port/instance are correct.
SVC0001	SS7-000001	The SS7 stack received no answer or an incorrect answer. The SS7 stack did not send an `anyTimeInterrogation` call answer or the answer was incorrect. This probably indicates a problem with the SS7 stack. Make sure that the global title/SPC/SSN is configured correctly in the `NetworkSelection`. Start a SS7 stack trace and check the `ss7trace.log` file. See the SS7 documentation for more information.

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).

Interface: TerminalLocation

The endpoint for the **TerminalLocation** interface is:
`http://`*`host`*`:`*`port`*`/parlayx21/terminal_location/TerminalLocation`

Where values for *host* and *port* depend on the Services Gatekeeper deployment.

GetLocation

Gets the location of a terminal.

Table 8–10 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 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.

Table 8–10 (Cont.) Exceptions and Error Codes

Exception	Error code	Reason/Action
POL0230	N/A	The requested accuracy is not supported.

GetTerminalDistance

Gets the distance from a certain point to the location of a terminal.

Table 8–11 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.
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 8–12 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* depend on the Services Gatekeeper deployment.

StartGeographicalNotification

Initiates location notifications to the application when one or more terminals change their location according to a criteria.

Table 8–13 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 8–14 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 8–15 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.

Table 8–15 (Cont.) Exceptions and Error Codes

Exception	Error code	Reason/Action
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* depend on the Services Gatekeeper deployment.

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".

Parlay X 2.1 Part 11: Audio Call

This set of interfaces is compliant with ETSI ES 202 391-11 V1.2.1 (2006-12) Open Service Access (OSA); Parlay X Web Services; Part 11: Audio Call (Parlay X 2).

Interface: PlayAudio

The endpoint for the **PlayAudio** interface is: `http://host:port/parlayx21/audio_call/AudioCall`.

Where values for *host* and *port* depend on the Services Gatekeeper deployment.

EndMessage

Cancels previous play request.

GetMessageStatus

Requests the status of a previous play request.

PlayAudioMessage

Play an audio file (such as .WAV).

PlayTextMessage

Play text to a text-to-speech engine.

PlayVoiceXmlMessage

Play a VXML file.

Error Codes

See "General error codes".

Parlay X 2.1 Part 14: Presence

This set of interfaces is compliant with ETSI ES 202 391-14 V1.2.1 (2006-12), Open Service Access (OSA); Parlay X Web Services; Part 14: Presence (Parlay X 2).

Interface: PresenceConsumer

The endpoint for the **PresenceConsumer** interface is:
`http://host:port//parlayx21/presence/PresenceConsumer`

Where values for *host* and *port* depend on the Services Gatekeeper deployment.

subscribePresence

Requests a subscription for presence information about a presentity.

For the parameter presentity, only SIP URI can be used. Group-URI is not supported.

getUserPresence

Gets presence information about a presentity.

For the parameter presentity, only SIP URI can be used. Group-URI is not supported.

startPresenceNotification

Initiates presence notifications to the application when one or more presence attributes changes for a presentity.

For the parameter presentity, only SIP URI can be used. Group-URI is not supported.

The parameter frequency is not supported. The application is notified when an update of presence information is received from the network.

endPresenceNotification

Ends a previously started notification.

Interface: PresenceNotification

The **PresenceNotification** 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/presence/wsdls/parlayx_presence_notification_interface_2_3.wsdl`

`http://host:port/parlayx21/presence/wsdls/parlayx_presence_notification_service_2_3.wsdl`

`http://host:port/parlayx21/presence/wsdls/parlayx_presence_types_2_3.xsd`

Where values for *host* and *port* depend on the Services Gatekeeper deployment.

statusChanged

Notifies an application about a change of presence attributes for a presentity.

statusEnd

Notifies an application that no more notifications will be sent to the application.

notifySubscription

Notifies an application that the presentity has handled the request for presence information.

subscriptionEnded

Notifies an application that the subscription for presence information has ended.

Interface: PresenceSupplier

The endpoint for the **PresenceSupplier** interface is: `http://host:port/parlayx21/presence/PresenceSupplier` where values for *host* and *port* depend on the Services Gatekeeper deployment.

The Presence Supplier interface requires that a Presence Server be available in the underlying network. Services Gatekeeper interacts with this server to provide this functionality to the application.

By default, Services Gatekeeper maps a presentity URI parameter to the client's Application Instance ID. This information is required to interact with the network and Parlay X 2.1 does not provide it by default.

The application can override this value using the "Parameter Tunneling" function. The param to use is key = "wlng.presence.parlay21.presentity.uri" and value = $the_desired_URI.

publish

Publishes presence data about a presentity.

Note: The Presence Server in the network must support `http://tools.ietf.org/html/draft-ietf-simple-partial-publish-07#page-4` for this functionality to work.

---

getOpenSubscriptions

Gets a list of new watchers who wish to subscribe to this presentity's data.

> **Note:** The Presence Server in the network must support http://tools.ietf.org/html/draft-ietf-simple-partial-pidf-format-10 and http://tools.ietf.org/html/rfc3857 (http://tools.ietf.org/html/draft-ietf-simple-winfo-package-05) for this functionality to work.

updateSubscriptionAuthorization

Adds new watchers and updates the permissions of existing watchers of this presentity's data. This is the usual follow-up to the getOpenSubscriptions operation.

> **Note:** This operation requires the presence of both a Presence Server and a Data Manipulation Server in the underlying network.

An SVC0220 service exception (NoSubscriptionRequest) is thrown if the presentity attempts to confirm the subscription of a watcher who has not asked to subscribe to the presentity's data.

getMyWatchers

Returns an array of the watchers that are subscribed to the presentity's data.

> **Note:** The Presence Server in the network must support http://tools.ietf.org/html/draft-ietf-simple-partial-pidf-format-10 and http://tools.ietf.org/html/rfc3857 (http://tools.ietf.org/html/draft-ietf-simple-winfo-package-05) for this functionality to work.

getSubscribedAttributes

Not supported.

blockSubscription

Blocks the flow of presence information to a subscribed watcher by cancelling the subscription. The watcher is notified using "subscriptionEnded".

An SVC0221 service exception (Not a watcher) is thrown if the URI in the field watcher is not a watcher of the presentity.

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.
- PRESENCE-000001 Failed to use the default duration for a notification.
- PRESENCE-000002 Failed to use the default value for count for a notification.
- PRESENCE-000003 The application has no SIP-URI mapping configured.
- PRESENCE-000004 Internal error. Failed to put data in WorkContext.
- SIP-000001 Could not find remote ejb home.
- SIP-000002 Could not create the ejb.
- SIP-000003 Could not access remote ejb.
- SIP-000004 Could not create SIP session.
- SIP-000005 Failed to send SIP message.
- SIP-000006 Internal SIP stack error.
- 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 8–1 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 8–2 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 8–3 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 8–4 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 8–5 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 8–6 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());
```

9

Parlay X 3.0 Interfaces

This chapter describes the supported Parlay X 3.0 interfaces and contains information that is specific for Oracle Communications Services Gatekeeper, and not found in the specifications. For detailed descriptions of the interfaces, methods, and parameters, refer to the specifications.

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

Interaction between Audio Call, Third Party Call, and Call Notification

The Parlay X 3.0 Part 2: Third Party Call, Parlay X 3.0 Part 3: Call Notification, and Parlay X 3.0 Part 11: Audio call interfaces interact with Services Gatekeeper Parlay compliant plug-ins. Together you use the interface/plug-in combinations to implement applications that use a combination of the services exposed by the interfaces. These services include:

- Call setup
- Call redirection and transfer
- Playing of announcements
- Collection of input from participants in the call using Dual-tone Multi-frequency (DTMF)

A call can have several participants. The call as a whole is represented by a `callSessionIdentifier`, and each participant is identified by their URI (the phone number, with scheme tel:) can be added to the call.

> **Note:** When the call is initiated from an application, the `callSessionIdentifier` is returned from Services Gatekeeper when the call session is established. When the call is initiated from the network, the `callSessionIdentifier` is provided by Services Gatekeeper in the requests that report the event.

You manage application-initiated call setup, tear-down, and transfer using the Parlay X 3.0 Part 2: Third Party Call interfaces.

You subscribe for notification on network-initiated calls, and take action, depending on the events, using the Parlay X 3.0 Part 3: Call Notification interfaces.

You play announcements and initiate collecting input from call participants using the Parlay X 3.0 Part 11: Audio call interfaces. Results from the collection of input are reported using Parlay X 3.0 Part 3: Call Notification.

Parlay X 3.0 Part 11: Audio Call interfaces must be used together with either Parlay X 3.0 Part 2: Third Party Call or Parlay X 3.0 Part 3: Call Notification since Audio Call does not have any operations to establish a call.

Parlay X 3.0 Part 2: Third Party Call

The Third Party Call communication service interfaces comply with ETSI ES 202 504-2 v0.0.5 (2007-06) Open Service Access (OSA); Parlay X Web Services; Part 2: Third Party Call (Parlay X 3).

Interface: ThirdPartyCall

The **ThirdPartyCall** interface endpoint is:

`http://`*`host`*`:`*`port`*`/parlayx30/third_party_call/ThirdPartyCall`.

where values for *host* and *port* depend on your Services Gatekeeper implementation.

makeCallSession

Sets up a call between two parties.

- MediaInfo must be set to NULL.
- ChangeMediaNotAllowed must be set to false.

Table 9–1 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	TPC-000001	Error reported from the telecom network.
SVC0001	TPC-000002	Error reported from the telecom network.
SVC0001	OSA-000001	Error reported from the telecom network.
SVC0001	OSA-000002	Error reported from the telecom network.
SVC0001	OSA-000003	Error reported from the telecom network.
SVC0001	OSA-000004	Error reported from the telecom network.
SVC0001	OSA-000006	Error reported from the telecom network.
SVC0001	OSA-000007	Error reported from the telecom network.
SVC0001	OSA-000008	Error reported from the telecom network.
SVC0001	OS-A000009	Error reported from the telecom network.
SVC0001	OS-A000010	Error reported from the telecom network.
SVC0001	OS-A000011	Error reported from the telecom network.
SVC0001	OSA-000012	Error reported from the telecom network.
SVC0001	OS-A000013	Error reported from the telecom network.
SVC0001	OSA-000014	Error reported from the telecom network.
SVC0001	OSA-000015	Error reported from the telecom network.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper.
SVC0001	PLG-000004	General plug-in routing error.

Table 9–1 (Cont.) exceptions and error codes

Exception	Error Code	Explanation
SVC0002	N/A	Incorrect input parameter.
POL0001	N/A	Faults related to policies associated with the service, including service level agreements.
POL0008	N/A	Charging not supported.
POL0011	N/A	Media type not supported.

addCallParticipant

Adds a participant to an existing call session. The call session may have been established using **makeCallSession** or any of the methods in "Interface: CallDirection" and "Interface: CallNotification".

`mediaInfo` must be set to `NULL`.

Table 9–2 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	TPC-000002	Error reported from the telecom network.
SVC0001	OSA-000001	Error reported from the telecom network.
SVC0001	OSA-000002	Error reported from the telecom network.
SVC0001	OSA-000003	Error reported from the telecom network.
SVC0001	OSA-000004	Error reported from the telecom network.
SVC0001	OSA-000006	Error reported from the telecom network.
SVC0001	OSA-000007	Error reported from the telecom network.
SVC0001	OSA-000008	Error reported from the telecom network.
SVC0001	OSA-000009	Error reported from the telecom network.
SVC0001	OSA-000010	Error reported from the telecom network.
SVC0001	OSA-000011	Error reported from the telecom network.
SVC0001	OSA-000012	Error reported from the telecom network.
SVC0001	OSA-000014	Error reported from the telecom network.
SVC0001	OSA-000015	Error reported from the telecom network.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	Call session identifier is null; or error reported from the telecom network.
SVC0261	N/A	The call is already terminated.
POL0001	N/A	The application is not the owner of the call; or error reported form the network.
POL0011	N/A	Media type not supported.
POL0240	N/A	Too many participants.

transferCallParticipant

Transfers a participant from one call session to another call session.

Table 9–3 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	TPC-000002	No call leg identifier reported from network; or error reported from the telecom network.
SVC0001	TPC-000003	The destination call session reference; or call leg reference is null; or an internal error has occurred.
SVC0001	TPC-000007	The participant does not belong to this call.
SVC0001	OSA-000001	Error reported from the telecom network.
SVC0001	OSA-000002	Error reported from the telecom network.
SVC0001	OSA-000003	Error reported from the telecom network.
SVC0001	OSA-000004	Error reported from the telecom network.
SVC0001	OSA-000006	Error reported from the telecom network.
SVC0001	OSA-000007	Error reported from the telecom network.
SVC0001	OSA-000008	Error reported from the telecom network.
SVC0001	OSA-000009	Error reported from the telecom network.
SVC0001	OSA-000010	Error reported from the telecom network.
SVC0001	OSA-000011	Error reported from the telecom network.
SVC0001	OSA-000012	Error reported from the telecom network.
SVC0001	OSA-000014	Error reported from the telecom network.
SVC0001	OSA-000015	Error reported from the telecom network.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	Error reported from the telecom network; or source call session identifier is incorrect; or destination call session identifier is incorrect; or participant part is incorrect.
SVC0261	N/A	The call is already terminated.
POL0001	N/A	TPC100001
POL0240	N/A	Too many participants.

getCallParticipantInformation

Gets information about a certain participant in a call session.

Table 9–4 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	TPC-000007	The participant does not belong to this call.
SVC0001	OSA-000001	Error reported from the telecom network.
SVC0001	OSA-000002	Error reported from the telecom network.
SVC0001	OSA-000003	Error reported from the telecom network.
SVC0001	OSA-000004	Error reported from the telecom network.
SVC0001	OSA-000006	Error reported from the telecom network.
SVC0001	OSA-000007	Error reported from the telecom network.

Table 9–4 (Cont.) exceptions and error codes

Exception	Error Code	Explanation
SVC0001	OSA-000008	Error reported from the telecom network.
SVC0001	OSA-000009	Error reported from the telecom network.
SVC0001	OSA-000010	Error reported from the telecom network.
SVC0001	OSA-000011	Error reported from the telecom network.
SVC0001	OSA-000012	Error reported from the telecom network.
SVC0001	OSA-000014	Error reported from the telecom network.
SVC0001	OSA-000015	Error reported from the telecom network.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	Call session identifier is incorrect.
SVC0261	N/A	The call is already terminated.
POL0001	TPC-100001	The application is not the owner of the call session.

getCallSessionInformation

Displays information about a call session.

Table 9–5 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	TPC-000007	The participant does not belong to this call.
SVC0001	OSA-000001	Error reported from the telecom network.
SVC0001	OSA-000002	Error reported from the telecom network.
SVC0001	OSA-000003	Error reported from the telecom network.
SVC0001	OSA-000004	Error reported from the telecom network.
SVC0001	OSA-000006	Error reported from the telecom network.
SVC0001	OSA-000007	Error reported from the telecom network.
SVC0001	OSA-000008	Error reported from the telecom network.
SVC0001	OSA-000009	Error reported from the telecom network.
SVC0001	OSA-000010	Error reported from the telecom network.
SVC0001	OSA-000011	Error reported from the telecom network.
SVC0001	OSA-000012	Error reported from the telecom network.
SVC0001	OSA-000014	Error reported from the telecom network.
SVC0001	OSA-000015	Error reported from the telecom network.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	Network error; or call session identifier is incorrect.
SVC0261	N/A	The call is already terminated.

Table 9–5 (Cont.) exceptions and error codes

Exception	Error Code	Explanation
POL0001	TPC-100001	The application is not the owner of the call session.

deleteCallParticipant

Deletes a participant from a call session.

Table 9–6 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	TPC-000006	There are no participants in this call.
SVC0001	OSA-000001	Error reported from the telecom network.
SVC0001	OSA-000002	Error reported from the telecom network.
SVC0001	OSA-000003	Error reported from the telecom network.
SVC0001	OSA-000004	Error reported from the telecom network.
SVC0001	OSA-000006	Error reported from the telecom network.
SVC0001	OSA-000007	Error reported from the telecom network.
SVC0001	OSA-000008	Error reported from the telecom network.
SVC0001	OSA-000009	Error reported from the telecom network.
SVC0001	OSA-000010	Error reported from the telecom network.
SVC0001	OSA-000011	Error reported from the telecom network.
SVC0001	OSA-000012	Error reported from the telecom network.
SVC0001	OSA-000014	Error reported from the telecom network.
SVC0001	OSA-000015	Error reported from the telecom network.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	Network error; or call session identifier is incorrect; or call participant identifier is incorrect.
SVC0261	N/A	The call is already terminated.
POL0001	TPC-100001	The application is not the owner of the call session.

endCallSession

Ends a call session.

Table 9–7 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	TPC-000005	The call reference in Services Gatekeeper storage is incorrect.
SVC0001	TPC-000006	There are no participants in the call session.
SVC0001	OSA-000001	Error reported from the telecom network.

Table 9–7 (Cont.) exceptions and error codes

Exception	Error Code	Explanation
SVC0001	OSA-000002	Error reported from the telecom network.
SVC0001	OSA-000003	Error reported from the telecom network.
SVC0001	OSA-000004	Error reported from the telecom network.
SVC0001	OSA-000006	Error reported from the telecom network.
SVC0001	OSA-000007	Error reported from the telecom network.
SVC0001	OSA-000008	Error reported from the telecom network.
SVC0001	OSA-000009	Error reported from the telecom network.
SVC0001	OSA-000010	Error reported from the telecom network.
SVC0001	OSA-000011	Error reported from the telecom network.
SVC0001	OSA-000012	Error reported from the telecom network.
SVC0001	OSA-000014	Error reported from the telecom network.
SVC0001	OSA-000015	Error reported from the telecom network.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	N/A	Error reported from the telecom network; or call session identifier is incorrect; or call participant identifier is incorrect.
SVC0261	N/A	The call is already terminated.
POL0001	TPC-100001	The application is not the owner of the call session.

Parlay X 3.0 Part 3: Call Notification

The Call Notification communication service interfaces comply with ETSI ES 202 504-3 v0.0.3 (2007-06) Open Service Access (OSA); Parlay X Web Services; Part 3: Call Notification (Parlay X 3).

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/parlayx30/call_notification/wsdls/parlayx_call_direction_service_3_2.wsdl`

`http://host:port/parlayx30/call_notification/wsdls/parlayx_call_direction_interface_3_2.wsdl`

`http://host:port/parlayx30/call_notification/wsdls/parlayx_common_types_3_1.xsd`

`http://host:port/parlayx30/call_notification/wsdls/parlayx_common_faults_3_0.wsdl`

`http://host:port/parlayx30/call_notification/wsdls/parlayx_call_notification_types_3_1.xsd`

where values for *host* and *port* depend on the Services Gatekeeper deployment.

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, prior to call setup.

Interface: CallNotification

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/parlayx30/call_notification/wsdls/parlayx_call_notification_interface_3_2.wsdl`

`http://host:port/parlayx30/call_notification/wsdls/parlayx_call_notification_service_3_2.wsdl`

`http://host:port/parlayx30/call_notification/wsdls/parlayx_common_types_3_1.xsd`

`http://host:port/parlayx30/call_notification/wsdls/parlayx_common_faults_3_0.wsdl`

`http://host:port/parlayx30/call_notification/wsdls/parlayx_call_notification_types_3_1.xsd`

where values for host and port depend on the Services Gatekeeper deployment.

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, prior to call setup.

notifyAnswer

Services Gatekeeper calls this method, which is implemented by an application, when the called party answered.

notifyPlayAndCollectEvent

Services Gatekeeper calls this method, which is implemented by an application, to provide the result of a media interaction of type play and collect information.

notifyPlayAndRecordEvent

Services Gatekeeper calls this method, which is implemented by an application, to provide the result of a media interaction of type play and record information.

Interface: CallNotificationManager

The **CallNotificationManager** interface endpoint is:

`http://host:port/parlayx30/call_notification/CallNotificationManager`

where values for *host* and *port* depend on your Services Gatekeeper implementation.

startCallNotification

Starts a subscription for call notifications.

Table 9–8 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	OSA-000002	`P_INVALID_INTERFACE_TYPE` thrown by OSA Gateway. Check the interface name
SVC0001	OSA-000003	`P_INVALID_EVENT_TYPE` thrown by OSA Gateway. Check the event type.
SVC0001	OSA-000006	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_RESOURCE_UNAVAILABLE` (13). Check OSA Gateway status.
SVC0001	OSA-000007	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_TASK_REFUSED`(14). Check OSA Gateway status.
SVC0001	OSA-000008	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_TASK_CANCELLED`(15). Check invocation parameters of `createNotification`()
SVC0001	OSA-000009	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_NO_CALLBACK_ADDRESS_SET` (17). Check OSA Gateway.

Table 9–8 (Cont.) exceptions and error codes

Exception	Error Code	Explanation
SVC0001	OSA-000010	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_METHOD_NOT_SUPPORTED` (22). Check OSA Gateway.
SVC0001	OSA-000011	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_INVALID_STATE` (744). Check OSA Gateway.
SVC0001	OSA-000015	`P_INVALID_CRITERIA` thrown by OSA Gateway. Check the criteria.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	*reference*	Parameter reference is null.
SVC0002	*correlator*	Parameter correlator is null.
SVC0002	*endPoint*	Parameter endPoint is null or empty string.
SVC0002	*addresses*	Parameter reference is null.
SVC0005	*correlator*, *reference*	Correlator`%1` specified in message part`%2` is a duplicate.
POL0001	*Service contract not found.*	No Service Level Agreement found for the service provider or application associated with the request.

startPlayAndCollectNotification

Starts a subscription for notifications on media interactions of type `play` and `collect`.

Table 9–9 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	CN-000001	Parlay call session does not exist.
SVC0001	CN-000002	Parlay call session has terminated.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	*reference*	Parameter reference is null.
SVC0002	*correlator*	Parameter correlator is null.
SVC0002	*endPoint*	Parameter endPoint is null or empty string.
SVC0002	*callSessionIdentifier*	Parameter `callSessionIdentifier` is null.
SVC0005	*correlator*, *reference*	Correlator `%1` specified in message part `%2` is a duplicate.
SVC0005	*callSessionId:value*	`startPlayAndCollectNotification` was called earlier on the same `callSessionIdentifier` or `startCallDirection` was called earlier with an address that is identical to the address represented by the call session.
POL0001	*Service contract not found.*	No Service Level Agreement found for the service provider or application associated with the request.

startPlayAndRecordNotification

Not supported.

stopCallNotification

Stops a subscription for call notifications.

Table 9–10 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	CN-000003	The requester did not start the `notification` and is not the owner.
SVC0001	CN-000004	The parameter correlator does not exist.
SVC0001	OSA-000006	`TpCommonExceptions` thrown by OSA GW. Exception type is `P_RESOURCE_UNAVAILABLE` (13). Check OSA Gateway status.
SVC0001	OSA-000007	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_TASK_REFUSED` (14). Check OSA Gateway status.
SVC0001	OSA-000008	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_TASK_CANCELLED` (15). Check OSA Gateway status.
SVC0001	OSA-000009	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_NO_CALLBACK_ADDRESS_SET` (17). Check invocation parameters of `destroyNotification()`.
SVC0001	OSA-000010	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_METHOD_NOT_SUPPORTED` (22). Check OSA Gateway.
SVC0001	OSA-000011	`TpCommonExceptions` thrown by OSA Gateway. Exception type is `P_INVALID_STATE` (744). Check OSA Gateway.
SVC0001	*correlator*	The parameter correlator does not exist.
SVC0001	PLG-000004	General plug-in routing error.
POL0001	N/A	No Service Level Agreement found for the service provider or application associated with the request.

stopMediaInteractionNotification

Stops a subscription for `notifications` on media interactions.

Table 9–11 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	CN-000004	The parameter correlator does not exist.
SVC0001	CN-000003	The requester did not start the `notification` and is not the owner.

Table 9–11 (Cont.) exceptions and error codes

Exception	Error Code	Explanation
SVC0001	*correlator*	The parameter correlator does not exist.
SVC0001	PLG-000004	General plug-in routing error.
POL0001	*Service contract not found.*	No Service Level Agreement found for the service provider or application associated with the request.

Interface: CallDirectionManager

The CallDirectionManager interface endpoint is:

`http://host:port/parlayx30/call_notification/CallDirectionManager`

where values for *host* and *port* depend on your Services Gatekeeper implementation.

StartCallDirectionNotification

Starts a subscription for call direction `notifications`.

Table 9–12 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	OSA-000002	`P_INVALID_INTERFACE_TYPE` thrown by OSA Gateway. Check the interface name.
SVC0001	OSA-000003	`P_INVALID_EVENT_TYPE` thrown by OSA Gateway Check the event type.
SVC0001	OSA-000015	`P_INVALID_CRITERIA` thrown by OSA Gateway. Check the criteria.
SVC0001	OSA-000006	TpCommonExceptions thrown by OSA GW. Exception type is `P_RESOURCE_UNAVAILABLE` (13).Check OSA Gateway status.
SVC0001	OSA-000007	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_TASK_REFUSED`(14). Check OSA Gateway status.
SVC0001	OSA-000008	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_TASK_CANCELLED` (15). Check OSA Gateway status.
SVC0001	OSA-000009	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_NO_CALLBACK_ADDRESS_SET` (17). Check invocation parameters of `createNotification()`.
SVC0001	OSA-000010	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_METHOD_NOT_SUPPORTED` (22). Check OSA Gateway.
SVC0001	OSA-000011	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_INVALID_STATE` (744). Check OSA Gateway.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	*reference*	Parameter reference is null.

Table 9–12 (Cont.) exceptions and error codes

Exception	Error Code	Explanation
SVC0002	*correlator*	Parameter correlator is null.
SVC0002	*endPoint*	Parameter endPoint is null or empty string.
SVC0002	*addresses*	Parameter reference is null.
SVC0005	*correlator value, reference*	Correlator `%1` specified in message part `%2` is a duplicate.
POL0001	*Service contract not found.*	No Service Level Agreement found for the service provider or application associated with the request.

StopCallDirectionNotification

Stops a subscription for call direction `notification`s.

Table 9–13 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	CN-000003	The requester did not start the `notification` and is not the owner.
SVC0001	CN-000004	The parameter correlator does not exist.
SVC0001	OSA-000006	TpCommonExceptions thrown by OSA GW. Exception type is `P_RESOURCE_UNAVAILABLE` (13). Check OSA Gateway status.
SVC0001	OSA-000007	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_TASK_REFUSED` (14). Check OSA Gateway status.
SVC0001	OSA-000008	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_TASK_CANCELLED` (15). Check OSA Gateway status.
SVC0001	OSA-000009	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_NO_CALLBACK_ADDRESS_SET` (17). Check invocation parameters of `destroyNotification()`.
SVC0001	OSA-000010	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_METHOD_NOT_SUPPORTED` (22). Check OSA Gateway.
SVC0001	OSA-000011	TpCommonExceptions thrown by OSA Gateway. Exception type is `P_INVALID_STATE` (744). Check OSA Gateway.
SVC0001	*correlator*	The parameter correlator does not exist.
SVC0001	PLG-000004	General plug-in routing error.
POL0001	N/A	No Service Level Agreement found for the service provider or application associated with the request.

Parlay X 3.0 Part 6: Payment

The Payment communication service interfaces comply with 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* depend on your Services Gatekeeper implementation.

chargeAmount

Charges the account indicated by the end user identifier.

Table 9–14 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, 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 9–15 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)

chargeSplitAmount

Charges multiple end user accounts concurrently.

Table 9–16 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.

Table 9–16 (Cont.) Exceptions and Error Codes

Exception	Error Code	Explanation
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.

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* depend on your Services Gatekeeper implementation.

chargeVolume

Charges the specified volume to the end user account.

Table 9–17 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 9–18 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 9–19 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 9–20 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 9–21 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* depend on your Services Gatekeeper implementation.

reserveAmount

Reserves a charge for an account indicated by the end user identifier.

Table 9–22 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 9–23 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)

chargeReservation

Charges to a reservation indicated by the reservation ID.

Table 9–24 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 9–24 (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)

releaseReservation

Returns funds left in a reservation to the account from which this reservation was made.

Table 9–25 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 9–25 (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* depend on your Services Gatekeeper implementation.

reserveVolume

Reserves a volume amount for the end user account.

Table 9–26 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 9–27 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 11: Audio Call

The Audio Call communication service interfaces comply with ETSI ES 202 504-11 v0.0.3 (2007-06), Open Service Access (OSA); Parlay X Web Services; Part 11: Audio Call (Parlay X 3).

Interface: PlayMedia

The **PlayMedia** interface endpoint is:

`http://`*`host`*`:`*`port`*`/parlayx30/audio_call/AudioCallPlayMedia`

where values for *host* and *port* depend on your Services Gatekeeper implementation.

playTextMessage

Not supported.

playAudioMessage

Plays a message to the given destination address. The message is given as a URL to an audio file. The file must be reachable by the underlying telecom network node and the audio format must be supported by the telecom network.

Table 9–28 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	AC-100001	Call session has expired.
SVC0001	AC-100003	Call state is incorrect.
SVC0001	AC-100004	Call participant is not connected.
SVC0001	AC-100005	Not all call participants are available. The participant may already be in `playing` or `collecting` mode.
SVC0001	AC-100006	Could not find call. Call session may be empty.
SVC0001	OSA-000001	`P_INVALID_NETWORK_STATE` exception received from OSA Gateway.
SVC0001	OSA-000011	`EC_OSA_P_INVALID_STATE` exception received from OSA Gateway.
SVC0001	OSA-000012	`TP_COMMON_EXCEPTIONS` exception received from OSA Gateway.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	*callSessionIdentifier*	Invalid input value for message part `%1`.
SVC0002	*callParticipants*	Invalid input value for message part `%1`.
SVC0002	N/A	`P_INVALID_SESSION_ID` exception received from OSA Gateway.
SVC0261	N/A	Call has already been terminated.

Table 9–28 (Cont.) exceptions and error codes

Exception	Error Code	Explanation
POL0001	*Service contract not found.*	No Service Level Agreement found.
POL0001	AC-100002	Application is not the owner of the call.
POL0008	N/A	Charging not supported.

playVoiceXmlMessage

Not supported.

playVideoMessage

Not supported.

getMessageStatus

Gets the status of a message; that is, if the message is currently being played, if it is has finished playing.

Table 9–29 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	*correlator*	Invalid input value for message part `%1`.
POL0001	*Service contract not found.*	No Service Level Agreement found.
POL0001	AC-100002	Application is not the owner of the call.

endMessage

Cancels or stops the playing of the message.

Table 9–30 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to fetch information from Services Gatekeeper. Contact support.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	correlator	Invalid input value for message part `%1`.
POL0001	*Service contract not found.*	No Service Level Agreement found.
POL0001	AC-100002	Application is not the owner of the call.

Interface: CaptureMedia

The CaptureMedia interface endpoint is:

`http://`*`host`*`:`*`port`*`/parlayx30/audio_call/AudioCallCaptureMedia`

where values for host and port depend on the Services Gatekeeper deployment.

startPlayAndCollectInteraction

Starts a media interaction with one or all participants in a call session. Plays a media file and collects digits from one or all call participants. The results of the interaction is notified using **notifyPlayAndCollectEvent** in the Parlay X 3.0 Part 3: Call Notification set of interfaces.

Table 9–31 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
N/A	AC-100001	Call session has expired.
N/A	AC-100003	Call state is incorrect.
N/A	AC-100004	Participant is not connected.
N/A	AC-100005	Not all participants are available. Possibly in already in playing or collecting mode.
SVC0001	AC-100006	Could not find call. Call session may be empty.
SVC0001	OSA-000001	`P_INVALID_NETWORK_STATE` exception received from OSA Gateway.
SVC0001	OSA-000011	`EC_OSA_P_INVALID_STATE` exception received from OSA Gateway.
SVC0001	OSA-000012	`TP_COMMON_EXCEPTIONS` exception received from OSA Gateway.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	*callSessionIdentifier*	Invalid input value for message part `%1`.
SVC0002	*callParticipants*	Invalid input value for message part `%1`.
SVC0002	*playFileLocation*	Invalid input value for message part `%1`.
SVC0002	N/A	`P_INVALID_SESSION_ID` exception received from OSA Gateway.
SVC0261	N/A	Call has already been terminated.
POL0001	*Service contract not found.*	No Service Level Agreement found.
POL0001	AC-100002	Application is not the owner of the call.
POL0001	AC-100007	The value of `maxDigits` is too big.
POL0001	AC-100008	The value of `minDigits` is too small.

startPlayAndRecordInteraction

Not supported.

stopMediaInteraction

Explicitly stops an ongoing media interaction session.

Table 9–32 exceptions and error codes

Exception	Error Code	Explanation
SVC0001	OSA-000001	`P_INVALID_NETWORK_STATE` exception received from OSA Gateway.
SVC0001	OSA-000011	`EC_OSA_P_INVALID_STATE` exception received from OSA Gateway.
SVC0001	OSA-000012	`TP_COMMON_EXCEPTIONS` exception received from OSA Gateway.
SVC0001	WNG-000002	Failed to store information in Services Gatekeeper. Contact support.
SVC0001	PLG-000004	General plug-in routing error.
SVC0002	*mediaIdentifier*	Invalid input value for message part `%1`.
SVC0002	N/A	`P_INVALID_SESSION_ID` exception received from OSA Gateway.
POL0001	*Service contract not found.*	No Service Level Agreement found.
POL0001	AC-100002	Application is not the owner of the call.

Interface: Multimedia

This section describes the Multimedia interface.

addMediaForParticipants

Not supported.

deleteMediaForParticipants

Not supported.

getMediaForParticipant

Not supported.

getMediaForCall

Not supported.

Parlay X 3.0 Part 13: Address List Management

The Address List Management communication service interfaces comply with 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* depend on your Services Gatekeeper implementation.

createGroup

Creates an Address List Management group.

Table 9–33 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* depend on your Services Gatekeeper implementation.

addMember

Adds a single member to an Address List Management group.

addMembers

Adds multiple members to an Address List Management group.

Table 9–34 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* depend on your Services Gatekeeper implementation.

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 comply with 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* depend on your Services Gatekeeper implementation.

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 `notifications` are 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.

10

Native Interfaces

This chapter describes the supported native interfaces and contains information that is specific for Oracle Communications Services Gatekeeper, and not found in the specifications. For detailed descriptions of the interfaces, methods and parameters, refer to the specifications.

MM7

The MM7 specification can be found at:

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 depend on the Services Gatekeeper deployment.

Note: The MM7 facade uses HTTP basic authentication, username/password. The username is the application instance ID.

MM7_submit

Sends an application-initiated multimedia message

Table 10–1 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. Servcies Gatekeeper supports it as a tunneled parameter named **ussd_service_operation**. For details about these parameters, see the "Tunneled Parameters for Native SMPP" section of the "Native SMPP Communication Service" chapter in *Communication Service Guide*, another document in this set.

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 to the addApplicationSpecificSettings operation. See System Administrator's Guide for information about this operation.

Services Gatekeeper can be configured to terminate a session if:

- The session is inactive. See the **InactivityTimerValue** in *System Administrator's Guide*.
- The application takes too long time to respond to a request. See the **RequestTimerValue** in *System Administrator's Guide*.

Error Handling

All errors are reported in the command_status field of a response PDU.

Table 10–2 lists the error codes that are specific for Services Gatekeeper. Errors from the SMSC are transparently forwarded to the application.

Table 10–2 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.
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 System Administrator's 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 System Administrator's 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 via 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 via 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 System Administrator's 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 *System Administrator's 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 native 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: North 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: South 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)
```