Oracle® Communications Services Gatekeeper

Communication Service Guide

Release 5.0

E21362-01

April 2011

Oracle Communications Services Gatekeeper Communication Service Guide, Release 5.0

E21362-01

Contents

Preface

This book is a detailed reference for the communications services used in Oracle Communications Services Gatekeeper.

Audience

This document is for system administrators who install and maintain Oracle Communications Services Gatekeeper, as well as managers, support engineers, application developers, sales, and marketing.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

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

- *Accounts and SLAs Guide*
- *Alarm Handling Guide*
- *Application Developer's Guide*
- *Concepts Guide*
- *Installation Guide*
- *Licensing Guide*
- *Partner Relationship Management Guide*
- *Platform Development Studio Developer's Guide*
- *Platform Test Environment Guide*
- *RESTful Application Developer's Guide*
- *SDK User's Guide*
- *Statement of Compliance*
- *System Administrator's Guide*
- *System Backup and Restore Guide*

1 About Communication Services

This chapter presents a high level introduction to Oracle Communications Services Gatekeeper communication services.

Introduction

All application service request data flows through Services Gatekeeper communication services. A communication service consists of a service type, such as Multimedia Messaging, Terminal Location, and so on, an application-facing interface (also called a "north" interface), and a network-facing interface (also called "south" interface).

How They Work

Communication services are separated into two functional layers: the service facade and the service enabler. The service facade contains the application-facing interfaces and manages interactions with applications. The service enabler contains the mechanisms necessary for communicating with the underlying network nodes.

Application-initiated requests (also called mobile terminated, or MT requests) enter through the service facade. A facade comprises a set of application-facing interfaces of a particular type. Services Gatekeeper supplies facades for traditional SOAP Web Services interfaces, RESTful interfaces, and, in three cases (MM7, SMPP, and UCP) native telephony interfaces. There is also a facade specifically designed to work with the Oracle Service Bus, for SOA-style installations.

After the requests have been processed by the service facade, they are sent to the service enabler by using Remote Method Invocation (RMI). The service enabler layer manages service authorization and policy enforcement, charging, and traffic throttling and shaping. The enabler translates the request into a form appropriate for the underlying network node.

Although the operator may choose instead to run in a sessionless mode, by default Services Gatekeeper requires that applications (except those using native telephony interfaces) acquire a Services Gatekeeper session before sending request traffic. Applications do this using the Session Manager interface appropriate for their facade type. The Session Manager returns a session ID, which the application adds to the header of all its requests. Services Gatekeeper can use the session ID to keep track of all the traffic that an application sends for the duration of the session. Sessions allow correlation among sequences of operations. They are not used for authentication

Network-triggered (also called mobile originated, or MO) traffic enables applications to receive data from the telecom network. To do so, the application must first send a request to Services Gatekeeper, or have the operator perform the equivalent task using operation, administration, and maintenance (OAM) operations, to register a description of the types of data it is interested in – delivery notifications, incoming messages, etc. – and any criteria that the data must be meet to be acceptable. For example, an application might specify that it is only interested in receiving incoming SMS messages that are addressed to the **12345** short code and that begin with the string **blue**.

Typical Application-Initiated Traffic Flow

The following steps describe the application-initiated traffic flow. Steps 1-3 are optional.

1. An application establishes a session using the Session Management Web Service in the facade layer.
2. A session is established, and the session ID is returned to the application. After the application has been established, it may access multiple communication services across the cluster transparently.
3. The session is valid until the application terminates it or an operator-established time period has elapsed.
4. A request for a particular operation, usually transported over Secure Sockets Layer (SSL), enters at the application-facing interface in the facade layer, either directly from the application, or, if the particular installation uses an Oracle Service Bus, from the Oracle Service Bus. The application-facing interface is implemented as a SOAP-based Web Service or a RESTful Web Service. Requests using the RESTful requests are authenticated with HTTP basic authentication using a user name and password. SOAP-based Web Service requests are authenticated using WebLogic Server WS-Security, which supports plain text or digest passwords, X.509 certificates, or SAML tokens.

   All requests are authenticated in this manner, whether or not the application uses the session mode.

   In addition, SOAP-based requests may be further secured through encryption using the W3C's standard XML encryption and through digital signatures using the W3C XML digital signature standard. The particular security requirements of the installation are specified in the WS-Policy section of the operator-published WSDL file.

   For information about W3C's standard XML encryption, see

   `http://www.w3.org/TR/xmlenc-core/`

   For information about W3C XML digital signature standard, see

   `http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/`

   It is possible to use the appropriate standard Parlay X 2.1 or 3.0 WSDL to create SOAP-based requests, but the developer would then be required to ascertain the appropriate security type from the operator and insert the information manually.
5. The request is serialized and passed on to the service enabler over RMI.

   From this point on, requests that enter the communication service using the SOAP Service Facade and those using the RESTful Service Facade use the same service enablers. SLA construction, CDRs, EDRs, alarms, and so forth are same for the SOAP-based requests as they are for the RESTful requests of the same type.

   The entrance point for the service enabler marks the beginning of the application-initiated transaction.

6. The request is sent to the Plug-in Manager.
7. The Plug-in Manager invokes the Interceptor Stack to evaluate the request. The Interceptor Stack is a flexible set of chained evaluation steps that:
   - Validates the request
   - Enforces a range of policy decisions based on SLAs and possibly additional rules
   - Performs any necessary data manipulation
   - Routes the request to an appropriate protocol translation module (a network plug-in): Routing can be done on a wide variety of parameters.

   If a request fails because of an unavailable module, an interceptor retries the request using one of the remaining eligible modules.
8. The request is sent to the network plug-in to be translated into the protocol suitable for the underlying network node. All state information required by the underlying network node is stored within the network plug-in.
9. The request is passed to the network.
10. When the network node acknowledges the request, charging data about the completed request is recorded.
11. The transaction commits.

Typical Network-Triggered Traffic Flow

To receive network-triggered traffic, an application must indicate to Services Gatekeeper that it is interested in receiving traffic from the network. It does this by registering for (or subscribing to) notifications, either by sending a request to Services Gatekeeper or by having the operator set up the notification using OAM operations.

For example, the application could send Services Gatekeeper a request to begin receiving SMS messages from the network, indicating that it is only interested in messages that are sent to the address **12345** and that begin with the string **blue**. SOAP-based requests indicate the URL of the Web Service that the application has implemented to receive these notifications back from Services Gatekeeper.RESTful requests indicate the channel to which the notifications should be published.

The registration for notifications is stored in the appropriate network plug-in, which in most cases passes it on to the underlying network node. In certain cases the Services Gatekeeper operator must do this manually. When a matching SMS message reaches the plug-in from the network, the plug-in sends the message to the Plug-in Manager, which invokes the Interceptor Stack for evaluation. Then, using RMI, the final interceptor passes the notification, along with the appropriate location from the registration, to the facade layer, which sends it on either to the application, the channel, or to the Oracle Service Bus.

Installations that include multiple facade layers (for example, both RESTful and SOA) can be set up to use the same service enabler layer. Special configuration is required in such installations to route network-triggered traffic to the appropriate facade layer. See "Managing and Configuring the Tier Routing Manager" in *System Administrator's Guide* for more information.

Common Features

The following functionality is common to all communication services:

- Service level agreements related to policy enforcement
- Service level agreements related to network protection
- Traffic security
- Events, alarms, and charging
- Statistics and transaction units

For information about service level agreements, see *Accounts and SLAs Guide*.

For information about traffic security for SOAP-based interfaces to the communication services, see *Oracle Fusion Middleware Securing Oracle WebLogic Server* at:

`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13707/toc.htm`

RESTFul Web Services to the communications services use HTTP basic authentication with a user name and password. SSL is required. For information about basic HTTP authentication, see *HTTP Authentication: Basic and Digest Access Authentication* at:

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

For general information about events, alarms, and charging, see Appendix A, "Events, Alarms, and Charging."

For information about statistics, see the "Statistics" sections in the individual chapters in this guide.

Connectivity to SIP Network Infrastructure

For communication services that access Session Initiation Protocol (SIP) networks, Services Gatekeeper connects applications to SIP-based functionality by using Oracle Converged Application Server. Converged Application Server is collocated with Services Gatekeeper in the network tier.

2

Parlay X 2.1 Audio Call/SIP

This chapter describes the Parlay X 2.1 Audio Call/Session Initiation Protocol (SIP) communication service in detail.

Overview of the Parlay X 2.1 Audio Call / SIP Communication Service

The Parlay X 2.1 Audio Call/SIP communication service exposes the Parlay X 2.1 Audio Call 2.1 application interfaces.

The communication service connects to a SIP-IMS network using Oracle Converged Application Server. Converged Application Server is collocated with Services Gatekeeper in the network tier.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using this communication service, an application can:

- Set up a Parlay X 2.1 call between a terminal device and a media server to play an audio file (such as WAV).
- Get the status for a Parlay X 2.1 Audio call (played, playing, pending, or error).
- Explicitly end any of these audio calls.

The Audio Call communication service can be used by applications to start an audio call using the Parlay X 2.1 Part 11 protocol.

The audio message content to be played must be defined in an audio fileformat stored at a URL available to the network.. Services Gatekeeper does not actually render the message. This is the responsibility of equipment that must be present on the target telecom network.

Audio Call/SIP Plug-in Application Requests

The Audio Call/SIP plug-in uses these requests that applications send to play or manage audio calls:

- **PlayAudioMessage**: The application sends the URI of an audio file and the address of the terminal to the network. This request then returns a call correlator to the application and plays the audio file to the terminal.
- **EndMessage**: The application send in the correlator of a **PlayAudioMessage** request to end that call immediately.

- **GetMessageStatus**: The application sends in the correlator of a **PlayAudioMessage** request to get the status of that call. This method returns the status of the call (played, playing, pending, or error.

Audio Call/SIP Plug-in Call Flow

This is a sample Audio Call/SIP call flow:

1. The application sends a Parlay X 2.1 **PlayAudioCall** request to the terminal address with a SIP INVITE message.

   This message does not contain a Session Description Protocol (SDP).

2. The terminal returns a **SIP 200 OK** message to the application containing a valid SDP.
3. The application's media server controller then processes the SDP and sends an **ACK** message back to the terminal address.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 Audio Call/SIP communication service, see the discussion of Parlay X 2.1 Interfaces in *Application Developer's Guide*.

For information about the RESTful Audio Call interface, see the discussion of Audio Call in *RESTful Application Developer's Guide*.

The RESTful Service Short Messaging interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs, reading CDRs, and so on, they are the same.

Events and Statistics

The Parlay X 2.1 Audio Call/SIP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 2–1 lists the IDs of the EDRs created by the Parlay X 2.1 Audio Call/SIP communication service. This does not include EDRs created when exceptions are thrown.

Table 2–1 Event Types Generated by Parlay X 2.1 Audio Call/SIP

EDR ID	Method Called
1000	PlayTextMessage
1003	GetMessageStatus
1004	EndMessage

Charging Data Records

Audio Call/SIP-specific CDRs are generated under the following conditions:

- When **callConnected** is sent from the network to Services Gatekeeper, indicating that the audio message has connected to the terminal. The CDR number is 405001; the class is oracle.ocsg.plugin.audio_call.sip.south.call.AudioCallCdrContext.
- When **callReleased** is sent from the network to Services Gatekeeper, indicating that the audio message has completed playing. The CDR number is 405002; the class is oracle.ocsg.plugin.audio_call.sip.south.call.AudioCallCdrContext.

Statistics

Table 2–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 2–2 Methods and Transaction Types for Parlay X 2.1 Audio Call/SIP

Method	Transaction Type
PlayAudioMessage	TRANSACTION_TYPE_CALL_CONTROL_SERVICE_INITIATED

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing Parlay X 2.1 Audio Call / SIP

This section describes the properties and workflow for setting up the Parlay X 2.1 Audio Call/SIP plug-in instance.

The Parlay 2.1 Audio Call/SIP plug-in supports sending an audio fileto a single terminal.

The Audio Call/SIP plug-in is usable with high availability systems only if your media server supports clustering. See your media server documentation for details.

This plug-in service does not support multiple instantiation using the Plug-in Manager. You can create only one instance by using the Plug-in Manager. The plug-in instance is not automatically created when the plug-in service is started.

Properties for Parlay X 2.1 Audio Call/SIP

Table 2–3 lists the technical specifications for the communication service.

Table 2–3 Properties for Parlay X 2.1 Audio Call/SIP

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > plugin_instance_id

MBean	Domain=wlng_nt_audio_call_px21#5.0
	Name=wlng_nt
	InstanceName=Audio_Call_sip
	Type=Plugin_px21_audio_call_sip.AudioCallMBean

Network protocol plug-in service ID	Plugin_px21_audio_call_sip
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.

Table 2–3 (Cont.) Properties for Parlay X 2.1 Audio Call/SIP

Property	Description
Supported Address Schemes	tel, sip
Application-facing interfaces	com.bea.wlcp.wlng.px21.plugin.AudioCallPlugin
Service type	AudioCall
Exposes to the service communication layer a Java representation of:	Parlay X 2.1 Part 11: Audio Call
Interfaces with the network nodes using:	RFC 3261. `http://www.ietf.org/rfc/rfc3261.txt`
Deployment artifacts	Application-facing: wlng_at_audio_call_px21.ear Network-facing: wlng_nt_audio_call_px21.ear

Configuration Workflow for Parlay X 2.1 Audio Call/SIP

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Using the Administration Console or an MBean browser, select the MBean listed in the "Properties for Parlay X 2.1 Audio Call/SIP" section.
2. Configure the behavior of the plug-in instance. See "Reference: Attributes for Parlay X 2.1 Audio Call/SIP" for more information.
3. If required, create and load a node SLA. For details see “Defining Global Node and Service Provider Group Node SLAs” and “Managing SLAs” in the *Accounts and SLAs Guide*.

   It is not necessary to set up routing rules to the plug-in instance.

Reference: Attributes for Parlay X 2.1 Audio Call/SIP

This section describes the attributes for configuration and maintenance:

- Attribute: FromAddress

  Attribute: MediaServerFactoryJNDI
- Attribute: RetentionDuration
- Attribute: RouteURI

Attribute: FromAddress

Scope: Cluster

Unit: URI

The URI of the connecting subscriber

Attribute: MediaServerFactoryJNDI

Scope: Cluster

Format: Integer

Unit type: Seconds

Default Value: **900**

Valid Range: **1–3600**

The JNDI used by the media server driver instance

Attribute: RetentionDuration

Scope: Cluster

Format: String

Default Value: null, for the default instance

Time period to retain the audio call status before deleting it

Attribute: RouteURI

Scope: Cluster

Unit: URI

The network location of the terminal accepting the audio call

3

Parlay X 2.1 Call Notification/SIP

This chapter describes the Parlay X 2.1 Call Notification/Session Initiation Protocol (SIP) communication service in detail.

Overview of the Parlay X 2.1 Call Notification/SIP Communication Service

The Parlay X 2.1 Call Notification/SIP communication service exposes the Parlay X 2.1 Call Notification application interfaces.

The communication service connects to a SIP-IMS network using Oracle Converged Application Server. Converged Application Server is collocated with Services Gatekeeper in the network tier. The SIP servlet running on Converged Application Serve acts as both as a SIP User Agent and a SIP Proxy. Depending on which Parlay X operation and state of the call, the SIP servlet acts either as a proxy or as the calling party.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using this communication service, an application can:

- Set up and tear down notifications on call events for a specified combination of caller and callee.
- Receive additional notifications on call events related to the notification in question.
- Affect a call during call setup.

This communication service is not used to set up new calls. It is used only to reroute or terminate calls that are already in progress.

For an application to receive notifications about call setup attempts from the network, it must register its interest in these notifications by setting up a subscription in Services Gatekeeper. A subscription, or a notification, is defined by a set of addresses and a set of criteria. The criteria define the events in which the application is interested. The addresses may be translated by some mechanism in the telecom network prior to reaching Services Gatekeeper.

Two types of notifications exist:

- Simple monitoring
- Monitoring and rerouting

Simple monitoring

An application can register to be notified about the following events as the call between the caller and the callee is set up:

- Callee is busy.
- Callee is not reachable.
- Callee does not answer.
- Call is in progress.
- Call setup in progress.

Monitoring and rerouting

In addition to monitoring the state of call setup, an application can also choose to make certain changes to the call under certain conditions. An application can:

- Intercept a call setup attempt between the caller and the callee and reroute the call (to a C-party) without making an attempt to connect with the callee (B-party). An example might be a general technical support number that is routed to the appropriate call center based on time of day.

In addition, if one of the monitored events occurs (busy, not reachable, does not answer), an application can:

- Let further processing of the call be handled by the network.
- End the call.
- Reroute the call to another callee (C-party).

Requests (registration for notifications) using the Call Notification communication service flow only in one direction: from the application to Services Gatekeeper.

The communication service manages only the signalling aspect of a call. The media or audio channel is managed by the underlying telecom network. Only parties residing on the same network can be controlled unless:

- The network plug-in connects to a media gateway controller.
- One of the participants is connected to a signalling gateway so that, from a signalling point of view, all parties reside on the same network.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 Call Notification/SIP communication service, see the discussion of Parlay X 2.1 Interfaces in *Application Developer's Guide.*

For information about the RESTful Call Notification interface, see the discussion of Call Notification in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs, reading CDRs, and so on, they are the same.

Events and Statistics

The Parlay X 2.1 Call Notification/SIP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 3–1 lists IDs of the EDRs created by the Parlay X 2.1 Call Notification/SIP communication service. This list does not include EDRs created when exceptions are thrown.

Table 3–1 Event Types Generated by Parlay X 2.1 Call Notification/SIP

EDR ID	Method Called
3000	startCallNotification
3001	stopCallNotification
3002	startCallDirectionNotification
3003	stopCallDirectionNotification
8014	notifyBusy
8015	notifyCalledNumber
8016	notifyNoAnswer
8017	notifyNotReachable
8018	handleBusy
8019	handleCalledNumber
8020	handleNoAnswer
8021	handleNotReachable

Charging Data Records

Call Notification/SIP - specific CDRs are generated under the following conditions:

- After a **notifyBusy, notifyCalledNumber**, **notifyNoAnswer**, **notifyAnswer**, or **notifyNotReachable** is sent from the network.
- After a **handleBusy**, **handleCalledNumber**, **handleNoAnswer**, or **handleNotReachable** is called.

Statistics

Table 3–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 3–2 Methods and Transaction Types for Parlay X 2.1 Call Notification/SIP

Method	Transaction Type
notifyBusy	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED
notifyNotReachable	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED

Table 3–2 (Cont.) Methods and Transaction Types for Parlay X 2.1 Call Notification/SIP

Method	Transaction Type
notifyNoAnswer	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED
notifyCalledNumber	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED
handleBusy	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED
handleNotReachable	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED
handleNoAnswer	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED
handleCalledNumber	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED

Managing Parlay X 2.1 Call Notification/SIP

This section describes the properties and workflow the Parlay X 2.1 Call Notification/SIP plug-in instance.

Parlay X 2.1 Call Notification/SIP uses two parts for SIP connectivity: a part that executes as a network protocol plug-in instance in the Services Gatekeeper container and a part that executes as a SIP application in the SIP Server container. The two parts execute in different containers and must be configured in both.

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one to one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

Properties for Parlay X 2.1 Call Notification/SIP

Table 3–3 lists the technical specifications for the communication service.

Table 3–3 Properties for Parlay X 2.1 Call Notification/SIP

Property	Description
Managed object in Administration Console	*domain_name*- OCSG > *server_name* > Communication Services > Plugin_px21_call_notification_sip

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=Plugin_px21_call_notification_sip
	Type=com.bea.wlcp.wlng.plugin.callnotification.sip.management. CallNotificationMBean

Network protocol plug-in service ID	Plugin_px21_call_notification_sip
Supported Address Scheme	sip
Application-facing interfaces	com.bea.wlcp.wlng.px21.plugin.CallNotificationManagerPlugin om.bea.wlcp.wlng.px21.plugin.CallDirectionManagerPlugin com.bea.wlcp.wlng.px21.callback.CallNotificationCallback
Service type	CallNotification

Table 3–3 (Cont.) Properties for Parlay X 2.1 Call Notification/SIP

Property	Description
Exposes to the service communication layer a Java representation of:	Parlay X 2.1 Part 3: Call Notification
Interfaces with the network nodes using:	SIP: Session Initiation Protocol, RFC 3261
Deployment artifacts: NT EAR wlng_nt_call_notification_px21.ear	px21_call_notification_service.jar, Plugin_px21_call_notification_sip.jar, and px21_call_notification_sip.war
Deployment artifacts: AT EAR: Normal wlng_at_call_notification_px21.ear	px21_call_notification.war, px21_call_notification_callback.jar, and rest_call_notification.war
Deployment artifacts: AT EAR: SOAP Only wlng_at_call_notification_px21_soap.ear	px21_call_notification.war and px21_call_notification_callback.jar

Configuration Workflow for Parlay X 2.1 Call Notification/SIP

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 2.1 Call Notification/SIP" section.
2. If desired, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
3. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Reference: Operations for Parlay X 2.1 Call Notification/SIP

This section describes operations for configuration and maintenance:

- Operation: getCallDirectionSubscription
- Operation: getNotificationSubscription
- Operation: listCallDirectionSubscriptions
- Operation: listNotificationSubscriptions
- Operation: removeAllCallDirectionSubscriptions
- Operation: removeAllNotificationSubscriptions
- Operation: removeCallDirectionSubscription
- Operation: removeNotificationSubscription

Operation: getCallDirectionSubscription

Scope: Cluster

Displays call direction subscription information.

Signature:

```
getCallDirectionSubscription(Correlator: String)
```

Table 3–4 getCallDirectionSubscription Parameters

Parameter	Description
Correlator	ID for the subscription. Assigned by an application when the subscription was started.

Operation: getNotificationSubscription

Scope: Cluster

Displays call notification subscription information.

Signature:

```
getNotificationSubscription(Correlator: String)
```

Table 3–5 getNotificationDirectionSubscription Parameters

Parameter	Description
Correlator	ID for the subscription. Assigned by an application when the subscription was started.

Operation: listCallDirectionSubscriptions

Scope: Cluster

Displays a list of correlators for call direction subscriptions.

Signature:

```
listCallDirectionSubscriptions(Offset: int, Length: int)
```

Table 3–6 listCallDirectionSubscriptions Parameters

Parameter	Description
Offset	Start of offset.
Length	Number of entries returned.

Operation: listNotificationSubscriptions

Scope: Cluster

Displays a list of correlators for call notification subscriptions.

Signature:

```
listNotificationSubscriptions(Offset: int, Length: int)
```

Table 3–7 listNotificationSubscriptions Parameters

Parameter	Description
Offset	Start of offset.

Table 3–7 (Cont.) listNotificationSubscriptions Parameters

Parameter	Description
Length	Number of entries returned.

Operation: removeAllCallDirectionSubscriptions

Scope: Cluster

Removes all call direction subscriptions.

Signature:

```
removeAllCallDirectionSubscriptions()
```

Operation: removeAllNotificationSubscriptions

Scope: Cluster

Removes all call notification subscriptions.

Signature:

```
removeAllNotificationSubscriptions()
```

Operation: removeCallDirectionSubscription

Scope: Cluster

Removes a call direction subscription.

Signature:

```
removeCallDirectionSubscription(Correlator: String)
```

Table 3–8 removeCallDirectionSubscription Parameters

Parameter	Description
Correlator	ID for the subscription. Assigned by an application when the subscription was started.

Operation: removeNotificationSubscription

Scope: Cluster

Removes a call notification subscription.

Signature:

```
removeNotificationSubscription(Correlator: String)
```

Table 3–9 removeNotificationSubscription Parameters

Parameter	Description
Correlator	ID for the subscription. Assigned by an application when the subscription was started.

4 Parlay X 2.1 Multimedia Messaging/MM7

This chapter describes the Parlay X 2.1 Multimedia Messaging/MM7 communication service in detail.

Overview of the Parlay X 2.1 Multimedia Messaging/MM7 Communication Service

The Parlay X 2.1 Multimedia Messaging/MM7 communication service exposes the Parlay X 2.1 Multimedia Messaging set of application interfaces.

The communication service acts as a Value Added Service (VAS) application connecting to an MMS relay server using the MM7 protocol.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using a Multimedia Messaging communication service, an application can:

- Send multimedia messages to one or many destination addresses. The payload in these multimedia messages can be any type that can be specified using MIME, including multipart messages.
- Sign up to be notified that delivery receipts for sent multimedia messages have been received from the network.
- Receive delivery receipts on sent multimedia messages that have arrived from the network.
- Explicitly query Services Gatekeeper for delivery receipts on sent multimedia messages.
- Sign up to be notified if specified multimedia messages for the application have been received from the network.
- Receive notifications that specified multimedia messages for the application have arrived from the network. These notifications do not include the message payload, but they do provide a message ID.
- Explicitly poll Services Gatekeeper for multimedia messages sent to the application that have arrived from the network and been stored in Services Gatekeeper.

Requests can flow in two directions. They can be application-initiated or network-triggered,

Processing Application-initiated Requests

After an application has sent a multimedia message to one or more destination addresses, two different types of response can be returned:

- Send Receipts
- Delivery Receipts

Send Receipts

Send receipts are acknowledgements that the network node has received the multimedia message from the application by means of Services Gatekeeper. Although a single multimedia message may be sent to multiple destination addresses, normally only one send receipt is returned to the application. The receipt is returned synchronously in the response message to the **sendMessage** operation.

Delivery Receipts

Delivery receipts contain the delivery status of the multimedia message. They report whether the multimedia message has actually been delivered to the mobile terminal by the network. There is one delivery receipt per destination address, with one of three possible outcomes:

- Successful.
- Unsuccessful: The multimedia message could not be delivered before it expired.
- Unsupported: Delivery notification for this address is not supported. This can occur if the originating network supports delivery receipts but is unable to acquire the appropriate information for one or more destination addresses. This status is reported for each address for which this is the case.

Because actual delivery of the multimedia message may take several hours, or even days (if, for example, the mobile terminal is turned off at the time the multimedia message is sent), delivery receipts are returned asynchronously. Applications can either choose to have delivery receipts delivered to them automatically by supplying Services Gatekeeper with a callback interface or they can chose to poll Services Gatekeeper.

If the application supplies a callback interface, there are two possible outcomes:

- Services Gatekeeper sends the delivery receipt and the application receives and acknowledges it.
- Services Gatekeeper sends the delivery receipt but the application does not acknowledge reception. In this case, Services Gatekeeper stores the delivery receipt in temporary in-memory storage. The application can poll Services Gatekeeper for these receipts. Each stored delivery receipt is time stamped and, after a configurable time period, is removed.

If the application chooses not to supply a callback interface, Services Gatekeeper stores the delivery receipt in temporary in-memory storage. The application can poll Services Gatekeeper for these receipts. Each stored delivery receipt is time stamped and, after a configurable time period, removed.

Processing Network-triggered Requests

Two types of traffic destined for an application can arrive at Services Gatekeeper from the network:

- Delivery receipts for application-initiated sent multimedia messages (see "Delivery Receipts")
- Mobile-originated multimedia messages destined for the application

For an application to receive multimedia messages from the network, it must register its interest in these multimedia messages by setting up a subscription in Services Gatekeeper. A subscription, or notification, is defined by a service activation number, the destination address of the multimedia message. The service activation number may be translated by some mechanism, such as short codes, in the telecom network.

Additional criteria can be tied to the service activation number, such as the start of the first plain/text part in the multimedia message payload or the subject of the multimedia message. For the message to be accepted by Services Gatekeeper, both the service activation number and any additional criteria must match the subscription.

Mobile-originated messages to applications are routed based on the criteria that are specified when the notifications are created. The behavior for matching criteria is as follows:

- If the subject of the message is not null, the subject is used for criteria matching the specified criteria.
- If the subject of the message is null, the first word of first text attachment is used for matching the criteria.
- If the criteria is not null, messages with no subject and no text attachment are not delivered to the application.
- If the criteria is null, all messages are considered a match and delivered to the application.

Each registered subscription must be unique, and subscription attempts with overlapping criteria are rejected. The application can choose either to poll Services Gatekeeper for received multimedia messages (if polling is enabled) or to include a callback interface when it sets up the original subscription.

If a multimedia message that matches a subscription arrives at Services Gatekeeper from the network and the original subscription includes a callback interface, there are two possible results:

- Services Gatekeeper sends the notification that the multimedia message has arrived on to the application, and the application receives and acknowledges it. In this case, Services Gatekeeper stores the multimedia message in temporary in-memory storage and acknowledges the reception of the multimedia message to the network. Each stored multimedia message is time stamped and, after a configurable time period, removed. The application can poll Services Gatekeeper for any stored multimedia messages.
- Services Gatekeeper sends the notification that the multimedia message has arrived on to the application, but the application does not acknowledge reception. Services Gatekeeper does not acknowledge reception to the network. In this case, it is the responsibility of the network node to handle any further processing of the multimedia message.

If a multimedia message that matches a subscription arrives at Services Gatekeeper and the original subscription does not include a callback interface, but polling is available, the multimedia message is stored in temporary in-memory storage and Services Gatekeeper acknowledges the reception of the multimedia message to the network. The application can poll Services Gatekeeper for any such multimedia messages. Each stored multimedia message is time stamped and, after a configurable time period, removed.

If a multimedia message arrives at Services Gatekeeper and no matching subscription is found and polling is not otherwise enabled, Services Gatekeeper does not acknowledge reception to the network. It is the responsibility of the network node to handle any further processing of the multimedia message.

Polling Functionality

The polling capability must be set up in advance.

It is controlled through the combined use of parameters sent in the request and the **RequestDeliveryReportFlag** attribute. See "Attribute: RequestDeliveryReportFlag" for more information.

Short Code Translation

Messaging-capable networks use short codes and message prefixes to help route traffic and to make access to certain features easier for the end user. Instead of having to use the entire address, users can enter a short code that is mapped to the full address in the network. The Parlay X 2.1 Multimedia Messaging/MM7 communication service supports short codes and message prefixes, which allow the same short code to be mapped to multiple addresses based on the prefix to the enclosed message.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 MultiMedia Messaging/MM7 communication service, see the discussion of Parlay X 2.1 Interfaces in *Application Developer's Guide*.

For information about the RESTful Call Notification interface, see the discussion of Multimedia Messaging in *RESTful Application Developer's Guide*.

The RESTful Service Multimedia Messaging interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs, reading CDRs, and so on, they are the same

Events and Statistics

The Parlay X 2.1 Multimedia Messaging/MM7 communication service generates Event Data records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 4–1 lists the IDs of the EDRs created by the Parlay X 2.1 Multimedia Messaging/MM7 communication service.

Table 4–1 Event Types Generated by Parlay X 2.1 Multimedia Messaging/MM7

EDR ID	Description
8100	An MO message has arrived from the network.
8101	An MO delivery receipt has arrived from the network.
8102	The application has requested that a notification be started.

Table 4–1 (Cont.) Event Types Generated by Parlay X 2.1 Multimedia Messaging/MM7

EDR ID	Description
8103	The application has requested that a notification be stopped.
8104	The application has polled for a list of received messages.
8106	The application has polled for actual messages, returned as attachments.

Charging Data Records

Multimedia Messaging/MM7-specific CDRs are generated under the following conditions:

- After a **sendMessage** request has entered the network plug-in from the application.
- After a **notifyMessageDeliveryReceipt** request has entered the network plug-in from the network.
- After a **notifyMessageReception** request has been delivered to the application.
- When there is an error.

Statistics

Table 4–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 4–2 Methods and Transaction Types for Parlay X 2.1 Multimedia Messaging/MM7

Method	Transaction Type
sendMessage	TRANSACTION_TYPE_MESSAGING_MMS_SEND
deliver	TRANSACTION_TYPE_MESSAGING_MMS_RECEIVE
deliveryReport	TRANSACTION_TYPE_MESSAGING_MMS_RECEIVE

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Tunneled Parameters for Parlay X 2.1 MM7 Rel 6.8.0

This section lists, by parameter key, the parameters that can be tunneled or defined in the `<requestContext>` element of an SLA.

ChargedParty

Description

Specifies the party to be charged for a multimedia message submitted by the Value-Added Service Provider (VASP).

If defined, the **ChargedParty** xparameter is forwarded in the SOAP header in the north-bound interface.

Validated by the plug-in.

Format
String

Value
Valid values: **Sender**, **Recipient**, **Both**, **Neither**

ChargedPartyCD

Description
Specifies the address of the third party expected to pay for the multimedia message.

If defined, the **ChargedPartyID** xparameter is forwarded in the SOAP header in the north-bound interface.

Format
String

timeStamp

Description
Specifies date and time that the multimedia message was submitted.

Format
Date/Time

expiryDate

Description
Specifies the desired time for the expiration of the multimedia message.

Format
Date/Time

allowAdaptation

Description
Specifies if VASP allows adaptation of the content.

Set to `true` to allow adaptation, `false` to prohibit it.

Format
Boolean

DeliveryCondition

Description
In the event of a single "Delivery Condition", if the condition is met, the multimedia message is delivered to the recipient MMS User Agent. Otherwise the message is discarded.

Validated by the plug-in.

Format
Positive Integer

UAProf

Description
Specifies the UserAgent Name or URL to the UAProfile RDF.

Used for transferring user agent capabilities from R/S to VASP.

If not null, this parameter is forwarded to the application.

Format
String

StatusText

Description
Human-readable description of the numerical code that indicates the general type of an error.

If not null, this parameter is forwarded to the application.

Format
String

Managing Parlay X 2.1 Multimedia Messaging/MM7

This section describes the properties and workflow for Parlay X 2.1 Multimedia Messaging/MM7 plug-in instances.

Properties for Parlay X 2.1 Multimedia Messaging/MM7

Table 4–3 lists the technical specifications for the communication service

Table 4–3 Properties for Multimedia Messaging/MM7

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plug-in_instance_id*

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=same as the network protocol *instance_id* assigned when the plug-in instance is created
	Type=com.bea.wlcp.wlng.plugin.multimediamessaging.mm7.management.MessagingManagementMBean

Network protocol plug-in service ID	Plugin_px21_multimedia_messaging_mm7
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
Supported Address Scheme	tel, mailto, short

Table 4–3 (Cont.) Properties for Multimedia Messaging/MM7

Property	Description
Application-facing interfaces	com.bea.wlcp.wlng.px21.plugin.MessageNotificationManagerPlugin com.bea.wlcp.wlng.px21.plugin.ReceiveMessagePlugin com.bea.wlcp.wlng.px21.plugin.SendMessagePlugin com.bea.wlcp.wlng.px21.callback.MessageNotificationCallback
Service type	MultimediaMessaging
Exposes to the service communication layer a Java representation of:	Parlay X 3.0 Part 5: Multimedia Messaging
Interfaces with the network nodes using:	MM7
Deployment artifact: NT EAR wlng_nt_multimedia_messaging_px21.ear	Plugin_px21_multimedia_messaging_mm7.jar, px21_multimedia_messaging_service.jar, multimedia_messaging_mm7_rel5mm712.war, and multimedia_messaging_mm7_rel5mm715.war
Deployment artifact: AT EAR: Normal wlng_at_multimedia_messaging_px21.ear	px21_multimedia_messaging_callback.jar, px21_multimedia_messaging.war and rest_multimedia_messaging.war
Deployment artifact: AT EAR: SOAP Only wlng_at_multimedia_messaging_px21_soap.ear	px21_multimedia_messaging_callback.jar and px21_multimedia_messaging.war

Configuration Workflow for Parlay X 2.1 MultiMedia Messaging/MM7

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID listed in the "Properties for Parlay X 2.1 Multimedia Messaging/MM7" section.
2. Select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID given when the plug-in instance was created.
3. Configure the behavior of the plug-in instance suing the following attributes:
   - Attribute: HTTPBasicAuthentication

     If you are using HTTP basic authentication, also define:
     - Attribute: HTTPBasicAuthenticationUsername
     - Attribute: HTTPBasicAuthenticationPassword
   - Attribute: DefaultPriority
   - Attribute: MM7Version
   - Attribute: Mm7relayserverAddress

- Attribute: VaspId
- Attribute: VasId
- Attribute: RequestDeliveryReportFlag
- Attribute: XSDVersion

4. Specify heartbeat behavior. See "Configuring Heartbeats" in *System Administrator's Guide*.
5. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 2.1 Multimedia Messaging/MM7" section.
6. Provide the administrator of the MM7 server with the URL to which the MM7 server should deliver mobile originated messages and delivery reports. The default URL is

   ```
   http://IP_Address_of_NT_server:port/server:port/context-root/plug-in_instance_
   ID
   ```
   If you are using the REL-5-MM7-1-2 XSD, the default *context-root* is **mmm-mm7**.

   If you are using the REL-5-MM7-1-5 XSD, the default *context-root* is **mmm-mm7-rel5mm7-1-5**.

   If you are using the REL-6-MM7-1-4 XSDm the default *context-root* is **mmm-mm7-rel6mm7-1-4**.
7. If required, create and load a node SLA. For details see “Defining Global Node and Service Provider Group Node SLAs” and “Managing SLAs” in the *Accounts and SLAs Guide*.
8. Provision the service provider accounts and application accounts.

Provisioning Workflow for Parlay X 2.1 MultiMedia Messaging/MM7

The following steps outline the provisioning workflow for the communication service.

1. Use "Operation: enableReceiveMms" to register offline notifications. This specifies that mobile-originated messages should not result in notifications to an application but should instead be stored in Services Gatekeeper for polling.

   Use the following operations to manage the offline registrations:

   - Operation: listOfflineNotificationInfo
   - Operation: getOfflineNotificationInfo
   - Operation: removeOfflineNotificationInfo
2. Use "Operation: startMessageNotification" to register online notifications. This is to manage registrations for mobile-originated messages on behalf of an application.

   Use the following operations to manage the online registrations:

   - Operation: listOnlineNotificationInfo
   - Operation: getOnlineNotificationInfo
   - Operation: removeOnlineNotificationInfo

Reference: Attributes and Operations for Parlay X 2.1 MultiMedia Messaging/MM7

This section describes the attributes and operations for configuration and maintenance:

Attribute: DefaultPriority

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the default priority for sent MMS messages. Enter one the following:

- **normal**
- **high**
- **low**

Attribute: HTTPBasicAuthentication

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if HTTP basic authentication will be used for authentication with the MM7 server.

Set to `true` to use HTTP basic authentication, otherwise `false`.

If `true`, "Attribute: HTTPBasicAuthenticationUsername" and "Attribute: HTTPBasicAuthenticationPassword" must be specified.

Attribute: HTTPBasicAuthenticationUsername

Scope: Cluster

Unit: Not applicable

Format: String

The user name to use for HTTP basic authentication towards the MM7 server.

Attribute: HTTPBasicAuthenticationPassword

Scope: Cluster

Unit: Not applicable

Format: String

The password to use for HTTP basic authentication towards the MM7 server.

Attribute: Mm7relayserverAddress

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the address to the MM7 Relay Server. The address is an HTTP url.

Attribute: MM7Version

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the version of the MM7 protocol to be used. Applicable versions are:

- **5.3.0**
- **6.8.0**

Attribute: VasId

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the Value Added Service (VAS) ID to be used for the plug-in instance when connecting to the MMSC.

Attribute: VaspId

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the Value Added Service Provider (VASP) ID to be used for the plug-in instance when connecting to the MMSC.

Attribute: RequestDeliveryReportFlag

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies how the plug-in instance requests and handles delivery reports for sent messages. Enter one of the following:

- **0**: Delivery notifications are not processed, which means that no polling functionality is available to the applications using the communication service.
- **1**: Delivery notifications are processed if the application provided a **receiptRequest** in the **SendMessage** requests or the application provided a tunnelled parameter with ID **com.bea.wlcp.wlng.plugin.multimediamessaging.RequestDeliveryReportFlag** with the value `true` in the SOAP header of the **SendMessage** request.
- **2**: Delivery notifications are always processed.

Attribute: ServiceCode

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the service used for billing purposes.

Attribute: XSDVersion

Scope: Server

Unit: Not applicable

Format: String

The MM7 xsd version that should be used for requests towards the MMSC.

Enter one of the following:

- **REL-5-MM7-1-0** to use an altered version of the **REL-5-MM7-1-0.xsd**. The altered version allows use of delivery notifications when the MMC-S requires this version of the xsd. This is a requirement when connecting to, among others, Comverse MMSCs.
- **REL-5-MM7-1-2** to use **REL-5-MM7-1-2.xsd**
- **REL-5-MM7-1-5** to use **REL-5-MM7-1-5.xsd**
- **REL-6-MM7-1-4** to use **REL-6-MM7-1-4.xsd**

Operation: enableReceiveMms

Scope: Cluster

Adds an offline notification for applications that will poll for mobile originated messages. Mobile-originated messages matching this notification will not result in a callback to an application. Instead the application has to use the correlator returned by this method and poll for new messages.

Returns the correlator uniquely identifying the new notification.

Signature:

```
enableReceiveMms(shortcode: String, criteria: String, appInstanceID: String)
```

Table 4–4 enableReceiveMms Parameters

Parameter	Description
shortcode	The destination address or service activation number of the Multmedia message. Prefixed with the URI, for example **tel:**
criteria	The first word in the text in the subject field of the MMS message to match. Exact matches only.
appInstanceID	The application instance ID associated with the notification.

Operation: getOfflineNotificationInfo

Scope: Cluster

Displays information about a notification registered offline. See "Operation: enableReceiveMms" for more information.

Signature:

```
getOfflineNotificationInfo(correlator: String)
```

Table 4–5 getOfflineNotificationInfo Parameters

Parameter	Description
correlator	Correlator identifying the notification.

Operation: getOnlineNotificationInfo

Scope: Cluster

Displays information about a notification registered by an application or by "Operation: startMessageNotification".

Signature:

```
getOnlineNotificationInfo(correlator: String)
```

Table 4–6 getOnlineNotificationInfo Parameters

Parameter	Description
correlator	Correlator identifying the notification.

Operation: listOfflineNotificationInfo

Scope: Cluster

Displays a list of all notifications registered offline by Operation: enableReceiveMms.

Signature:

```
listOfflineNotificationInfo()
```

Operation: listOnlineNotificationInfo

Scope: Cluster

Displays a list of all notifications registered online by the application or by "Operation: startMessageNotification".

Signature:

```
listOnlineNotificationInfo()
```

Operation: removeOfflineNotificationInfo

Scope: Cluster

Removes a notification registered offline using "Operation: enableReceiveMms".

Signature:

```
removeOfflineNotificationInfo(Registration Identifier: String)
```

Table 4–7 removeOfflineNotificationInfo Parameters

Parameter	Description
Registration Identifier	ID of the notification.

Operation: removeOnlineNotificationInfo

Scope: Cluster

Removes a notification registered by an application or on behalf of an application by "Operation: startMessageNotification".

Signature:

```
removeOnlineNotificationInfo(Registration Identifier: String)
```

Table 4–8 removeOnlineNotificationInfo Parameters

Parameter	Description
Correlator	ID of the notification.

Operation: startMessageNotification

Scope: Cluster

Creates an online notification on behalf of an application. Produces the same results as if an application registered for notifications using the **startMessageNotification** operation in the Parlay X 2.1 Multimedia Messaging MessageNotificationManager interface.

This operation can be used, for example, if the application is not allowed to register for notifications by restrictions defined in its SLA. Returns a correlator that uniquely identifies the notification.

Signature:

```
startMessageNotification(endpoint: String, shortcode: String, criteria: String,
appInstanceID: String)
```

Table 4–9 startMessageNotification Parameters

Parameter	Description
endpoint	Notification endpoint implemented by the application. This endpoint implements the Parlay X 2.1 Multimedia MessagingMessageNotification interface. Format: URL
shortcode	Destination address for the MMS message. Must have the prefix **tel:**; for example, **tel:1234**
criteria	The first word in the text in the subject field of the MMS message to match. Exact matches only.
appInstanceID	ID of the application instance for the application.

5 Parlay X 2.1 Presence/SIP

This chapter describes the Parlay X 2.1 Presence/SIP communication service in detail.

Overview of the Parlay X 2.1 Presence/SIP Communication Service

The Parlay X 2.1 Presence/SIP communication service exposes both the watcher aspect and the presentity aspect of the Parlay X 2.1 Presence set of application interfaces.

The communication service connects to a SIP-IMS network using Oracle Converged Application Server. Converged Application Server is collocated with Services Gatekeeper in the network tier.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Presence information is a collection of data on an end user's status, such as current activity, environment, available communication means, and contact addressees. Using the presence functionality, an application can function as a client in two modes: as a watcher or as a presentity. A watcher is a client that is interested in consuming presence information. A presentity is a client that allows its presence information to be delivered to watchers.

Client as Presence Consumer

An application acting as a watcher can:

- Subscribe to obtain presence data.

  Each subscription requires authorization by the presentity. The authorization is returned asynchronously via the notification interface.

- Choose to acquire presence information when a subscription has been established using:

  – Direct synchronous polling. This is only effective for a single presentity. Groups are not supported.

  – Specific notifications. These can be used for a single presentity. The watcher sets a notification trigger based on certain user presence attribute changes.

    Possible attribute types include:

    – Activity (User's status: Available, Busy, At Lunch, and so on.)

    – Place (User's current location: Home, In a Public Place, and so on.)

  - Privacy (Degree of privacy the user has: Surrounded by Others, Alone and Can Talk Openly, and so on.)
  - Sphere (User's personal status: In his Work Capacity; In his Personal Capacity)
  - Communication means (Type of communication client preferred: Phone, Email, SMS, and so on.)
  - Other (name-value pair for arbitrary information)

  Non-attribute notification parameters can include:

  - Maximum frequency of notifications
  - Duration of time during which notifications should occur
  - Maximum number of notifications
  - Whether status should be checked immediately after notification setup

- End notifications. In this case, the subscription to the presentity is retained, but the specific notification is ended.
- Receive information that:
  - The initial conditions of the notification setup have been met (count or duration) and this specific setup has been ended.
  - The subscription itself has ended.

Client as Presence Supplier

An application acting as a presentity can:

- Publish present information.
- Get a list of new watchers who have asked to subscribe to the client's presence information.
- Approve new watchers and update the subscriptions of current watchers.
- Get a list of currently subscribed watchers.
- Block the subscription of a currently subscribed watcher.

The presentity functionality requires a presence server in the underlying network. To approve new watchers and update the subscriptions of current watchers, there must also be a data manipulation server (DMS) in the underlying network. The block functionality is supported in two modes, one using a DMS and one not.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 Presence communication service, the discussion of Parlay X 2.1 Interfaces in *Application Developer's Guide*.

For information about the RESTful Presence interface, see the discussion of Presence in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on, they are the same.

Events and Statistics

The Parlay X 2.1 Presence/SIP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 5–1 lists IDs of the EDRs created by the Presence/SIP communication service. This list does not include EDRs created when exceptions are thrown.

Table 5–1 Event Types Generated by Parlay X 2.1 Presence/SIP

EDR ID	Method Called
2000	notifyReceived
2001	makeNotifySubscriptionCallback (includes Endpoint, string)
2002	makeSubscriptionEndedCallback (includes Endpoint, string)
2003	makeStatusChangedCallback (includes Endpoint, string)
2004	makeStatusEndCallback (includes Endpoint, string)
2005	subscribePresence (processes from application)
2006	getUserPresence
2007	startPresenceNotification
2008	endPresenceNotification
2009	publish
2010	blockSubscription
2011	getMyWatchers
2012	getOpenSubscriptions
2013	updateSubscriptionAuthorization
2015	onRequest (Watcher Info Notify)

Charging Data Records

Presence/SIP-specific CDRs are generated under the following conditions:

- After the result of a poll for presence is successfully returned to the application.
- After a notification for presence information is successfully sent to the application.

A Presence CDR contains an **additional_info** field for a notification call. This field contains the endpoint.

Statistics

Table 5–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 5–2 Methods and Transaction Types for Parlay X 2.1 Presence/SIP

Method	Transaction Type
getUserPresence	TRANSACTION_TYPE_PRESENCE_SERVICE_ INITIATED
makeStatusChangedCallback	TRANSACTION_TYPE_PRESENCE_NETWORK_ INITIATED

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Tunneled Parameters for Parlay X 2.1 Presence / SIP

This section lists the parameters that can be tunneled.

expireskey

Description
This value is used to cancel a subscription to a user presence. The parameter is configured in the SIP plug-in.

If set, this value replaces the default **SubscribeExpiryValue** configured in the Presence MBean.

Can be set using parameter tunneling.

Format
String

Example
```
<xparams> <param key="expireskey" value="1210"/> </xparams>
```

passidkey

Description
This value is used among trusted intermediaries to assert the identity of a user sending a SIP message as it was identified by authentication.

Can be set using parameter tunneling.

Format
String

Example
```
<xparams> <param key="passidkey" value="sip:leffe@keffo.com"/> </xparams>
```

Managing Parlay X 2.1 Presence/SIP

This section describes the properties and workflow for the Parlay X 2.1 Presence/SIP plug-in instance. It also describes the caches used by the plug-in instance.

Parlay X 2.1 Presence/SIP uses two parts for SIP connectivity: a part that executes as a network protocol plug-in instance in the Services Gatekeeper container and a part that executes as a SIP application in the SIP Server container.

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one to one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

URI Cache

In Parlay X 2.1 Presence, the SIP URI of the user (the presentity in Presence Supplier cases, the watcher in Presence Consumer cases) is not passed as an argument. Instead, Services Gatekeeper maps the user URI to the application instance ID of the user application. The URI mapping is configured as a part of the service provider and application provisioning workflow and is then stored in the URI cache.

For requests that originate from an application, the URI is fetched from this cache before being put into the **from** header in the SIP requests. For application-terminating requests, the **to** header URI passed in the SIP **NOTIFY** requests is used to look up the account user name and application instance ID.

In the case of the Presence Supplier interface, the application can override the default mapping by including a tunneled parameter in the header of its request.

Subscriptions Cache

Every subscription, pending or not, is stored in the subscriptions cache during the subscription's lifetime. It is added when an application invokes the **subscribePresence** operation on the application-facing interface. It is removed when the subscription is terminated.

Notifications Cache

All registered notifications are cached. The entries are created when an application invokes **startPresenceNotification** on the application-facing interface. They are removed when **endPresenceNotification** is called, the end criteria are reached, or the subscription is ended.

Properties for Parlay X 2.1 Presence/SIP

Table 5–3 lists the technical specifications for the communication service.

Table 5–3 Properties for Parlay X 2.1 Presence/SIP

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > Plugin_px21_presence_sip

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=Plugin_px21_presence_sip
	Type=com.bea.wlcp.wlng.plugin.presence.sip.management.PresenceMBean

Network protocol plug-in service ID	Plugin_px21_presence_sip
Network protocol plug-in instance ID	Plugin_px21_presence_sip
Supported Address Scheme	sip

Table 5–3 (Cont.) Properties for Parlay X 2.1 Presence/SIP

Property	Description
Application-facing interfaces	com.bea.wlcp.wlng.px21.plugin.PresenceConsumerPlugin com.bea.wlcp.wlng.px21.plugin.PresenceSupplierPlugin com.bea.wlcp.wlng.px21.callback.PresenceNotificationCallback
Service type	Presence
Exposes to the service communication layer a Java representation of:	Parlay X 2.1 Part 14: Presence
Interfaces with the network nodes using:	SIP: Session Initiation Protocol, RFC 3261.
Deployment artifact: NT EAR wlng_nt_presence_px21.ear	Plugin_px21_presence_sip.jar, px21_presence_service.jar, and px21_presence_sip.jar
Deployment artifact: AT EAR: Normal wlng_at_presence_px21.ear	px21_presence.war, px21_presence_callback.jar, and rest_presence.war
Deployment artifact: AT EAR: SOAP Only wlng_at_presence_px21_soap.ear	px21_presence.war and px21_presence_callback.jar

Configuration Workflow for Parlay X 2.1 Presence/SIP

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Select the MBean described in "Properties for Parlay X 2.1 Presence/SIP".
2. Configure the attributes of the network protocol plug-in instance:
   - Attribute: DefaultNotificationCount
   - Attribute: DefaultNotificationDuration
   - Attribute: NotificationCleanupTimerValue
   - Attribute: NotificationCleanupTimerValue
3. Configure connection information to the SIP server:
   - Attribute: SubscriptionCleanupTimerValue
4. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 2.1 Presence/SIP"section.
5. If required, create and load a node SLA. For details see “Defining Global Node and Service Provider Group Node SLAs” and “Managing SLAs” in the *Accounts and SLAs Guide*.
6. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Provisioning Workflow for Parlay X 2.1 Presence/SIP

For every application, use "Operation: setApplicationInstanceSIPURI" to define a mapping between a SIP URI and application instance ID.

Use "Operation: getApplicationInstanceSIPURI" to display the mapping.

If an application is deleted, use "Operation: removeApplicationInstanceFromCache" remove the data for the application.

Management Operations for Parlay X 2.1 Presence/SIP

- Operation: clearCache
- Operation: listNotificationsCache
- Operation: listSubscriptionsCache
- Operation: listURImappingCache
- Operation: updateSubscriptionToBeconfirmed

Reference: Attributes and Operations for Parlay X 2.1 Presence/SIP

This section describes the attributes and operations for configuration and maintenance:

- Attribute: DefaultNotificationCount
- Attribute: DefaultNotificationDuration
- Attribute: NotificationCleanupTimerValue
- Attribute: PresenceServerAddress
- Attribute: PresenceXDMSAddress
- Attribute: PresenceXDMSPresrulesPostfix
- Attribute: PresenceXDMSPresrulesPrefix
- Attribute: PresenceXDMSProviderClassName
- Attribute: SubscriptionCleanupTimerValue
- Attribute: SubscribeExpiryValue
- Operation: clearCache
- Operation: getApplicationInstance
- Operation: getApplicationInstanceSIPURI
- Operation: listNotificationsCache
- Operation: listSubscriptionsCache
- Operation: listURImappingCache
- Operation: removeApplicationInstanceFromCache
- Operation: removeNotification
- Operation: removeSubscription
- Operation: setApplicationInstanceSIPURI
- Operation: updateSubscriptionToBeconfirmed

Attribute: DefaultNotificationCount

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the default notification count value. This value is used if none is provided in the **startPresenceNotification** requests from the application.

Attribute: DefaultNotificationDuration

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the value of the default notification duration. This value is used if none is provided in the **startPresenceNotification** request from the application.

Example values:

- 86400 seconds is 1 day
- 604800 seconds is 1 week

Attribute: NotificationCleanupTimerValue

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the value of the timer used for checking on and cleaning up old notifications.

Each time the timer expires, it initiates a check for old notifications. If an old notification is found during the check, it is removed internally and a **statusEnd** callback is made to the application.

Attribute: PresenceServerAddress

Scope: Cluster

Unit: Not applicable

Format: String formatted as a SIP URI

Specifies the address to which the subscribe messages are sent. It can be the IP address of the Presence server or another IMS node that proxies the request.

Attribute: PresenceXDMSAddress

Scope: Cluster

Unit: Not applicable

Format: String formatted as a SIP URI

Specifies the XCAP root URI of the XDM server.

Attribute: PresenceXDMSPresrulesPostfix

Scope: Cluster

Unit: Not applicable

Format: String formatted as a partial path

Specifies the last-part of the XCAP Document selector for Presence rules. This part is after the XCAP User Identifier(XUI). Generally the selector URI should be the string of the following type: [XCAP_ROOT]/[AUID]/users/[XUI]/[document_name]

Example:

```
/presrules
```

Attribute: PresenceXDMSPresrulesPrefix

Scope: Cluster

Unit: Not applicable

Format: String formatted as a path

The pre-part of the XCAP Document selector for presence rules. This part is before the XCAP User Identifier(XUI). Generally the selector URI should be the string of the following type: [XCAP_ROOT]/[AUID]/users/[XUI]/[document_name]

Example:

```
/services/pres-rules/users/
```

Attribute: PresenceXDMSProviderClassName

Scope: Cluster

Unit: Not applicable

Format: Class name as string

The class name of XDM Server provider. This is a pluggable function to allow third party vendors of XDMS to customize their own XCAP client. This class must implement the "com.bea.wlcp.wlng.plugin.presence.sip.south.xcap.XCAPClient" interface.

Attribute: SubscriptionCleanupTimerValue

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the value of the timer used for checking on and cleaning up old subscriptions.

Each time the timer expires, it initiates a check for old subscriptions. If an old subscription is found during the check, it is removed and a callback made to the application.

Attribute: SubscribeExpiryValue

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the maximum lifetime of a subscription.

This value might not be accepted by the Presence Server. The Presence server may override this expiry value and give the suggested value to be used in the first **NOTIFY** sent to the plug-in instance. In that case, the lifetime for the presence subscription will be according to the value received from the Presence Server.

Operation: clearCache

Scope: Cluster

Clears one or all caches used by this plug-in instance.

Note: Use this method with care.

Signature:

```
clearCache(cacheToClear: String)
```

Table 5–4 clearCache Parameters

Parameter	Description
cacheToClear	Name of the cache to clear. Valid options: ■ NOTIFICATIONS - clears the notifications cache ■ SUBSCRIPTIONS - clears the subscriptions cache ■ URIMAPPINGS - clears the URI mappings cache ■ ALL - clears all caches.

Operation: getApplicationInstance

Scope: Cluster

Displays the application instance ID associated with a SIP URI. The application instance identifies an application.

Signature:

```
getApplicationInstance(Uri: String)
```

Table 5–5 getApplicationInstance Parameters

Parameter	Description
Uri	The SIP URI.

Operation: getApplicationInstanceSIPURI

Scope: Cluster

Displays the SIP URI associated with an application instance. The application instance is used by an application.

Signature:

```
getApplicationInstanceSIPURI(ApplicationInstanceID: String)
```

Table 5–6 getApplicationInstanceSIPURI Parameters

Parameter	Description
ApplicationInstanceID	ID of the application instance.

Operation: listNotificationsCache

Scope: Cluster

Displays the cache where notification information is stored. Used for troubleshooting.

Note: Use with caution: lists data from all entries in the notification cache.

Signature:

```
listNotificationsCache()
```

Operation: listSubscriptionsCache

Scope: Cluster

Displays the cache where subscription information is stored. Used for troubleshooting.

Note: Use with caution: lists data from all entries in the subscriptions cache.

Signature:

```
listSubscriptionsCache()
```

Operation: listURImappingCache

Scope: Cluster

Displays the cache where URI mappings information is stored. Used for troubleshooting.

Note: Use with caution: lists data from all entries in the URI mappings cache.

Signature:

```
listURImappingCache()
```

Operation: removeApplicationInstanceFromCache

Scope: Cluster

Removes entries that are associated with an application instance from the URI mappings cache. If an application instance has been removed, the associated entries in the cache must be removed, too.

Signature:

```
removeApplicationInstanceFromCache(ApplicationInstance: String)
```

Table 5–7 removeApplicationInstanceFromCache Parameters

Parameter	Description
ApplicationInstance	ID of the application instance.

Operation: removeNotification

Scope: Cluster

Removes a notification. The application will not be notified that the notification has been removed.

Signature:

```
removeNotification(ApplicationInstanceID: String, Presentity: String)
```

Table 5–8 removeNotification Parameters

Parameter	Description
ApplicationInstanceID	ID of the application instance.
Presentity	ID of the presentity.

Operation: removeSubscription

Scope: Cluster

Removes a subscription and notifications. The application will not be notified that the subscription has been removed.

Signature:

```
removeSubscription(ApplicationInstanceID: String, Presentity: String)
```

Table 5–9 removeSubscription Parameters

Parameter	Description
ApplicationInstanceID	ID of the application instance.
Presentity	ID of the presentity.

Operation: setApplicationInstanceSIPURI

Scope: Cluster

Associates a SIP URI with an application instance. See "URI Cache".

Signature:

```
setApplicationInstanceSIPURI(applicationInstanceID: String, URI: String)
```

Table 5–10 setApplicationInstanceSIPURI Parameters

Parameter	Description
applicationInstanceID	ID of the application instance.
URI	SIP URI. For example: sip:name@somedomain.org

Operation: updateSubscriptionToBeconfirmed

Scope: Cluster

Updates the xml rule in XDMS to status: confirm.

Forces a blocked subscription to be pending or confirmed.

Signature:

```
updateSubscriptionToBeconfirmed(presentity: String, watcher: String)
```

Table 5–11 updateSubscriptionToBeconfirmed Parameters

Parameter	Description
Presentity	ID of the application instance.
Watcher	SIP URI. For example: sip:name@somedomain.org

6 Parlay X 2.1 Short Messaging/SMPP

This chapter describes the Parlay X 2.1 Short Messaging/Short Message Peer to Peer (SMPP) communication service in detail.

Overview of the Parlay X 2.1 Short Messaging/SMPP Communication Service

The Parlay X 2.1 Short Messaging/SMPP communication service exposes the Parlay X 2.1 Short Messaging set of application interfaces.

The communication service acts as an External Short Message Entity (ESME) that connects to a Short Messaging Service Center (SMSC) over TCP/IP.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Services Gatekeeper provides support for the billing identification identifier, **smpp_billing_id**, defined in SMPP Specification 5.0, through the use of a tunneled parameter. It also supports the **ussd_service_operatio**n, which was added as an optional parameter to the **DELIVER_SM** operation as a tunneled parameter in SMPP Specification 5.0. See "smpp_billing_id" and "ussd_service_operation" for information about these parameters.

Using a Short Messaging communication service, an application can:

- Send short messages to one or many destination addresses. The payload in these short messages can be text, logos, or ringtones

  Logos must be in either Smart Messaging or Enhanced Messaging Service (EMS) format. The image is not scaled. Ringtones must be in either Smart Messaging or EMS (iMelody) format.
- Request to be notified that delivery receipts for sent short messages have been received from the network.
- Receive delivery receipts on sent short messages that have arrived from the network.
- Explicitly query Services Gatekeeper for delivery receipts on sent short messages.
- Subscribe to be notified if specified short messages for the application have been received from the network.
- Receive notifications that specified short messages for the application have arrived from the network. These notifications include the short message payload.

- Explicitly poll Services Gatekeeper for short messages sent to the application that have arrived from the network and been stored in Services Gatekeeper.

Requests can flow in two directions: from the application to the network (called application-initiated or mobile-terminated) and from the network to the application (called network-triggered or mobile-originated). Both of these scenarios are covered in the following sections.

Processing Application-Initiated Requests

After an application has sent a short message to one or more destination addresses, two different types of responses can be returned:

- Send Receipts
- Delivery Receipts

Send Receipts

Send receipts are acknowledgements that the network node has received the short message from the application by means of Services Gatekeeper. Although a single short message may be sent to multiple destination addresses, Services Gatekeeper normally returns only one send receipt to the application. The receipt is returned synchronously in the response message to the **sendSms** operation.

Delivery Receipts

Delivery receipts contain the delivery status of the short message; that is, whether the short message has actually been delivered by the network to the mobile terminal. There is one delivery receipt per destination address, with one of three possible outcomes:

- Successful: In the case of concatenated short messages, this is returned only when all the parts have been successfully delivered.
- Unsuccessful: The short message could not be delivered before it expired.
- Unsupported: Delivery notification for this address is not supported. This can occur if the originating network supports delivery receipts but is unable to acquire the appropriate information for one or more destination addresses. This status is reported for each address for which this is the case.

Because actual delivery of the short message may take several hours, or even days (if, for example, the mobile terminal is turned off at the time the short message is sent), delivery receipts are returned asynchronously. Applications can choose either to have delivery receipts delivered to them automatically by supplying Services Gatekeeper with a callback interface or they can poll Services Gatekeeper.

If the application supplies a callback interface, there are two possible outcomes:

- Services Gatekeeper sends the delivery receipt and the application receives and acknowledges it.
- Services Gatekeeper sends the delivery receipt but the application does not acknowledge reception. In this case, Services Gatekeeper stores the delivery receipt in a reliable storage. The application can poll Services Gatekeeper for these receipts. Each stored delivery receipt is time stamped and, after a configurable time period, is removed.

If the application chooses not to supply a callback interface, Services Gatekeeper stores the delivery receipt in reliable storage. The application can poll Services Gatekeeper for these receipts. Each stored delivery receipt is time stamped and, after a configurable time period, is removed.

To correlate a sent message with a delivery receipt from the network node, Services Gatekeeper stores the information about the message for a period of time. This information has a life span. If the delivery receipt does not arrive prior to the expiration of the message, a cancel request for the message is sent to the SMSC.

Processing Network-Triggered Requests

Two types of traffic destined for an application can arrive at Services Gatekeeper from the network. They are:

- Delivery receipts for application-initiated sent short messages
- Mobile-originated short messages destined for the application

For an application to receive short messages from the network, it must register its interest in these short messages by setting up a subscription in Services Gatekeeper. A subscription, or notification, is defined by a service activation number, which is the destination address to which the mobile sender directs the short message. This is usually a short code.

Additional criteria can be tied to the service activation number, such as the first word of the text in the short message payload. For Services Gatekeeper to accept a message, both the service activation number and the additional criteria must match the details set up in the subscription. Each registered subscription must be unique, and subscription attempts with overlapping criteria are rejected. The application can either poll Services Gatekeeper for received short messages or include a callback interface in setting up the original subscription.

If a short message that matches a subscription arrives at Services Gatekeeper from the network and the original subscription includes a call-back interface, there are two possible results:

- Services Gatekeeper sends the short message to the application, and the application receives and acknowledges it. In this case Services Gatekeeper acknowledges the reception of the short message to the network.
- Services Gatekeeper sends the short message to the application, but the application does not acknowledge reception. In this case, Services Gatekeeper can store the short message in reliable storage, if offline provisioning has occurred and a registration identifier has been established. In such a case, Services Gatekeeper acknowledges the reception of the short message to the network. If no provisioning has been done, Services Gatekeeper returns an error to the network. The application can poll Services Gatekeeper for any stored short messages. Messages are removed after the polling application has had the chance to consume them. Each stored short message is time stamped and, after a configurable time period, is removed from storage.

If a short message that matches a subscription arrives at Services Gatekeeper, and the original subscription does not include a callback interface, the short message is stored in reliable storage and Services Gatekeeper acknowledges the reception of the short message to the network. The application can poll Services Gatekeeper for any such short messages. Each stored short message is time stamped and, after a configurable time period, is removed.

If a short message arrives at Services Gatekeeper and no matching subscription is found, Services Gatekeeper does not acknowledge reception to the network. It is the responsibility of the network node to handle any further processing of the short message.

Connection Handling and Provisioning

The Parlay X 2.1 Short Messaging/SMPP communication service uses the Services Gatekeeper SMPP Server Service to establish and manage southbound connections between Services Gatekeeper and SMSCs. The SMPP Server Service is deployed as an Oracle WebLogic Server service.

See "System Properties for SMPP Server Service" and "Reference: Attributes and Operations for SMPP Server Service" for information about configuration options pertaining to these client connections.

The client connection ID is created on the plug-in's successful bind with the SMSC. The connection ID changes on a successful rebind.

When a client connection is successfully established, the connection is verified periodically by using **ENQUIRE_LINK** requests (heartbeats). If the **ENQUIRE_LINK** requests fail a configurable number of times, Services Gatekeeper attempts to reconnect with the SMSC. If the reconnect attempts fail a configurable number of times, the client connection is closed and removed.

The plug-in instance MBean provides the following configurable timers for southbound connections between Services Gatekeeper and SMSCs:

- Connection timer: This timer sets the heartbeat interval that Services Gatekeeper uses to request the connection status on the client connection. If the **ENQUIRE_LINK** requests fail, Services Gatekeeper closes the connection and attempts to reconnect. See "Attribute: EnquireLinkTimerValue" for more information.
- Transaction timer: This timer establishes the interval between an SMPP request to the SMSC and the corresponding SMPP response. If the interval is reached, Services Gatekeeper does not resend the request. In this case, Services Gatekeeper removes the transaction information and discards the PDU response. See "Attribute: RequestTimerValue" for more information.

Multiple Connections and Multiple Plug-in Instances

A Parlay X 2.1 Short Messaging/SMPP plug-in can bind to an SMSC as an ESME transmitter/receiver or transceiver. If more than one account in the SMSC is used, create one plug-in instance for each account. If more than one SMSC is used, create a plug-in instance for each account in each of the SMSCs.

If plug-in instances have the same bind type, they can share a connection to the SMSC. If they have different bind types, each must have its own client connection.

Each plug-in instance executes on all network tier servers. Shared storage is used, so network-triggered messages and delivery notifications can be accepted by all network tier servers and match them with all application subscriptions, thus creating a configuration with high availability.

Windowing

To maximize throughput, the Parlay X 2.1 Short Messaging/SMPP communication service supports windowing on the network-facing interfaces. Windowing provides a way to specify the amount of data that can be transmitted without receiving an acknowledgment.

Windowing for requests to the SMSC is configured in the plug-in.

Requests wait in a windowing queue until they can be submitted. Two attributes apply to the windowing queue. The **WindowingMaxQueueSize** attribute sets the size of the queue, specifying the maximum number of requests that can wait in the queue at one time. The **WindowingMaxWaitTime** attribute specifies the maximum amount of time that a single request can wait in the windowing queue.

The **WindowingSize** attribute sets the number of unacknowledged requests that can be sent simultaneously.

A request moves from the windowing queue to the window. From the window it is submitted for processing. A submitted request remains in the window until its response is received. When the response is received, the request is released and another request can be moved from the windowing queue to the window.

If any one of these three windowing parameters is set to a value less than zero, windowing is turned off. If all of these three parameters are greater than zero, windowing is turned on.

For descriptions of these attributes see:

- Attribute: WindowingMaxQueueSize
- Attribute: WindowingMaxWaitTime
- Attribute: WindowingSize

If the windowing request queue is full or the timer has expired, the request is not sent and an error code is returned to the plug-in instance.

Segments

If an SMS message is larger than the maximum payload size, the message content is concatenated into segments before it is delivered to the application.

The maximum payload size defaults to the standard set by the Parlay X 2.1 Short Messaging specification. You can set the maximum payload size using the **wlng.smpp.max_payload_size** system property on the command line when starting Services Gatekeeper.

For configuration attributes regarding segments, see:

- Attribute: ReceiveSegmentsWaitTime
- Attribute: ReceiveSmsIgnoreMissingSegments

Short Code Translation

Messaging-capable networks use short codes and message prefixes to help route traffic and to make access to certain features easier for the end user. Instead of having to use the entire address, users can enter a short code that is mapped to the full address in the network. The Parlay X 2.1 Short Messaging/ SMPP communication service supports short codes and message prefixes, which allow the same short code to be mapped to different applications based on the prefix to the enclosed message.

Load Balancing, High Availability, and Failover

To optimize system utilization, applications should load-balance application-triggered requests among all application tier servers.

When there are multiple connections to the SMSC within a single plug-in instance, the SMPP Server Service selects one of the connections to the SMSC.

A prerequisite for high-availability for the Parlay X 2.1 Short Messaging/SMPP communication service is redundant network tier servers, redundant network interface cards in each network tier server, and a redundant set of SMPP servers to connect to. High availability between Services Gatekeeper and the network is achieved by using at least two different plug-in instances per network tier server and having the plug-in instances connect to different SMPP servers.

High availability behavior is as follows:

- The SMPP server runs in every network node. If one network node is unavailable, mobile-terminated requests are automatically routed to a healthy node. Related delivery reports are routed from the healthy network node that handled the request to the application.
- If the application tier is unavailable, mobile-originated messages are routed to a healthy application tier node.
- In a Services Gatekeeper cluster, if the server becomes unavailable after sending a **SUBMIT_SM** request to and receiving the **SUBMIT_SM_RESP** from the SMSC, the SMSC routes the subsequent delivery receipt to another server. This other server retrieves the message information from cluster-level storage and processes it.
- In a Services Gatekeeper cluster, if a server becomes unavailable after sending a **SUBMIT_SM** request to and receiving the **SUBMIT_SM_RESP** from an application, the application routes the subsequent cancel, query, or replace request to another server. This other server retrieves the message information from cluster-level storage and processes it.

Character Set

The SMPP protocol expects the sender name value to be in ASCII characters. The use of non-ASCII characters may cause the request to become garbled or even to be removed at the SMSC

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 Short Messaging/SMPP communication service, see the discussion of Parlay X 2.1 Interfaces in *Application Developer's Guide*.

For information about the RESTful Short Messaging interface, see the discussion of Short Messaging in *RESTful Application Developer's Guide*.

The RESTful Service Short Messaging interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs, reading CDRs, and so on, they are the same.

Events and Statistics

The Parlay X 2.1 Short Messaging/SMPP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data

Table 6–1 lists the IDs of the EDRs created by the Parlay X 2.1 Short Messaging/SMPP communication service. This list does not include EDRs created when exceptions are thrown.

Table 6–1 Event Types Generated by Parlay X 2.1 Short Messaging/SMPP

EDR ID	Method Called
6000	notifySmsDeliveryReceipt
6001	notifiySmsReception
7000	sendSms
7001	sendSmsLogo
7002	sendSmsRingtone
7003	startSmsNotification
7004	stopSmsNotification
7011	getSmsDeliveryStatus
7012	getReceivedSms

See Table 20–2, " Event Types Generated by the SMPP Server Service" for the list of EDRs generated by the SMPP Server Service.

Charging Data Records

Short Messaging/SMPP-specific CDRs are generated under the following conditions:

- After a **sendSms** request is sent from Services Gatekeeper to the network.
- After a **reportNotification** request is sent from the network to Services Gatekeeper, indicating that a delivery receipt has been returned for the application.
- When a mobile-originated message has been successfully delivered to the application.

Statistics

Table 6–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 6–2 Methods and Transaction Types for Parlay X 2.1 Short Messaging/SMPP

Method	Transaction Type
sendSms	TRANSACTION_TYPE_MESSAGING_SEND
sendSmsLogo	TRANSACTION_TYPE_MESSAGING_SEND
sendSmsRingtone	TRANSACTION_TYPE_MESSAGING_SEND
receivedMobileOriginatedSMS	TRANSACTION_TYPE_MESSAGING_RECEIVE

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Tunneled Parameters for Parlay X 2.1 Short Messaging / SMPP

This section lists the parameters that can be tunneled or defined in the `<requestContext>` element of an SLA.

The last four parameters described, **dest_bearer_type**, **service_type**, **ussd_service_operation**, and **its_session_info**, are used to support Unstructured Supplementary Service Data (USSD).

sms.protocol.id

Description

This parameter defines the mandatory SMPP **protocol_id** parameter.

It is valid for application-initiated requests only.

This parameter can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

This parameter key name can be configured in the **wlng.sms.protocol.id** system property. The default is **sms.protocol.id**.

Format

Integer

Value

Value range is **0–65535**.

source_port

Description

This parameter defines the optional SMPP **source_port** parameter.

It is valid for application-initiated requests.

It is valid for network-triggered requests if the forwarding parameter is enabled. See "Attribute: ForwardXParams" for more information.

This parameter can be set using parameter tunneling.

Format

Integer

Value

Value range is **0–65535**.

destination_port

Description

This parameter defines the optional SMPP **destination_port** parameter.

It is valid for application-initiated requests.

It is valid for network-triggered requests if the forwarding parameter is enabled. See "Attribute: ForwardXParams" for more information.

This parameter can be set using parameter tunneling.

Format

Integer

Value
Value range is **0–65535**.

data_coding

Description
This parameter defines the mandatory SMPP **data_coding** parameter.

Overrides the **DefaultDataCoding** configuration attribute. See "Attribute: DefaultDataCoding" for more information.

It is valid for application-initiated requests.

It is valid for network-triggered requests if the forwarding parameter is enabled. See "Attribute: ForwardXParams" for more information.

This parameter can be set using parameter tunneling.

Format
Integer

Value
Value range is **0–255**. Some values are restricted. See the SMPP specification for details.

esm_class

Description
This parameter defines the mandatory SMPP **esm_class** parameter.

It is valid for application-initiated requests only.

This parameter can be set using parameter tunneling.

Format
Integer

Value
Value range is **0–255**. Some values are restricted. See the SMPP specification for details.

sms.service.type

Description
This parameter defines the mandatory SMPP **service_type** parameter.

It is valid for application-initiated requests only.

This parameter can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

This parameter name can be configured in the **wlng.sms.service.type** system property. The default is **sms.service.type**.

Format
String enumeration

Value
Valid values are **CMT**, **CPT**, **VMN**, **VMA**, **WAP**, **USSD**, and an empty string (**""**). See the SMPP specification for details.

sms.replace.if.present

Description
This parameter defines the mandatory SMPP **replace_if_present_flag** parameter.

It is valid for application-initiated requests only.

This parameter can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

This parameter key name can be configured in the wlng.sms.replace.if.present system property. The default is "sms.replace.if.present".

Format
Integer

Value
Value values are **0** and **1**. See the SMPP specification for details.

com.bea.wlcp.wlng.plugin.sms.OriginatingAddressType

Description
This parameter defines a mapping ID.

The ID is used for looking up an SMPP ESME Type Of Number (TON) and an SMPP ESME Numbering Plan Indicator (NPI). The TON and NPI are configured using OAM.

This parameter is valid for application-initiated requests only.

This parameter can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

com.bea.wlcp.wlng.plugin.sms.DestinationAddressType.*n*

Description
This parameter defines a mapping ID.

The ID is used for looking up an SMPP ESME Type Of Number (TON) and an SMPP ESME Numbering Plan Indicator (NPI). The TON and NPI are configured using OAM.

The *n* is the number of the destination address. Valid values are 0 to one less than the number of destination addresses. An example of this parameter name would be:

com.bea.wlcp.wlng.plugin.sms.DestinationAddressType.2

This parameter is valid for application-initiated requests only.

This parameter can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

com.bea.wlcp.wlng.plugin.sms.RequestDeliveryReportFlag

Description
This parameter defines the mandatory SMPP **registered_delivery** parameter.

It specifies whether delivery reports are requested for application-initiated requests.

It is valid for application-initiated requests only.

This parameter can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, delivery reports are requested and the SMPP **registered_delivery** parameter is set to 0x01.

If `false`, delivery reports are not requested and the SMPP **registered_delivery** parameter is set to 0x00.

com.bea.wlcp.wlng.plugin.sms.DataCoding

Description
This parameter defines the mandatory SMPP **data_coding** parameter.

The plug-in uses it for encoding the message string.

It is valid for application-initiated requests only.

This parameter can be set using SLAs.

Format
Integer

Value
Value range is **0–255**. Some values are restricted. See the SMPP specification for details.

com.bea.wlcp.wlng.plugin.sms.Priority

Description
This parameter defines the mandatory SMPP **priority_flag** parameter.

It is valid for application-initiated requests only.

This parameter can be set using SLAs.

Format
String

Value
Valid values are:

- HIGH; Sets **priority_flag** to **3**.
- LOW; Sets **priority_flag** to **0**.
- DEFAULT; Sets **priority_flag** to **0**.
- UNDEFINED; Sets **priority_flag** to **0**.

originating_address

Description
This parameter defines the originating address of the SMS in the delivery receipt.

When this parameter is used, the SmsMBean's Boolean **forwardXParam** attribute must be set to `true` to make the parameter appear in the SOAP header. By default, **forwardXParam** is `false`. See "Attribute: ForwardXParams" for more information.

This parameter can be set using parameter tunneling.

Format
String

smpp_billing_id

Description
This parameter defines the billing information according to the format in the SMPP Specification 5.0, section 4.8.4.3 titled "billing_identification".

The parameter works with SMPP 5.0 SMSCs, but with not with SMPP 3.4 SMSCs.

The parameter is intended for use with Parlay X 2.1 SMPP.

Format
Hexadecimal String

Table 6–3 describes the format.

Table 6–3 Format for smpp_bliing_id Value

Field	Size (octets)	Type	Description
parameter tag	2	Integer	0x060B
length	2	Integer	Length of value part in octets
value	1 - 1024	Octet String	Bits 7......0 0XXXXXXX (Reserved) 1XXXXXXX (Vendor Specific) The first octet represents the Billing Format tag and indicates the format of the billing information contained in the remaining octets.

If the value is not sent as a hexadecimal string, it is ignored and a warning is logged.

Here is sample code for encoding the string.

```
private String getHexEncodedString(String normalString) {
  byte[] bHexStr = normalString.getBytes();
  String retVal = "";
..String sOctet = null;
  for (int i = 0; i < bHexStr.length; i++) {
    sOctet = Integer.toHexString((int) (bHexStr[i] & 0xFF));
    if (sOctet.length() == 1) {
      sOctet = "0" + sOctet;
    }
    retVal = retVal.concat(sOctet);
  }
  return retVal.toUpperCase();
}
```

dest_addr_subunit

Description

This parameter defines the **dest_addr_subunit** field in the following SMPP operations:

- **SUBMIT_SM**
- **SUBMIT_MULTI**
- **DATA_SM**

It can be set using parameter tunneling.

Format

Decimal

The decimal value is automatically converted to a hexadecimal string before it is passed to the SMPP **dest_addr_subunit** field.

Example

```
<xparams> <param key="dest_addr_subunit" value="1"/> </xparams>
```

dest_bearer_type

Description

This parameter is used to request the desired bearer for delivery of the message to the destination address.

If the receiving system (the SMSC) does not support the indicated bearer type, it may return a response PDU reporting a failure.

See section 5.3.2.5 of the *Short Message Peer to Peer Protocol Specification v3.4* for the formal definition of the parameter and section 4.7.1 for its specification as an optional parameter for the **DATA_SM** operation.

This parameter can be set using parameter tunneling.

Format

Unsigned Byte [**0–255**]

Value

Valid values are:

- **0x00** = Unknown
- **0x01** = SMS
- **0x02** = Circuit Switched Data (CSD)
- **0x03** = Packet Data
- **0x04** = USSD
- **0x05** = CDPD
- **0x06** = DataTAC
- **0x07** = FLEX/ReFLEX
- **0x08** = Cell Broadcast (cellcast)
- **9** to **255** Reserved

service_type

Description

This parameter indicates the SMS application service associated with the message. Allows the ESME to use enhanced messaging services such as **replace_if_present** and to control the teleservice used on the air interface (for example, ANSI-136/TDMA and IS-95/CDMA).

It is used to support tunneling USSD (3G TS 23.090 version 3.0.0) messages through the SMPP protocol.

See section 5.2.11 of the *Short Message Peer to Peer Protocol Specification v3.4* for the formal definition of the parameter and the appropriate subsections of section 4 for its specification as a mandatory parameter for **SUBMIT_SM**, **SUBMIT_MULTI**, **DELIVER_SM**, **DATA_SM**, and **CANCEL_SM**.

This parameter can be set using parameter tunneling.

Format

Octet string

Value

The predefined generic service type value for USSD is **USSD**.

ussd_service_operation

Description

This parameter defines the USSD service operation that is required when SMPP is used as an interface to a (GSM) USSD system.

It is used to support tunneling USSD (3G TS 23.090 version 3.0.0) messages through the SMPP protocol.

It is used as an optional parameter to the SMPP **SUBMIT_SM** operation.

This parameter is defined in section 5.3.2.44 of the *Short Message Peer to Peer Protocol Specification v3.4*.

It was added to the **DELIVER_SM** operation in the SMPP 5.0 specification. See *Short Message Peer to Peer Protocol Specification Version 5.0*.

This parameter can be set using parameter tunneling.

Format

Unsigned byte [**0–255**]

Value

Valid values are:

- **0** = PSSD indication
- **1** = PSSR indication
- **2** = USSR request
- **3** = USSN request
- **4** to **15** Reserved
- **16** = PSSD response
- **17** = PSSR response

- **18** = USSR confirm
- **19** = USSN confirm
- **20** to **31** Reserved
- **32** to **255** Reserved for vendor-specific USSD operations

its_session_info

Description

This is a required parameter for the CDMA Interactive Teleservice as defined by the Korean PCS carriers [KORITS]. Contains control information for the interactive session between an MS and an ESME.

See section 5.3.2.43 of the *Short Message Peer to Peer Protocol Specification v3.4* for the formal definition of the parameter and the appropriate subsections of section 4 for its specification as an optional parameter for **SUBMIT_SM**, **DELIVER_SM**, and **DATA_ SM**.

This parameter is also supported for native SMPP.

This parameter can be set using parameter tunneling.

Format

Unsigned Short (2 bytes) [0–65535] as an octet string

Following is a description of the octet string.

Bits 7...............0

SSSS SSSS (octet 1)

NNNN NNNE (octet 2)

Octet 1 contains the session number (0–255) encoded in binary. The session number remains constant for each session.

Octet 2 encodes the sequence number of the dialog unit (as assigned by the ESME) within the session in bits [7. . . 1] .

Bit 0 of octet 2 is the End of Session Indicator, indicating that the message is the end of the conversation session. Valid values for the End of Session Indicator are:

- **0** = End of Session Indicator inactive
- **1** = End of Session Indicator active

Managing Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP

This section describes the properties and workflow for setting up Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP network protocol plug-in instances. These plug-in instances share the same configuration options.

The Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP communication services rely on an SMPP Server Service for connecting to the Small Message Service Center (SMSC).

The SMPP Server Service is also used by the Native SMPP Communication Service. See Chapter 20 for information on managing client connections using SMPP Server Service.

Configuration facilities for the SMPP Server Service are described in detail in the following sections of Chapter 20, "Native SMPP."

- Properties for SMPP Server Service
- Reference: Attributes and Operations for SMPP Server Service

Properties for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP

Table 6–4 lists the technical specifications for the communication service.

Table 6–4 Properties for Parlay X 2.1 Short Messaging/SMPP and EWS Binary SMS/SMPP

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plugin_instance_id*

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=same as the network protocol *instance_id* assigned when the plug-in instance is created
	Type=oracle.ocsg.sms.smpp.management.SmsMBean

Network protocol plug-in service ID	Plugin_px21_short_messaging_smpp
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
Supported Address Scheme	tel
Service type	Sms
Exposes to the service communication layer a Java representation of:	Parlay X 2.1 Short Messaging/SMPP: ▪ Parlay X 2.1 Part 4: Short Messaging Extended Web Services Binary SMS/SMPP: ▪ Extended Web Services Binary SMS
Interfaces with the network nodes using:	SMPP 3.4
Deployment artifacts	wlng_nt_sms_px21.ear and wlng_at_sms_px21.ear

Configuration Workflow for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID as listed in the "Properties for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP" section.
2. Using the Administration Console or an MBean browser, select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID given when the plug-in instance was created.

3. Configure the behavior of the plug-in instance. See "Reference: Attributes and Operations for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP" for the list of attributes and operations.
4. If the plug-in uses short code mappings, configure the Short Code Mapper. For more information. See "Managing and Configuring Shortcode Mappings" in *System Administrator's Guide*.
5. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP" section.
6. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
7. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Management Operations in the SMPP Server Service

The Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP communication services use the SMPP Server Service to establish and manage client connections with the SMSC.

The SMPP Server Service establishes a client connection to the SMSC when the plug-in instance is activated or when the administrator resets the client connection by using the **resetClientConnection** SMPP Server Service operation.

The following Server Service operations, described in Chapter 20, "Native SMPP," are used to manage client connections:

- Operation: closeClientConnection
- Operation: listClientConnections
- Operation: listPluginInstances
- Operation: resetClientConnection

Reference: Attributes and Operations for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP

This section describes the attributes and operations for configuration and maintenance:

- Attribute: ActiveStatus (read-only)
- Attribute: BindType
- Attribute: DataSm
- Attribute: DefaultDataCoding
- Attribute: DeliverSmRespCommandStatus
- Attribute: DestinationAddressNpi
- Attribute: DestinationAddressTon
- Attribute: EnquireLinkTimerValue
- Attribute: EsmeAddressRange

- Operation: addOriginatingAddressTypeMapping
- Operation: addDestinationAddressTypeMapping
- Operation: countOfflineNotificationCache
- Operation: countOnlineNotificationCache
- Operation: countSmsCache
- Operation: enableReceiveSms
- Operation: getOfflineNotificationInfo
- Operation: getOnlineNotificationInfo
- Operation: listOnlineBinaryNotificationInfo
- Operation: getOfflineNotificationInfo
- Operation: listOnlineNotificationInfo
- Operation: listOriginatingAddressTypeMappings
- Operation: listDestinationAddressTypeMappings
- Operation: removeOfflineNotificationInfo
- Operation: removeOnlineNotificationInfo
- Operation: removeOriginatingAddressTypeMapping
- Operation: removeDestinationAddressTypeMapping
- Operation: startSmsNotification
- Operation: translateDestinationAddressNpi
- Operation: translateDestinationAddressTon
- Operation: translateOriginatingAddressNpi
- Operation: translateOriginatingAddressTon

Attribute: ActiveStatus (read-only)

Scope: Server

Unit: Not applicable

Format: Boolean

Read-only attribute reporting the active status of the plug-in:

- `true` if the plug-in is currently active
- `false` if the plug-in is currently inactive

Attribute: BindType

Scope: Server

Unit: Not applicable

Format: Integer

Specifies how the plug-in binds to the SMSC

Use:

- **0** to bind as transmitter and receiver

- **1** to bind as transceiver
- **2** to bind as transmitter only
- **3** to bind as receiver only

The default is **0**.

Attribute: DataSm

Scope: Server

Unit: Not applicable

Format: Boolean

Set to `true` to use the **DATA_SM** operation when sending binary data.

The default is `false`.

Attribute: DefaultDataCoding

Scope: Server

Unit: Not applicable

Format: Integer

Specifies the default data coding to use when sending SMS messages. This value will be used if a data coding is not provided by the application-facing interface.

See the **data_coding** parameter in the SMPP specification for valid values.

Use:

- **0** for SMSC Default Alphabet
- **1** for ASCII
- **8** for USC2

If this attribute is set to 0, the message is 7-bit encoded with a maximum of 160 characters in a single SMS message. Messages greater than 160 characters are split.

If this attribute is set to 1, the message is 8-bit encoded with a maximum of 140 characters in a single SMS message. Messages greater than 140 characters are split.

The default is **0**.

Attribute: DeliverSmRespCommandStatus

Scope: Server

Unit: Not applicable

Format: Integer

Specifies the command status of the **DELIVER_SM_RESP** operation to send to the SMSC if a delivery notification or mobile-originated message can not be delivered to the requesting application and there is no matching offline notification.

Attribute: DestinationAddressNpi

Scope: Server

Unit: Not applicable

Format: Integer

ESME Numbering Plan Indicator (NPI). Used as a default for the destination address.

Use:

- **0** for Unknown
- **1** for ISDN (E163/E164)
- **3** for Data (X.121)
- **4** for Telex (F.69)
- **6** for Land Mobile (E.212)
- **8** for National
- **9** for Private
- **10** for ERMES
- **14** for Internet (IP)
- **18** for WAP Client ID

Attribute: DestinationAddressTon

Scope: Server

Unit: Not applicable

Format: Integer

ESME Type Of Number (TON). Used as a default for the destination address.

Use:

- **0** for Unknown
- **1** for International
- **2** for National
- **3** for Network
- **4** for Subscriber
- **5** for Alphanumeric
- **6** for Abbreviated
- **7** Reserved

Attribute: EnquireLinkTimerValue

Scope: Server

Unit: Minutes

Format: Integer

Minimum interval between **ENQUIRE_LINK** requests to the SMSC.

The default is 60 seconds.

The plug-in instance performs **ENQUIRE_LINK** requests (heartbeats) to the SMSC to verify that the connection is alive.

The setting is applied as follows:

- If the plug-in has received traffic subsequent to the last scheduled time, no ENQUIRE_LINK request is made and a new timer (**EnquireLinkTimerValue**) is scheduled.
- If no response is received, the plug-in unbinds and attempts to re-bind.
- If the plug-in has outstanding requests that prevent it from sending **ENQUIRE_LINK** requests, it unbinds. This typically occurs if the SMSC is unresponsive while the plug-in is filling the window with unacknowledged **SUBMIT_SM** requests.

To turn off sending **ENQUIRE_LINK** requests, set the **EnquireLinkTimerValue** to **0**.

Attribute: EsmeAddressRange

Scope: Server

Unit: Not applicable

Format: String formatted as a regular expression.

Address range of the SMS messages to be sent to the plug-in instance by the SMSC. The address range is specified as a UNIX regular expression.

Attribute: EsmeNpi

Scope: Server

Unit: Not applicable

Format: Integer

ESME Numbering Plan Indicator (NPI).

Used for the destination address and as a default for the originating address. Also used for both destination address and originating address during the **BIND** operation.

Use:

- **0** for Unknown
- **1** for ISDN (E163/E164)
- **3** for Data (X.121)
- **4** for Telex (F.69)
- **6** for Land Mobile (E.212)
- **8** for National
- **9** for Private
- **10** for ERMES
- **14** for Internet (IP)
- **18** for WAP Client ID

Attribute: EsmePassword

Scope: Server

Unit: Not applicable

Format: String

Password used by the plug-in instance when connecting to the SMSC as an ESME.

Attribute: EsmeSystemId

Scope: Server

Unit: Not applicable

Format: String

System ID used by the plug-in instance when connecting to the SMSC as an ESME.

Attribute: EsmeSystemType

Scope: Server

Unit: Not applicable

Format: String

System type used by the plug-in instance when connecting to the SMSC as an ESME.

The default is **mess_gateway**.

Attribute: EsmeTon

Scope: Server

Unit: Not applicable

Format: Integer

ESME Type Of Number (TON).

Used for destination address and as a default for originating address. Also used for both destination address and originating address during the **BIND** operation. Use:

- **0** for Unknown
- **1** for International
- **2** for National
- **3** for Network
- **4** for Subscriber
- **5** for Alphanumeric
- **6** for Abbreviated
- **7** Reserved

Attribute: ForwardXParams

Scope: Server

Unit: Not applicable

Format: Boolean

Specifies if tunneled parameters are forwarded to applications for network-triggered requests.

Use:

- `true` to enable forwarding.
- `false` to disable forwarding.

The default is `false`.

Attribute: LocalAddress

Scope: Server

Unit: Not applicable

Format: String

Local server address used by the plug-in to connect to the SMSC. The address can be expressed as an IP address or host name. The address or host name must resolve to a local address.

The default is **""**.

Attribute: LocalPort

Scope: Server

Unit: Not applicable

Format: Integer [**1–65535**]

Local port used by the plug-in to connect to the SMSC.

The default is **3000**.

Attribute: MaxKeywordLimit

Scope: Server

Unit: Not applicable

Format: Integer

Maximum number of keywords that can be used to match in a short message to determine whether the application receives a notification.

The text is matched in two steps:

1. The entire string is compared against the incoming short message for an exact match.
2. If no match is found, the plug-in matches the short message one word at a time against the criteria string, up to the number set in the this attribute.

Attribute: MessageIdInHexFormat

Scope: Cluster

Unit: Not applicable

Format: Boolean

Describes the format of the **message_id** in **SUBMIT_SM_RESP**, **SUBMIT_MULTI_RESP**, and **DATA_SM_RESP** operations.

If `true`, the **message_id** is in hexadecimal format; if `false`, it is in decimal format.

The default is `false`.

Attribute: MessagingMode

Scope: Server

Unit: Not applicable

Format: Integer

ESM_CLASS Messaging Mode for packets.

- **2** for Forward mode. Used for the **DATA_SM** operation.
- **3** for Store and Forward mode.

Attribute: MobileCountryCode

Scope: Server

Unit: Not applicable

Format: Integer

Mobile country code for sending operator logos.

Attribute: MobileNetworkCode

Scope: Server

Unit: Not applicable

Format: Integer

Mobile network code for sending operator logos.

Attribute: ModuleId (read-only)

Scope: Server

Unit: Not applicable

Format: String

Read-only module identifier assigned to the plug-in instance when it is created.

Attribute: NumberReceiverConnections

Scope: Server

Unit: Not applicable

Format: Integer

Number of receiver connections used to connect to the SMSC. Used when the bind type is 0 or 3. See "Attribute: BindType".

The connections are established when the plug-in instance is activated.

The default is **1**.

Attribute: NumberTransceiverConnections

Scope: Server

Unit: Not applicable

Format: Integer

Number of transceiver connections used to connect to the SMSC. Used when the bind type is **1**. See "Attribute: BindType" for more information.

The connections are established when the plug-in instance is activated.

The default is **1**.

Attribute: NumberTransmitterConnections

Scope: Server

Unit: Not applicable

Format: Integer

Number of transmitter connections used to connect to the SMSC. Used when the bind type is **0** or **2**. See "Attribute: BindType" for more information.

The connections are established when the plug-in instance is activated.

The default is **1**.

Attribute: OriginatingAddressNpi

Scope: Server

Unit: Not applicable

Format: Integer

ESME Numbering Plan Indicator (NPI).

Used as a default for the originating address.

Use:

- **0** for Unknown
- **1** for ISDN (E163/E164)
- **3** for Data (X.121)
- **4** for Telex (F.69)
- **6** for Land Mobile (E.212)
- **8** for National
- **9** for Private
- **10** for ERMES
- **14** for Internet (IP)
- **18** for WAP Client ID

Attribute: OriginatingAddressTon

Scope: Server

Unit: Not applicable

Format: Integer

ESME Type Of Number (TON).

Used as a default for originating address.

Use:

- **0** for Unknown
- **1** for International

- **2** for National
- **3** for Network
- **4** for Subscriber
- **5** for Alphanumeric
- **6** for Abbreviated
- **7** Reserved

Attribute: ReceiveSegmentsWaitTime

Scope: Server

Unit: Seconds

Format: Integer

Maximum time to wait for the arrival of the subsequent segments of a concatenated short message from the SMSC after the arrival of the first segment.

Attribute: ReceiveSmsIgnoreMissingSegments

Scope: Server

Unit: Not applicable

Format: Boolean

Specifies if the plug-in instance will deliver network-triggered short messages with missing message segments to applications.

Use:

- `true` if the plug-in assembles received segments and delivers the incomplete message to the application.
- `false` if the plug-in does not deliver messages to the application unless all segments are received.

Attribute: RequestDeliveryReports

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if the default behavior of the plug-in instance is to request delivery reports.

Use:

- `true` if delivery reports should be requested.
- `false` if delivery reports should not be requested.

The default is `true`.

If delivery requests are not requested, applications will, by default, not have the ability to poll for delivery status. However, it is possible to override the default setting in the service provider SLA or in an application SLA.

Attribute: RequestTimerValue

Scope: Server

Unit: Seconds

Format: Integer

Maximum time between the submission of a request to the SMSC and the receipt of the corresponding response, before the connection is terminated.

The default is **20** seconds.

Attribute: RetryTimesBeforeGiveUp

Scope: Server

Unit: Not applicable

Format: Integer

Maximum number of times for the plug-in to try to reconnect to the SMPP Server Service.

The default is **30**.

Attribute: RetryTimesBeforeReconnect

Scope: Server

Unit: Not applicable

Format: Integer

Maximum number of times for the plug-in to try to connect to the SMPP Server Service before attempting to reconnect.

The default is **3**.

Attribute: SMSCDefaultAlphabet

Scope: Server

Unit: Not applicable

Format: Integer

SMSC default alphabet. This is the default character encoding scheme used by the SMSC when encoding short messages. The plug-in instance needs to use the same character encoding scheme for the characters to be decoded correctly. All encoding schemes supported by Java are possible.

Use:

- **ASCII** for American Standard Code for Information Interchange.
- **Cp1252** for Windows Latin-1.
- **ISO8859_1** for ISO 8859-1, Latin alphabet No. 1.
- **GSM_DEFAULT** for the default GSM character set.

If the default character set used by the SMSC is GSM 7- bit, set this attribute to **GSM_DEFAULT**.

If the SMSC has a limit of 160 characters per SMS message, set the **wlng.smpp.max_payload_size** system property to **140**.

Attribute: SegmentsLimit

Scope: Server

Unit: Not applicable

Format: Integer

Maximum number of SMPP segments an application is allowed to send when using the Extended Web Services Binary SMS interface.

Attribute: SequenceNumberRangeEndId

Scope: Server

Unit: Not applicable

Format: Integer

End ID of the sequence number range. Sequential numbers generated for plug-in instances cannot exceed this value.

Attribute: SequenceNumberRangeStartId

Scope: Server

Unit: Not applicable

Format: Integer

Start ID of the sequence number range. Sequential numbers for plug-in instances begin with this value.

Attribute: SmppVersion

Scope: Server

Unit: Not applicable

Format: String

Version of the SMPP protocol used by the plug-in.

Attribute: SmscAddress

Scope: Server

Unit: Not applicable

Format: String

SMSC address as an IP-address or a host name.

The setting is not applied until the plug-in service is restarted or "Operation: resetClientConnection" is performed.

Attribute: SmscPort

Scope: Server

Unit: Not applicable

Format: Integer

Port used by the SMSC.

The setting is applied until the plug-in service is restarted or the **resetClientConnection** operation is performed in the SMPP Server Service. See "Operation: resetClientConnection" for more information.

Attribute: UseMessagePayload

Scope: Server

Unit: Not applicable

Format: Boolean

If `true`, the message is carried in the optional **message_payload** field in the SMPP PDU.

Current behavior predicates that only segmented messages such as ringtones, logos, and long SMS messages that have a UDH are carried in the **message_payload** field.

If this attribute is `false`, both segmented and unsegmented messages are sent in the **short_message** field.

The short message data can be inserted in either the **short_message** or **message_payload** fields, but not both simultaneously.

The default is `true`.

Attribute: UserTextMaxLength

Scope: Server

Unit: Not applicable

Format: Integer

Maximum number of characters allowed in a short message.

The default is **1600**.

Attribute: WindowingMaxQueueSize

Scope: Server

Unit: Not applicable

Format: Integer

Maximum number of mobile-terminated requests to the SMSC allowed in the windowing queue.

The default is **100**.

If any one of the three windowing attributes (**WindowingMaxQueueSize**, **WindowingMaxWaitTime**, or **WindowingSize**) is set to a value less than zero, windowing is turned off. If all of these three attributes have values greater than zero, windowing is turned on.

See "Windowing" for general information about windowing.

Attribute: WindowingMaxWaitTime

Scope: Server

Unit: Seconds

Format: Integer

Maximum time that a mobile-terminated request to the SMSC is allowed to wait in the windowing queue.

Valid only when the **WindowingSize** is enforced. See "Attribute: WindowingSize" for more information.

The default is **15** seconds.

If any one of the three windowing attributes (**WindowingMaxQueueSize**, **WindowingMaxWaitTime**, or **WindowingSize**) is set to a value less than zero, windowing is turned off. If all of these three attributes have values greater than zero, windowing is turned on.

See "Windowing" for general information about windowing.

Attribute: WindowingSize

Scope: Server

Unit: Not applicable

Format: Integer

Maximum number of simultaneous unacknowledged mobile-terminated requests to the SMSC enforced for each connection.

This setting applies only to requests sent from the plug-in to the SMSC, not to requests from the SMSC to the plug-in.

A value of -1 indicates that the number of unacknowledged operations is not restricted. Other valid values must be greater than 0 (zero).

The default is **5**.

If any one of the three windowing attributes (**WindowingMaxQueueSize**, **WindowingMaxWaitTime**, or **WindowingSize**) is set to a value less than zero, windowing is turned off. If all of these three attributes have values greater than zero, windowing is turned on.

See "Windowing" for general information about windowing.

Operation: addOriginatingAddressTypeMapping

Scope: Cluster

If the tunneled parameter **com.bea.wlcp.wlng.plugin.sms.OriginatingAddressType** is available in a request, the value of the parameter is extracted and matched against the originating address type mapping list. The matching is with the type parameter.

Signature:

```
addOriginatingAddressTypeMapping(type: String, ton: int, npi: int)
```

Table 6–5 addOriginatingAddressTypeMapping Parameters

Parameter	Description
type	Specifies the originating address type to be mapped.

Table 6–5 (Cont.) addOriginatingAddressTypeMapping Parameters

Parameter	Description
ton	Specifies the ESME Type Of Number (TON). Use: ■ **0** for Unknown ■ **1** for International ■ **2** for National ■ **3** for Network ■ **4** for Subscriber ■ **5** for Alphanumeric ■ **6** for Abbreviated
npi	Specifies the ESME Numbering Plan Indicator (NPI). Use: ■ **0** for Unknown ■ **1** for ISDN (E163/E164) ■ **3** for Data (X.121) ■ **4** for Telex (F.69) ■ **6** for Land Mobile (E.212) ■ **8** for National ■ **9** for Private ■ **10** for ERMES ■ **14** for Internet (IP) ■ **18** for WAP Client ID

Operation: addDestinationAddressTypeMapping

Scope: Cluster

If the tunneled parameter **com.bea.wlcp.wlng.plugin.sms.DestinationAddressType** is available in a request, the value of the parameter is extracted and matched against the destination address type mapping list. The matching is with the type parameter.

Signature:

```
addDestinationAddressTypeMapping(type: String, ton: int, npi: int)
```

Table 6–6 addDestinationAddressTypeMapping Parameters

Parameter	Description
type	Specifies the destination address type to be mapped.
ton	Specifies the ESME Type Of Number (TON). Use: ■ **0** for Unknown ■ **1** for International ■ **2** for National ■ **3** for Network ■ **4** for Subscriber ■ **5** for Alphanumeric ■ **6** for Abbreviated

Table 6–6 (Cont.) addDestinationAddressTypeMapping Parameters

Parameter	Description
npi	Specifies the ESME Numbering Plan Indicator (NPI). Use: ▪ **0** for Unknown ▪ **1** for ISDN (E163/E164) ▪ **3** for Data (X.121) ▪ **4** for Telex (F.69) ▪ **6** for Land Mobile (E.212) ▪ **8** for National ▪ **9** for Private ▪ **10** for ERMES ▪ **14** for Internet (IP) ▪ **18** for WAP Client ID

Operation: countOfflineNotificationCache

Scope: Cluster

Displays the number of entries in the offline notification cache.

Signature:

```
countOfflineNotificationCache()
```

Operation: countOnlineNotificationCache

Scope: Cluster

Displays the number of entries in the online notification cache.

Signature:

```
countOnlineNotificationCache()
```

Operation: countSmsCache

Scope: Cluster

Displays the sum of short messages in the cache for mobile-originated messages and mobile-terminated short messages. There are separate caches (stores) for mobile-originated and mobile-terminated short messages. This method returns the sum of both caches.

Signature:

```
countSmsCache()
```

Operation: enableReceiveSms

Scope: Cluster

Adds an offline notification for applications that poll for mobile-originated short messages. Those mobile-originated short messages that match the criteria will not result in a notification callback to an application. Instead the message is stored in Services Gatekeeper. The application must use the correlator returned by this method to poll for short messages.

Signature:

```
enableReceiveSms(shortcode: String, criteria: String, appInstanceID: String)
```

Table 6–7 enableReceiveSms Parameters

Parameter	Description
shortcode	Destination address of the short message. Prefixed with the URI; for example, **tel:**
criteria	Text to match against to determine if the application should receive the notification.
appInstanceID	ID of the application instance associated with the notification.

Operation: getOfflineNotificationInfo

Scope: Cluster

Displays information about a notification registered offline. See "Operation: enableReceiveSms" for more information.

Signature:

```
getOfflineNotificationInfo(correlator: String)
```

Table 6–8 getOfflineNotificationInfo Parameters

Parameter	Description
correlator	Correlator identifying the notification.

Operation: getOnlineNotificationInfo

Scope: Cluster

Displays information about a notification registered online by an application.

Signature:

```
getOnlineNotificationInfo(correlator: String)
```

Table 6–9 getOnlineNotificationInfo Parameters

Parameter	Description
correlator	Correlator identifying the notification.

Operation: listDestinationAddressTypeMappings

Scope: Cluster

Displays a list of all destination address type mappings.

Signature:

```
listDestinationAddressTypeMappings()
```

Operation: listOnlineBinaryNotificationInfo

Scope: Cluster

Lists all online notifications for binary SMS messages.

These are notifications added using **StartBinarySmsNotification**.

Operation: listOfflineNotificationInfo

Scope: Cluster

Displays a list of all notifications registered offline.

Signature:

```
listOfflineNotificationInfo()
```

Operation: listOnlineNotificationInfo

Scope: Cluster

Displays a list of all notifications registered by an application.

Signature:

```
listOnlineNotificationInfo()
```

Operation: listOriginatingAddressTypeMappings

Scope: Cluster

Displays a list of all originating address type mappings.

Signature:

```
listOriginatingAddressTypeMappings()
```

Operation: removeOfflineNotificationInfo

Scope: Cluster

Removes a notification registered off-line.

Signature:

```
removeOfflineNotificationInfo(correlator: String)
```

Table 6–10 removeOfflineNotificationInfo Parameters

Parameter	Description
correlator	Correlator identifying the notification.

Operation: removeOnlineNotificationInfo

Scope: Cluster

Removes a notification registered by an application.

Signature:

```
removeOnlineNotificationInfo(correlator: String)
```

Table 6–11 removeOnlineNotificationInfo Parameters

Parameter	Description
correlator	Correlator identifying the notification.

Operation: removeOriginatingAddressTypeMapping

Scope: Cluster

Removes an existing TON or NPI address type mapping for a specified originating address type.

Signature:

```
removeOriginatingAddressTypeMapping(type: String)
```

Table 6–12 removeOriginatingAddressTypeMapping Parameters

Parameter	Description
type	Originating address type for the mapping. See "Operation: addOriginatingAddressTypeMapping" and "Operation: listOriginatingAddressTypeMappings" for more information.

Operation: removeDestinationAddressTypeMapping

Scope: Cluster

Removes an existing TON or NPI address type mapping for a specified destination address type.

Signature:

```
removeDestinationAddressTypeMapping(type: String)
```

Table 6–13 removeDestinationAddressTypeMapping Parameters

Parameter	Description
type	Destination address type for the mapping. See "Operation: addDestinationAddressTypeMapping" and "Operation: listDestinationAddressTypeMappings" for more information.

Operation: startSmsNotification

Scope: Cluster

Registers a notification for mobile-originated short messages on behalf of an application. Has the same result as if the application used the **startSmsNotification** operation in the Parlay X 2.1 SmsNotificationManager interface.

The text in the **criteria** parameter is matched in two steps to determine the target application:

1. The entire string is compared with the incoming short message for an exact match.
2. If an exact match is not found, the message is matched one word at a time from left to right up to the number of words set by the **MaxKeywordLimit** attribute. See "Attribute: MaxKeywordLimit" for more information.

Services Gatekeeper rejects overlapping criteria. For example, if the text "FUNNY JOKE" and "FUNNY JOKE 5439" are both specified as the criteria text for the same shortcode, an exception will be raised when the notification is started.

An exact match takes precedence over a partial match. For example, if Application1 sets its criteria to "FUNNY JOKE" and Application2 sets its criteria to "JOKE", both for the same short code, a message with the content "FUNNY JOKE" will trigger a notification to Application1 but not to Application2.

Signature:

```
startSmsNotification(endpoint: String, shortcode: String, criteria: String,
appInstanceID: String)
```

Table 6–14 startSmsNotification Parameters

Parameter	Description
endpoint	Notification endpoint implemented by the application. This endpoint implements the Parlay X 2.1 SmsNotification interface. Format: URL
shortcode	Destination address or service activation number for the short message.
criteria	Text in the payload of the short message to match to determine if the application receives the notification.
appInstanceID	ID of the application instance associated with this notification.

Operation: translateDestinationAddressNpi

Scope: Cluster

Gets the ESME Numbering Plan Indicator (NPI) of the destination address mapping added for the specified type.

Signature:

```
translateDestinationAddressNpi(type: String)
```

Table 6–15 translateDestinationAddressNpi Parameters

Parameter	Description
type	Type for used for the mapping. See "Operation: addOriginatingAddressTypeMapping" for more information.

Operation: translateDestinationAddressTon

Scope: Cluster

Gets the ESME Type of Number (TON) of the destination address mapping added for the specified type.

Signature:

```
translateDestinationAddressTon(type: String)
```

Table 6–16 translateDestinationAddressTon Parameters

Parameter	Description
type	Type for used for the mapping. See "Operation: addDestinationAddressTypeMapping" for more information.

Operation: translateOriginatingAddressNpi

Scope: Cluster

Gets the ESME Numbering Plan Indicator (NPI) of the originating address mapping added for the specified type.

Signature:

```
translateOriginalAddressNpi(type: String)
```

Table 6–17 translateOriginatingAddressNpi Parameters

Parameter	Description
type	Type for used for the mapping. See "Operation: addOriginatingAddressTypeMapping" for more information.

Operation: translateOriginatingAddressTon

Scope: Cluster

Gets the ESME Type of Number (TON) of the originating address mapping added for the specified type.

Signature:

```
translateOriginalAddressTon(type: String)
```

Table 6–18 translateOriginatingAddressTon Parameters

Parameter	Description
type	Type for used for the mapping. See "Operation: addOriginatingAddressTypeMapping"for more information.

7

Parlay X 2.1 Terminal Location/MLP

This chapter describes the Parlay X 2.1 Terminal Location/Mobile Location Protocol (MLP) communication service in detail.

Overview of the Parlay X 2.1 Terminal Location/MLP Communication Service

The Parlay X 2.1 Terminal Location/MLP communication service exposes the Parlay X 2.1 Terminal Location application interfaces.

The communication service acts as an LCS -MLS client to a location server using MLP over HTTP.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Services Gatekeeper connects to the location server using HTTP. It always acts as a single LCS/MLS client to the location server. It can be configured to act in Standard Location or Emergency Location Immediate mode on the node level.

Using the Terminal Location communication service, an application can:

- Ask for the location of one or many terminals by polling.
- Ask for the distance between a specified terminal and a specified position.
- Sign up to be notified when a terminal enters or leaves a specified geographical area.
- Receive notifications when the terminal enters or leaves the specified geographical area.
- Sign up to be notified periodically about the location of a terminal.
- Receive periodic location notifications about the location of a terminal.

The application can specify a number of parameters concerning the nature of the notification. These include:

- Requested accuracy
- Accepted accuracy
- Accepted response time
- Maximum age of location data
- Tolerance, which expresses the priority of response time versus accuracy

- Minimum frequency of notifications
- Duration of notifications
- Maximum number of notifications

The nature of the information available as well as the accuracy of the location provided, the response times, and the frequency of notification are all dependent on the specifics of the protocol and network node used. Not all networks or protocols support all operations.

Processing Direct Queries/Application-initiated Requests

If an application directly queries Services Gatekeeper for the location of a terminal or group of terminals, Services Gatekeeper sends the request to the network node. The location information is sent back synchronously in the response to the request.

Processing Notifications/Network-triggered Requests

If an application registers for periodic or geographically-defined notifications, information for the application (which may or may not include the location data for one or more terminals) arrives at Services Gatekeeper from the network. The notification is passed on to the application. If the application acknowledges the reception of the notification, Services Gatekeeper acknowledges the reception of the notification to the network. If the application does not acknowledge the reception of the notification, Services Gatekeeper does not acknowledge the reception of the notification to the network.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 Terminal Location/MLP communication service, see the discussion of Parlay X 2.1 Interfaces in *Application Developer's Guide*.

For information about the RESTful Audio Call interface, see the discussion of Terminal Location in *RESTful Application Developer's Guide*.

The RESTful Service Short Messaging interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs, reading CDRs, and so on, they are the same.

Events and Statistics

The Parlay X 2.1 Terminal Location/MLP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 7–1 lists IDs of the EDRs created by the Parlay X 2.1 Terminal Location/MLP communication service.

Table 7–1 Event Types Generated by Parlay X 2.1 Terminal Location/MLP

EDR ID	Method Called
9001	getLocation

Table 7–1 (Cont.) Event Types Generated by Parlay X 2.1 Terminal Location/MLP

EDR ID	Method Called
9002	getTerminalDistance
9003	getLocationForGroup
9004	sendLocationRequest
9011	LocationEnd
9012	LocationError
9013	LocationNotification

Charging Data Records

Terminal Location/MLP - specific CDRs occur under the following conditions:

- When the response to a polling request (of whatever type) is successfully delivered to the application.
- When a notification is received from the network.
- When an error occurs.

Statistics

Table 7–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Method names for network-initiated requests are specified by the internal Services Gatekeeper name, which is not necessarily the same as the message from the network.

Table 7–2 Transaction Types for Parlay X 2.1 Terminal Location/MLP

Method	Transaction type
getLocation	TRANSACTION_TYPE_USER_LOCATION
getLocationForGroup	TRANSACTION_TYPE_USER_LOCATION
getLocationDistance	TRANSACTION_TYPE_USER_LOCATION
locationNotification	TRANSACTION_TYPE_USER_LOCATION

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Tunneled Parameters for Parlay X 2.1 Terminal Location /MLP

This section lists the parameters that can be tunneled.

terminal_location.name_area

Description

Defines the MLP `<name_area>` element.

Complements the existing Parlay X 2.1 functionality for **startGeographicalNotification** with support for named areas. The **startGeographicalNotification** operation is defined in theTerminalLocationNotificationManager interface.

Valid for application-initiated requests only.

When this parameter key is used to define the area, the latitude, longitude, and radius parameters provided by an application are not used.

Can be set using parameter tunneling

Format
String

Example
```
<param key=" terminal_location.name_area" value="Sausalito" />
```

com.wlcp.wlng.terminal_location.start_time / com.wlcp.wlng.terminal_location.stop_time

Description
Defines the `<start_time>` and `<stop_time>` MLP elements.

Complements the existing Parlay X 2.1 functionality for **startGeographicalNotification** with support for explicit start and stop times.

Valid for application-initiated requests only.

The **utc_off** attribute is used in both elements.

Format
String

The time is expressed in UTC format: yyyy-MM-ddThh:mm:ss[+|-]HH:MM using the definitions in Table 7–3.

Table 7–3 Format for terminal_location.start_time and stop_time Values

Element	Meaning
yyyy	year
MM	month
dd	day
T	a constant
hh	hours
mm	minutes
ss	seconds
[+|-]	Positive or negative GMT offset. Use either + or -.
HH	GMT offset in hours
MM	GMT offset in minutes

Example
```
<param key=" com.wlcp.wlng.terminal_location.start_time"
value="2008-09-26T16:15:00T+08:00" />
<param key=" com.wlcp.wlng.terminal_location. stop _time"
value="2008-09-26T19:15:00T+08:00" />
```

terminal_location.polygon.point.*n*

Description

This parameter represents a set of keys in which the prefix is **terminal_location.polygon.point.** and the suffix *n* is a number in the range of 1–15.

Defines the MLP `<x>` and `<y>` elements within the `<coord>` element. The `<coord>` elements are contained by the `<LinearRing>`. The `<LinearRing>` element is contained by the `<outerBoundaryIs>` and `<polygon>` elements.

Can be set using parameter tunneling.

Format

Each **terminal_location.polygon.point** key value is expressed in the format *x*:*y* where:

- *x* corresponds to the `<x>` element
- *y* corresponds to the `<y>` element

x and *y* are expressed as decimal degrees.

Example

```
<xparams>
  <param key="com.wlcp.wlng.terminal_location.polygon.point.1"
value="6.999:43.564" />
  <param key="com.wlcp.wlng.terminal_location.polygon.point.2"
value="7.027:43.564" />
  <param key="com.wlcp.wlng.terminal_location.polygon.point.3"
value="7.027:43.564" />
  <param key="com.wlcp.wlng.terminal_location.polygon.point.4"
value="6.999:43.564" />
</xparams>
```

Managing Parlay X 2.1 Terminal Location/MLP

This section describes the properties and workflow for the Parlay X 2.1 Terminal Location/MLP plug-in instance.

Properties for Parlay X 2.1 Terminal Location/MLP

Table 7–4 lists the technical specifications for the communication service.

Table 7–4 Properties for Parlay X 2.1 Terminal Location/MLP

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plugin_instance_id*

MBean	Domain=com.bea.wlcp.wlng
	
	Name=wlng_nt
	
	InstanceName=same as the network protocol *instance_id* assigned when the plug-in instance is created
	
	Type=com.bea.wlcp.wlng.plugin.terminallocation.mlp.management.TerminalLocationMLPMBean

Network protocol plug-in service ID	Plugin_px21_terminal_location_mlp

Table 7–4 (Cont.) Properties for Parlay X 2.1 Terminal Location/MLP

Property	Description
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
Supported Address Scheme	tel
Application-facing interfaces	com.bea.wlcp.wlng.px21.plugin.TerminalLocationPlugin com.bea.wlcp.wlng.px21.plugin.TerminalLocationNotificationManagerPlugin com.bea.wlcp.wlng.px21.callback.TerminalLocationNotificationCallback
Service type	TerminalLocation
Exposes to the service communication layer a Java representation of:	Parlay X 2.1 Part 9: Terminal Location
Interfaces with the network nodes using:	MLP 3.0/3.2
Deployment artifact: NT EAR wlng_nt_terminal_location_px21.ear	Plugin_px21_terminal_location_mlp.jar, px21_terminal_location_service.jar, and terminal_location_mlp.war
Deployment artifact: AT EAR: Normal wlng_at_terminal_location_px21.ear	px21_terminal_location.war, px21_terminal_location_callback.jar, and rest_terminal_location.war
Deployment artifact: AT EAR: SOAP Only wlng_at_terminal_location_px21_soap.ear	px21_terminal_location.war and px21_terminal_location_callback.jar

Configuration Workflow for Parlay X 2.1 Terminal Location/MLP

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID listed in the "Properties for Parlay X 2.1 Terminal Location/MLP"section.
2. Using the Administration Console or an MBean browser, select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID given when the plug-in instance was created.
3. Configure the attributes of the plug-in instance:
   - Attribute: CharacterEncoding
   - Attribute: CleanupInterval
   - Attribute: DecimalDegreesToDMSH
   - Attribute: MaxDuration

- Attribute: MlpVersionSupported
- Attribute: MlpLocationEstimates
- Attribute: MlpPushAddr
- Attribute: MlpRequestType
- Attribute: MlpServerUrl
- Attribute: MlpSrsName
- Attribute: MlpVersionSupported
- Attribute: MsidType
- Attribute: Password
- Attribute: Requestor
- Attribute: RequestTimeout
- Attribute: ServiceId
- Attribute: Username
- Attribute: XMLDoctypeTagUsage

4. Specify heartbeat behavior. See "Configuring Heartbeats" in *System Administrator's Guide*.
5. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 2.1 Terminal Location/MLP"section.
6. If required, create and load a node SLA. For details see “Defining Global Node and Service Provider Group Node SLAs” and “Managing SLAs” in the *Accounts and SLAs Guide*.
7. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Reference: Attributes for Parlay X 2.1 Terminal Location/MLP

This section describes the attributes for configuration and maintenance:

- Attribute: CharacterEncoding
- Attribute: CleanupInterval
- Attribute: DecimalDegreesToDMSH
- Attribute: MaxDuration
- Attribute: MlpAltitudeSupported
- Attribute: MlpLocationEstimates
- Attribute: MlpPushAddr
- Attribute: MlpRequestType
- Attribute: MlpServerUrl
- Attribute: MlpSrsName
- Attribute: MlpVersionSupported

- Attribute: MsidType
- Attribute: Password
- Attribute: Requestor
- Attribute: RequestTimeout
- Attribute: ServiceId
- Attribute: Username
- Attribute: XMLDoctypeTagUsage

Attribute: CharacterEncoding

Scope: Cluster

Unit: Not applicable

Format: String

Indicates the type of Unicode character encoding accepted by the MLP node. The values are not case sensitive. A typical value is UTF-8.

Attribute: CleanupInterval

Scope: Cluster

Unit: Seconds

Format: Integer [**0–3600**]

Specifies the time interval at which periodic notification expiration checks are performed.

Attribute: DecimalDegreesToDMSH

Scope: Cluster

Unit: Not applicable

Format: Boolean.

Specifies if the coordinates provided by an application, in the form of decimal degrees, should be converted to Degrees Minutes Seconds Hemisphere (DMSH) format.

Enter:

- `true` to convert to DMSH
- `false` to use decimal degrees

Attribute: MaxDuration

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the maximum duration for a periodic location request.

Rejects **startPeriodicNotification** and **startGeographicalNotification** requests on the TerminalLocationNotificationManager interface if the duration is larger than this value.

If the duration is not provided in the request, this value is used.

Attribute: MlpAltitudeSupported

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if the MLP server supports altitude requests. When set to `true`, the `<alt_acc>` element is included in requests towards the MLP server.

Only applicable when the plug-in instance operates in MLP 3.2 mode. See "Attribute: MlpVersionSupported" for more information.

Attribute: MlpLocationEstimates

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if the MLP server is allowed to estimate locations. Use `true` if estimates are allowed, otherwise `false`.

Defines the value of the **loc_estimates** attribute in MLP.

Attribute: MlpVersionSupported

Scope: Cluster

Unit: Not applicable

Format: String

Specifies which version of MLP to use.

Valid values are:

- **3.0.0**
- **3.2.0**

Attribute: MlpPushAddr

Scope: Server

Unit: Not applicable

Format: URL

Specifies the callback URL to which the MLP server delivers location reports, periodic or triggered. This is the URL at which the plug-in instance listens for location reports. The format for the URL is:

```
http://ipaddressOfNTMachine:portOfWLS/tl-mlp/mlp_client
```

For example:

```
http://172.16.0.0:8001/tl-mlp/mlp_client
```

Attribute: MlpRequestType

Scope: Cluster

Unit: Not applicable

Format: String

Specifies which type of location request to use towards the MLP server.

Defines the DTD to be used for constructing the request towards the MLP server.

Valid values are:

- **eme_lir** for EME_LIR (Emergency location request)
- **slir** for SLIR (Standard location request)

Attribute: MlpServerUrl

Scope: Cluster

Unit: Not applicable

Format: URL

Specifies the MLP server's URL.

Attribute: MlpSrsName

Scope: Cluster

Unit: Not applicable

Format: String

Specifies requested MLP **srsName** attribute.

Normally, this is **www.epsg.org#4326**.

Attribute: MsidType

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the MSID type of the subscriber's Mobile Station ID (MSID).

Valid values are "MSISDN" and "MDN". The default is "MSISDN".

Attribute: Password

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the password used when Services Gatekeeper connects to the MLP server. The password is provided by the MLP server administrator.

Attribute: Requestor

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the requestor ID. If set to an empty string, the `<requestorid>` element will not be used in the MLP request. The requestor ID is provided by the MLP server administrator.

Attribute: RequestTimeout

Scope: Cluster

Unit: Seconds

Format: Integer [**0–3600**]

Specifies the HTTP time-out for MLP requests.

Attribute: ServiceId

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the Services Gatekeeper service ID. If set to an empty string, the `<serviceid>` element will not be used in the MLP request. The service ID is provided by the MLP server administrator.

Attribute: Username

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the Services Gatekeeper user ID used for connecting to the MLP server. The user ID is provided by the MLP administrator.

Attribute: XMLDoctypeTagUsage

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if the XML tag `!`*DOCTYPE* should be included in requests towards the MLP node. Valid values are:

- `true` - include the tag
- `false` - do not include the tag

8 Parlay X 2.1 Third Party Call/INAP-SS7

This chapter describes the Parlay X 2.1 Third Party Call/Intelligent Network Application Part (INAP)-Signaling System #7 (SS7) communication service in detail.

Overview of the Parlay X 2.1 Third Party Call/INAP-SS7 Communication Service

The Parlay X 2.1 Third Party Call/INAP-SS7 communication service exposes the Parlay X 2.1 Third Party Call application interfaces.

The communication service uses the Tieto-SS7 stack to connect to an SS7 network. It acts as a Service Control Function (SCF) communicating with a Service Switching Function (SSF) in the SS7 network.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using a Parlay X 2.1 Third Party Call/ INAP- SS7 communication service, an application can:

- Set up a call between two parties.

  For example, an application could set up a call between an investor and a broker if a particular stock reaches a predetermined price. Or a computer user could set up a call between himself and someone in the address book with a mouse click.
- Query Services Gatekeeper for the status of a previously set up call.
- Cancel a call as it is about to be set up.
- Terminate an ongoing call it created.

How It Works

In the Parlay X 2.1 Third Party Call model, a call has two distinct stages:

- Call Setup
- Call Duration

Call Setup

There are two parties involved in Third Party Call calls: the A-party (the caller) and the B-party (the callee). When a call is set up using the Parlay X 2.1 Third Party Call /INAP-SS7 communication service, Services Gatekeeper attempts to set up a call leg to the A-party. When the caller goes off-hook (answers), Services Gatekeeper attempts to set up a call leg to the B-party. When the callee goes off-hook, the two call legs are connected using the underlying telecom network. This ends the call setup phase.

The application can cancel the call during this phase.

Call Duration

While the call is underway, the audio channel that connects the caller and the callee is completely managed by the telecom network. During this phase of the call, the application can only query as to the status of the call. A call can be terminated in two ways, either using the application-facing interface or having the caller or callee hang up.

Requests using this communication service flow only in one direction, from the application to the network. Therefore this communication service supports only application-initiated functionality.

The Parlay X 2.1 Third Party Call /INAP - SS7 communication service manages only the signalling, or controlling, aspect of a call. The media or audio channel is managed by the telecom network. Only parties residing on the same network can be controlled, unless:

- The network plug-in connects to a media gateway controller.
- One of the participants is connected to a signalling gateway so that, from a signalling point of view, all parties reside on the same network.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 Third Party Call communication service, see the discussion of Parlay X 2.1 Interfaces in *Application Developer's Guide*.

For information about the RESTful Third Party Call interface, see the discussion of Third Party Call in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on., they are the same.

Events and Statistics

The Parlay X 2.1 Third Party Call /INAP-SS7 communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 8–1 lists the IDs of the EDRs created by the Third Party Call/INAP-SS7 communication service. This does not include EDRs created when exceptions are thrown.

Table 8–1 Event Types Generated by Parlay X 2.1 Third Party Call/INAP-SS7

ED RID	Method Called
8022	makeCall

Table 8–1 (Cont.) Event Types Generated by Parlay X 2.1 Third Party Call/INAP-SS7

ED RID	Method Called
8023	getCallInformation
8024	endCall
8025	cancelCallRequest
8026	callConnected
8027	callReleasedNotification

Charging Data Records

Third Party Call /INAP-SS7 -specific CDRs are generated under the following conditions:

- When Services Gatekeeper receives an event from the network indicating that the second call leg has been connected and the associated phone has started to ring. This CDR is not dependent on whether the call is answered.
- When call information has been successfully delivered to the application.
- When the call is ended by the application.
- When the call request is canceled by the application.
- When the network notifies Services Gatekeeper that the call is connected. This occurs when the second participant has answered the call.
- When the network notifies Services Gatekeeper that a call participant has disconnected.

Statistics

Table 8–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counter:

Table 8–2 Methods and Transaction Types for Parlay X 2.1 Third Party Call/INAP-SS7

Method	Transaction Type
makeCall	TRANSACTION_TYPE_CALL_CONTROL_SERVICE_INITIATED

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing Parlay X 2.1 Third Party Call/INAP-SS7

This section describes the properties and workflow for setting up a Parlay X 2.1 Third Party Call/INAP-SS7 plug-in instance.

To configure SS7 connectivity, you must relate the settings in the management interface for the plug-in instance to a subset of the settings in the Stack-in-a-Box configuration files. See "INAP-SS7 Configuration Dependencies" for details.

Configuration and management of other parts of Stack-in-a-Box are outside the scope of this description. Refer to the TietoEnator SS7 product documentation.

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one to one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

Properties for Parlay X 2.1 Third Party Call/INAP-SS7

Table 8–3 lists the technical specifications for the communication service.

Table 8–3 Properties for Parlay X 2.1 Third Party call/INAP-SS7

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > Plugin_third_party_call_inap

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=Plugin_third_party_call_inap
	Type=com.bea.wlcp.wlng.plugin.tpc.inap.management.InapTpc MBean

Network protocol plug-in service ID	Plugin_third_party_call_inap
Network protocol plug-in instance ID	Plugin_px21_third_party_call_inap
Supported Address Scheme	tel
Application-facing interface	com.bea.wlcp.wlng.px21.plugin.ThirdPartyCallPlugin
Service type	ThirdPartyCall
Exposes to the service communication layer a Java representation of:	Parlay X 2.1 Part 2: Third Party Call
Interfaces with the network nodes using:	ETSI 94 INAP CS1, ETS 300 374-1
Deployment artifact: NT EAR wlng_nt_third_party_call_px21.ear	Plugin_px21_third_party_call_inap.jar. px21_third_party_call_service.jar
AT EAR: Normal Deployment artifact: wlng_at_third_party_call_px21.ear	px21_third_party_call.war and rest_third_party_call.war
AT EAR: SOAP Only Deployment artifact: wlng_at_third_party_call_px21_soap.ear	px21_third_party_call.war

Configuration Workflow for Parlay X 2.1 Third Party Call/INAP/SS7

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Make sure the SS7 stack is configured and running. You must define an INAP user for each plug-in instance.
2. Configure connection information for the connection to the SS7 stack:
   - Attribute: LocalSpc
   - Attribute: LocalSsn
   - Attribute: RemoteSpc
   - Attribute: RemoteSsn
   - Attribute: TSCFTimeout
   - Attribute: NoAnswerTimeout
   - Attribute: SccpPriority
   - Attribute: SccpQualityOfService
   - Attribute: InapUserId
   - Attribute: Ss7Host
   - Attribute: Ss7PortNumber
   - Attribute: InapBindTimeout

   Note: When any of these attributes are changed, the "INAP API Configuration File" is overwritten. The plug-in service must be restarted for the change to take effect.

3. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 2.1 Third Party Call/INAP-SS7" section.
4. If required, create and load a node SLA. For details see “Defining Global Node and Service Provider Group Node SLAs” and “Managing SLAs” in the *Accounts and SLAs Guide*.
5. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Reference: Attributes and Operations for Parlay X 2.1 Third Party Call/INAP

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: InapBindTimeout
- Attribute: InapUserId
- Attribute: LocalSpc
- Attribute: LocalSsn
- Attribute: NoAnswerTimeout
- Attribute: RemoteSpc
- Attribute: RemoteSsn
- Attribute: TSCFTimeout

- Attribute: SccpPriority
- Attribute: SccpQualityOfService
- Attribute: Ss7Host
- Attribute: Ss7PortNumber

Attribute: InapBindTimeout

Scope: Server

Unit: Milliseconds

Format: Integer

Specifies the stack bind timeout value.

Attribute: InapUserId

Scope: Server

Unit: Not applicable

Format: String

Specifies the user ID used by the INAP plug-in when connecting to the SS7 stack. Must be defined in the common parts configuration file. See "Common Parts Configuration File" for more information.

Attribute: LocalSpc

Scope: Server

Unit: Not applicable

Format: Integer [**0–16383**] or [**0–16777215**], depending on the standard used.

Specifies the local SCCP Signaling Point Code (SPC) served by the local SS7 stack. This is the SS7 network address for the plug-in instance. Used as the originating SPC by the plug-in instance.

Must be correlated with the property SCCP Local SPC in the back-end configuration file for the SS7 stack.

Attribute: LocalSsn

Scope: Server

Unit: Not applicable

Format: Integer [**2–254**]

Specifies the local SCCP Sub System Number to which the plug-in instance will bind itself.

Must be correlated with the property SCCP Local SSN in the back-end configuration file for the SS7 stack.

Attribute: NoAnswerTimeout

Scope: Server

Unit: Seconds

Format: Integer [**0–2047**]

Specifies the time-out value for an INAP noAnswer event. Used towards the signaling network in INAP DpSpecificCriteria when arming the noAnswer event.

Attribute: RemoteSpc

Scope: Server

Unit: Not applicable

Format: Integer [**0–16383**] or [**0–16777215**], depending on the standard used.

Specifies the remote SCCP Signaling Point Code (SPC). Used in the destination address.

Must be correlated with the property SCCP Remote SPC in the back-end configuration file for the SS7 stack.

Attribute: RemoteSsn

Scope: Server

Unit: Not applicable

Format: Integer [**2–254**]

Specifies the remote SCCP Signaling Subsystem Number (SSN). Used in the destination address.

Must be correlated with the property SCCP Remote SSN in the back-end configuration file for the SS7 stack.

Attribute: SccpPriority

Scope: Server

Unit: Not applicable

Format: Integer [**0–3**]

Specifies the SCCP priority indicator. 0 is the lowest priority and 3 is the highest priority.

Attribute: SccpQualityOfService

Scope: Server

Unit: Not applicable

Format: Integer [**0–3**]

Specifies the SCCP quality of service indicator.

Attribute: Ss7Host

Scope: Server

Unit: Not applicable

Format: String

Specifies the host name or IP address of the SS7 stack. Separate the host names or IP addresses with a comma (,) if the stack is running in HA mode. If you are using HA mode, use this attribute to define the port number (for example, 192.168.0.19:99) and do not use "Attribute: Ss7PortNumber" to specify the port number.

Attribute: Ss7PortNumber

Scope: Server

Unit: Not applicable

Format: Integer

Specifies the port number to use in connecting to the stack.

Attribute: TSCFTimeout

Scope: Server

Unit: Seconds

Format: Integer

Specifies the timeout value for the T(SCF) timer. Used for supervising call establishment.

This attribute specifies how long the plug-in instance should wait for a response from an SCP after sending a request. If the time-out value is exceeded, the TCAP dialog is aborted.

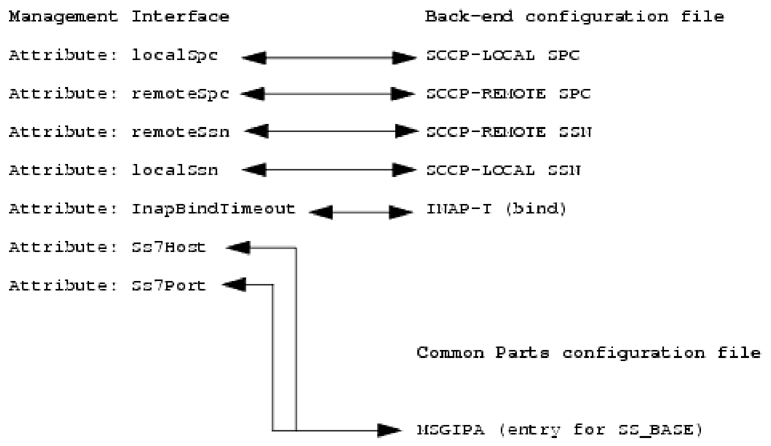
INAP-SS7 Configuration Dependencies

There is a set of dependencies and settings that must be correlated between the configuration files and the configuration settings in the plug-in instance. The following files have touch-points:

- INAP API configuration file (**Plugin_px21_third_party_call_inap.properties**). See "INAP API Configuration File" for more information.
- SS7 back-end configuration file (**itu_ss7.cfg**). See "Back-end Configuration File" for more information.
- Common parts configuration file (**cp.cnf**). See "Common Parts Configuration File" for more information.

The specific settings are explained in the sections describing the settings in the management interface for the plug-in instance and the description of the files. Figure 8–1 presents an overview of the dependencies.

Figure 8–1 Plug-in Property to SS7 Configuration File Dependencies



INAP API Configuration File

The INAP API configuration file is a configuration file for the TietoEnator JAIN INAP API library used by the plug-in instance.

This file provides the API with information on where to connect, how to bind to the stack, and values for some parameters that are not exposed in the API. If any of the properties are not set, default values are used.

Table 8–4 describes the properties that are related to the interface between the plug-in instance and the stack. See the documentation for the stack for a full description of all settings.

The file is named **Plugin_px21_third_party_call_inap.properties**. In default installations, it is located in *Domain_Home* on the file system of the host where the plug-in instance is running. The file is created and updated whenever an attribute is updated using the MBean for the Parlay x 2.1 Third Party Call/INAP plug-in. The plug-in service needs to be restarted for the changes to take effect.

Any changes to the MBean attributes cause the file to be overwritten, and hence any modifications to it are lost.

Table 8–4 INAP API Properties

Property	Comments
priority	SCCP Message priority.
quality-of-service	SCCP QoS.

Table 8–4 (Cont.) INAP API Properties

Property	Comments
inap-user-id	The common parts module ID used by the plug-in instance. See "Common Parts Configuration File". Either INAPUP or any of the USERxx IDs should be used by the plug-in instance. The numeric identifier of the ID should be used, not the ID itself (as used in the common parts configuration file). The module ID numeric values can be found in **/opt/ss7/ss7_ITU/include/portss7.h** in an installed stack. USER01 has decimal value 40, USER02 41, and so on.
ss7host	The host name or IP address of the host running the SS7 back-end. This is the address to which the SS7_BASE module ID is bound in the common parts configuration file. See "Common Parts Configuration File". If several SS7 back-ends are used, either in high-availability mode or horizontally distributed mode, enter the host name (or IP address) for the servers in a comma-separated list.
port-number	The port number to which the SS7_BASE module ID is bound in the common parts configuration file.
bind-timeout	Time to wait for bind response before a bind operation is considered a failure. Unit: milliseconds.
heartbeat-interval	Heartbeat interval between the INAP API and the common parts module. Must correspond to the MSGHBRATE and MSGHBLOST properties defined in the common parts configuration file. See "Common Parts Configuration File". **Note:** This property is not generated from the settings in the MBean of the Parlay X 2.1 Third Party Call/INAP plug-in. The absence of the property means that no heartbeats are sent. If heartbeats are used, this property must be added manually in the configuration file. If used, the recommended value is MSGHBRATE times MSGHBLOST. Any changes to the MBean attributes cause the file to be overwritten, and hence this setting is lost. Unit: milliseconds.

Example 8–1 Example INAP JAIN API Configuration file

```
local-ssn: 254
priority: 0
quality-of-service: 0
trace-level: 0
inap-user-id: 40
ss7host: 192.168.20.1,192.168.20.2
port-number: 7001
bind-timeout: 5000
```

Common Parts Configuration File

The SS7 common parts configuration file specifies the inter-process communication for the SS7 stack, including users of the stack. The plug-in instance acts as a user of the stack through the INAP API. Table 8–5 describes the dependencies on the plug-in instance. All other settings are related to the stack itself. See the documentation for the stack for a description of these settings.

The file is located on the file system of the host running the back-end part of the SS7 stack. In default installations this is in **/opt/ss7/ss7_ITU/etc/cp.cnf**.

Table 8–5 Common Parts Configuration File Properties with Dependencies on Plug-in Instance Settings

Property	Comments
MSGIPA	There must be one MSGIPA entry per plug-in instance. First, choose a Message Port owner ID (MP OwnerID). Use one of the following: ■ INAPUP ■ USER01 ■ USER02 ■ USER03 ■ USER04 ■ USER05 ■ USER06 ■ USER07 ■ USER08 ■ USER09 ■ USER10 MP OwnerID should correspond to inap-user-id specified in "Attribute: InapUserId". The IP-address (or host name) with TCP port number must correspond to the host where the plug-in instance is deployed. Make sure there is a MSGINTERACT entry per MP OwnerID. Instances of MP OwnerIDs are not supported. Example: MSGIPA=USER01,192.168.20.2:6701 10.10.10.11:6701
MSGHBLOST	Must correspond to "Attribute: LocalSpc".
MSGHBRATE	Must correspond to heartbeat-interval in the "INAP API Configuration File".

Back-end Configuration File

The back-end configuration file contains the configuration of the SS7 back-end stack layers. Each stack layer has a dedicated section in this file, and it is where, for example, SS7 network routing and protocol timers are configured. Table 8–6 describes the dependencies between the plug-in instance and the stack. All other settings are related to the stack itself. See the documentation for the stack for a description of these settings.

The file is located on the file system of the host running the back-end part of the SS7 stack. In default installations, this is in **/opt/ss7/ss7_ITU/etc/ss7_itu.cnf**.

Table 8–6 Back-end Configuration File Properties With Dependencies on Plug-in Instance Settings

Property	**Comments**
INAP-T (bind)	Must correspond to Attribute: InapBindTimeout.
SCCP-LOCAL SPC	Must correspond to Attribute: LocalSpc.
SCCP-LOCAL SSN	Must correspond to Attribute: LocalSsn.
SCCP-REMOTE SPC	Must correspond to Attribute: RemoteSpc.
SCCP-REMOTE SSN	Must correspond to Attribute: RemoteSsn.

9 Parlay X 2.1 Third Party Call/SIP

This chapter describes the Parlay X 2.1 Third Party Call/Session Initiation Protocol (SIP) communication service in detail.

Overview of the Parlay X 2.1 Third Party Call/SIP Communication Service

The Parlay X 2.1 Third Party Call/SIP communication service exposes the Parlay X 2.1 Third Party Call application interfaces.

The communication service connects to a SIP-IMS network using Oracle Converged Application Server. Converged Application Server is collocated with Services Gatekeeper in the network tier. In this relationship, Services Gatekeeper acts as a Back-to-Back User Agent for all calls.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specification in *Concepts Guid*e.

Using a Third Party Call Parlay X 2.1 communication service, an application can:

- Set up a call between two parties.

  For example, an application could set up a call between an investor and a broker if a particular stock reaches a predetermined price. Or a computer user could set up a call between himself and someone in the address book with a mouse click.
- Query Services Gatekeeper for the status of a previously set up call.
- Cancel a call as it is about to be set up.
- Terminate an ongoing call it created.

How It Works

In the Parlay X 2.1 Third Party Call model, a call has two distinct stages:

- Call Setup
- Call Duration

Call Setup

There are two parties involved in Third Party Call calls: the A-party (the caller) and the B-party (the callee). When a call is set up using a Third Party Call communication service, Services Gatekeeper attempts to set up a call leg to the A-party. When the caller goes off-hook (answers), Services Gatekeeper attempts to set up a call leg to the B-party. When the callee goes off-hook, the two call legs are connected using the underlying telecom network. This ends the call setup phase.

The application can cancel the call during this phase.

Call Duration

While the call is underway, the audio channel that connects the caller and the callee is completely managed by the telecom network. During this phase of the call, the application can only query for the status of the call. A call can be terminated in two ways, either using the application-facing interface, or having the caller or callee hang up.

Requests using a Parlay X 2.1 Third Party Call communication service flow only in one direction, from the application to the network. Therefore this communication service supports only application-initiated functionality.

The Parlay X 2.1 Third Party Call /SIP communication service manages only the signalling, or controlling, aspect of a call. The media or audio channel is managed by the telecom network. Only parties residing on the same network can be controlled, unless:

- The network plug-in connects to a media gateway controller.
- One of the participants is connected to a signalling gateway so that, from a signalling point of view, all parties reside on the same network.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 Third Party Call communication service, see the discussion of Parlay X 2.1 Interface in *Application Developer's Guide*.

For information about the RESTful Third Party Call interface, see the discussion of Third Party Call in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on, they are the same.

Events and Statistics

The Parlay X 2.1 Third Party Call/SIP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 9–1 lists the IDs of the EDRs created by the Parlay X 2.1 Third Party Call/ SIP communication service. This does not include EDRs created when exceptions are thrown.

Table 9–1 Event Types Generated by Parlay X 2.1 Third Party Call/SIP

EDR ID	Method Called
8022	makeCall
8023	getCallInformation
8024	endCall

Table 9–1 (Cont.) Event Types Generated by Parlay X 2.1 Third Party Call/SIP

EDR ID	Method Called
8025	cancelCallRequest

Charging Data Records

Parlay X 2.1 Third Party Call-specific CDRs are generated under the following conditions:

- When Services Gatekeeper has received an event from the network stating that the second call leg has been connected and the associated phone has started to ring. This CDR is not dependent on whether the call is answered.
- When call information has been successfully delivered to the application.
- When the call is ended by the application.
- When the call request is canceled by the application.

Statistics

Table 9–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counter:

Table 9–2 Methods and Transaction Types for Parlay X 2.1 Third Party Call /SIP

Method	Transaction Type
makeCall	TRANSACTION_TYPE_CALL_CONTROL_SERVICE_INITIATED

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing Parlay X 2.1 Third Party Call/SIP

This section describes the properties and workflow for setting up a Parlay X 2.1 Third Party Call/SIP protocol translation module.

Parlay X 2.1 Third Party Call/SIP uses two parts for SIP connectivity: a part that executes as a network protocol plug-in instance in Services Gatekeeper container and a part that executes as a SIP application in the SIP Server container.

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one to one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

Properties for Parlay X 2.1 Third Party Call/SIP

Table 9–3 lists the technical specifications for the communication service.

Table 9–3 Properties for Parlay X 2.1 Third Party Call/SIP

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > Plugin_third_party_call_sip

Table 9–3 (Cont.) Properties for Parlay X 2.1 Third Party Call/SIP

Property	Description

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=Plugin_px21_third_party_call_sip
	Type=com.bea.wlcp.wlng.plugin.tpc.sip.management.TPCMBean

Network protocol plug-in service ID	Plugin_px21_third_party_call_sip
Network protocol plug-in instance ID	Plugin_px21_third_party_call_sip
Supported Address Scheme	sip, tel
Application-facing interface	com.bea.wlcp.wlng.px21.plugin.ThirdPartyCallPlugin
Service type	ThirdPartyCall
Exposes to the service communication layer a Java representation of:	Parlay X 2.1 Part 2: Third Party Call
Interfaces with the network nodes using:	SIP: Session Initiation Protocol, RFC 3261
Deployment artifacts	px21_third_party_call_service.jar, Plugin_px21_third_party_call_sip.jar and Plugin_px21_third_party_call_sip.war, packaged in wlng_nt_third_party_call_px21.ear px21_third_party_call.war, packaged in wlng_at_third_party_call_px21.ear

This Plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one-to-one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

Configuration Workflow for Parlay X 2.1 Third Party Call/SIP

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Select the MBean listed in the "Properties for Parlay X 2.1 Third Party Call/SIP" section.
2. Configure behavior of the network protocol plug-in instance:
   - Attribute: ChargingAllowed
   - Attribute: ControllerURI
   - Attribute: ISCRouteURI
   - Attribute: MaximumCallLength
   - Attribute: StatusRetentionTime
3. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 2.1 Third Party Call/SIP" section.

4. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
5. Provision service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Reference: Attributes for Parlay X 2.1 Third Party Call/SIP

Following is a list of attributes for configuration and maintenance:

- Attribute: ChargingAllowed
- Attribute: ControllerURI
- Attribute: ISCRouteURI
- Attribute: MaximumCallLength
- Attribute: StatusRetentionTime

Attribute: ChargingAllowed

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if charging is allowed, meaning that the **charging** parameter is allowed to be present in a request from an application.

Attribute: ControllerURI

Scope: Cluster

Unit: Not applicable

Format: String in URI format

Specifies the Controller SIP URI that is used to establish the third party call. If this value is set, a call appears to the callee to come from this URI. By default, the value is **None**, where no controller URI is used to establish the call. In this case, the call appears to the callee to come from the caller

Attribute: ISCRouteURI

Scope: Cluster

Unit: Not applicable

Format: String in URI format

Specifies the URI of the IMS service control route.

Attribute: MaximumCallLength

Scope: Cluster

Unit: Minutes

Format: Integer

Specifies for maximum length allowed for a call. If this time expires, the call is terminated.

Attribute: StatusRetentionTime

Scope: Cluster

Unit: Minutes

Format: Integer

Specifies how long to retain status information about a call after it has been terminated.

10 Parlay X 2.1 Terminal Status/MAP

This chapter describes the Parlay X 2.1 Terminal Status/Mobile Application Part (MAP) communication service in detail.

Overview of the Parlay X 2.1 Terminal Status/MAP Communication Service

The Parlay X 2.1 Terminal Status/MAP communication service exposes the Parlay X 2.1 Terminal Status set of application interfaces.

The communication service uses the TietoEnator-SS7 stack to connect to an SS7 network. The communication service acts as an MAP application client for the SS7 network.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using the Parlay X 2.1 Terminal Status/MAP 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 only if the status changes. The terminal statuses are checked as frequently as you specify, for a time period you specify.

By default, the Parlay X 2.1 Terminal Status/MAP communication service directs the network to probe for the terminal status (network-triggered). You also have the choice of using the plug-in itself to do the probing (plug-in-triggered). Network-triggered is the default because it supports high-availability features and is generally more efficient. However if for some reason your network cannot support ATM operations, you can switch to plug-in-triggered.

The communication service receives requests from applications, opens a transaction, translates the requests into the SS7/MAP protocol and queries the network. The network then returns the status to the plug-in, which passes the status back to the application inside the transaction.

To receive current terminal statuses for multiple terminals in the same transaction, an application sends the addresses of the terminals to the communication service. The communication service then opens a transaction and sends a separate query to the network for each terminal address. The plug-in collects replies as they come in and returns them to the application inside a transaction.

To receive notification if a terminal status changes, an application sends the address of the terminal to the plug-in. The plug-in checks the status of the terminal periodically during the configured time period on a schedule defined by the frequency, duration, and count timer metrics to **startNotification**. See the Parlay X 2.1 Part 8 web site at:

http://docbox.etsi.org/TISPAN/Open/OSA/ParlayX21.html

or RESTful interface in the discussion of Terminal Status in the *Restful Application Developer's Guide* for details.

The Terminal Status/MAP plug-in takes these parameters as input:

- The terminal address or addresses.
- The request frequency (number of seconds, minutes, or hours).
- The total number of requests.
- A true/false value for the **checkImmediate** parameter.

If **checkImmediate**=`true` the plug-in checks the status of the terminal or terminals immediately and thereafter as often as you specified. If **checkImmediate**=`false` the plug-in checks for status at the end of the first time period.

By default the plug-in uses **anyTimeModification** to perform the network-triggered probe for a status change. If you change this to plug-in-triggered notification, the plug-in uses **anyTimeInterrogation** to probe for the status. If your network does not support **anyTimeModification**, use plug-in triggered notification.

The plug-in retains connection and identification information so it can query other nodes if the one it first contacts is unresponsive. This strategy also allows multiple applications to query the same terminal status. The plug-in confirms that all interested applications have ended their queries before ending the transaction.

If your network supports high-availability features, network-triggered probing can take advantage of them. The Parlay X 2.1 Terminal Status/MAP communication service can automatically query different network nodes if one is unresponsive and confirm that no other applications have an open status change query operation.

Status Request for a Single Terminal

An application queries Services Gatekeeper for the status of a terminal by sending the terminal's address in a **getStatus** operation. The communication service translates the request into a SS7/MAP **anyTimeInterrogation** operation and passes it to the network node. The network node returns the status (reachable, unreachable, or busy) to the requesting application synchronously.

Status Requests for Multiple Terminals

An application queries Services Gatekeeper for the status of a group of terminals by sending the terminal addresses in a **getStatus** operation. The communication service translates the request into an SS7/MAP **anyTimeInterrogation** operation and forwards it to the network node. The node returns the statuses for each terminal (reachable, unreachable, or busy) separately. The plug-in collects the results and returns them to the requesting application synchronously with a single message. Unreachable terminals are given a status of NotRetrieved.

Terminal Status Change Request: Network-Triggered

As long as your network supports **anyTimeModification** (ATM) calls, use network-triggered poll status operations to direct your network to do the probing. The **AllowATM** attribute controls this choice. If **AllowATM**=`true`, the Terminal

Server/MAP plug-in passes the **startNotification** terminal addresses and timer metrics through for your network to interpret.

If the terminal status changes during the time period, the network sends a **NoteMMEvent** call to the plug-in, which passes that information to the application using **statusNotification** and closes the transaction.

If the status does not change during the time period, the plug-in sends an **endNotification** back to the application and closes the transaction.

Requesting status for nested groups of terminals is not supported.

Terminal Status Change Request: Plug-in-Triggered

If your network cannot support ATM operations, you can set **AllowATM**=`false` and direct the plug-in to probe for terminal status changes. In this case, an application sends a terminal status change request and timer metrics to the Terminal Status/MAP plug-in using **startNotification** operation. The plug-in creates a transaction and starts querying the network for the terminal status with **anyTimeInterrogation** as directed by the timer metrics. If the terminal status changes during the time period, the communication service returns that information to the application using **statusNotification** and closes the transaction. If the status does not change during the time period, it sends a **statusEnd** message back to the application and closes the transaction. The application can also end the transaction itself by sending an **endNotification**.

Requesting status for nested groups of terminals is not supported.

This implementation creates a connection between the plug-in and a specific node. If that node crashes, the transaction never completes.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 2.1 Terminal Status communication service, see the discussion of Parlay X 2.1 Interfaces in *Application Developer's Guide*.

For information about the RESTful Call Notification interface, see the discussion of Terminal Status in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on., they are the same.

Events and Statistics

The Parlay X 2.1 Terminal Status/MAP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 10–1 lists the IDs of the EDRs created by the Parlay X 2.1 Terminal Status/MAP communication service.

Table 10–1 Event Types Generated by Parlay X 2.1Terminal Status/MAP

EDR ID	Method Called
4001	getStatus()
4000	getStatusForGroup()
4002	startNotification()
4003	endNotification()
4011	generate()
4015	statusNotification()
4016	statusError()
4017	statusEnd()

Charging Data Records

Terminal Status/MAP-specific CDRs are generated under the following conditions:

- With the results of a **getStatus** or **getStatusForGroup** operation
- When a notification is received from the network
- If you requested a terminal status change notification, the terminal's present status is sent periodically.

Statistics

Table 10–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 10–2 Methods and Transaction Types for Parlay X 2.1 Terminal Status/MAP

Method	Transaction type
getStatus()	TRANSACTION_TYPE_USER_STATUS
getStatusForGroup()	TRANSACTION_TYPE_USER_STATUS
startNotification()	TRANSACTION_TYPE_USER_STATUS
endNotification()	TRANSACTION_TYPE_USER_STATUS
generate()	TRANSACTION_TYPE_USER_STATUS
statusNotification()	TRANSACTION_TYPE_USER_STATUS
statusError()	TRANSACTION_TYPE_USER_STATUS
statusEnd()	TRANSACTION_TYPE_USER_STATUS

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing Parlay X 2.1 Terminal Status/MAP

This section describes the properties and workflow for the Parlay X 2.1 Terminal Status plug-in instance.

This plug-in service does not support multiple instantiation using the Plug-in Manager. You can create only one instance by using the Plug-in Manager. The plug-in instance is not automatically created when the plug-in service is started.

Properties for Parlay X 2.1 Terminal Status/MAP

Table 10–3 lists the technical specifications for the communication service.

Table 10–3 Properties for Parlay X 2.1 Terminal Status/MAP

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plugin_instance_id*

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=same as the network protocol instance_id assigned when the plug-in instance is created
	Type=com.bea.wlcp.wlng.plugin.ts.map.management.MapTsMBean

Network protocol plug-in service ID	Plugin_px21_terminal_status_map
Network protocol plug-in instance ID	The ID assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
Supported Address Scheme	tel
Application-facing interfaces	com.bea.wlcp.wlng.px21.plugin.TerminalStatusPlugin com.bea.wlcp.wlng.px21.plugin.TerminalStatusNotificationManagerPlugin com.bea.wlcp.wlng.px21.callback.TerminalStatusNotificationCallback
Service type	TerminalStatus
Exposes to the service communication layer a Java representation of:	Parlay X 2.1 Part 8: Terminal Status/MAP
Interfaces with the network nodes using:	3GPP TS 29.002 V4.18.0 (2007-09)
Deployment artifacts: NT EAR wlng_nt_terminal_status_px21.ear	px21_terminal_status.war, px21_terminal_status_callback.jar, and rest_terminal_location.war
AT EAR: Normal wlng_at_terminal_status_px21.ear	px21_terminal_status.war, px21_terminal_status_callback.jar, and rest_terminal_location.war
AT EAR: SOAP Only wlng_at_terminal_status_px21.ear	px21_terminal_status.war and px21_terminal_status_callback.jar

Configuration Workflow for Parlay X 2.1 Terminal Status/MAP

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID listed in the "Properties for Parlay X 2.1 Terminal Status/MAP" section.
2. Select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID assigned when the plug-in instance was created.
3. Configure the attributes of the plug-in instance:
   - Attribute: AllowATM
   - Attribute: ATITimeout
   - Attribute: CpUserId
   - Attribute: GroupRequestTimeout
   - Attribute: GsmSCFGT
   - Attribute: LocalSpc
   - Attribute: LocalSsn
   - Attribute: NoOfPluginInstances
   - Attribute: Ss7Host
   - Attribute: Ss7Instances
   - Attribute: Ss7PortNumber
4. Specify heartbeat behavior. See "Configuring Heartbeats" in *System Administrator's Guide*.
5. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 2.1 Terminal Status/MAP" section.
6. Set up at least one network selection and one network selection route to direct Terminal Server Status requests to the correct SSN using "Operation: addNetworkSelection" and "Operation: addNetworkSelectionRoute" operations. See "Setting Up Network Selection Routes and Network Selections" for more information.
7. (Optional) Create and load a node SLA with Terminal Status usage restrictions. These usage restrictions are supported:
   - SLA Usage Restriction: BusyAvailable
   - SLA Usage Restriction: MaximumCount
   - SLA Usage Restriction: MaximumNotificationAddresses
   - SLA Usage Restriction: MaximumNotificationDuration
   - SLA Usage Restriction: MaximumNotificationFrequency
   - SLA Usage Restriction: UnlimitedCountAllowed

   For details see “Defining Global Node and Service Provider Group Node SLAs” and “Managing SLAs” in the *Accounts and SLAs Guide*.

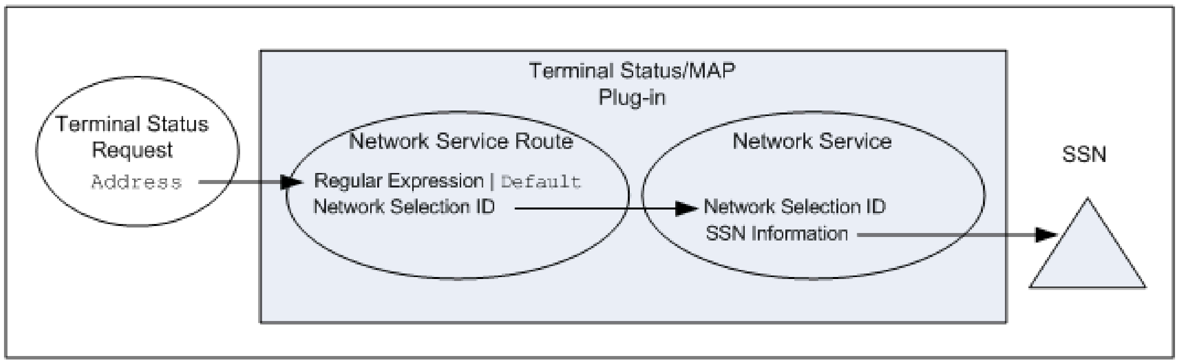
Setting Up Network Selection Routes and Network Selections

The Parlay X 2.1 Terminal Status/MAP plug-in matches each terminal status request with the correct SSN (HLR) node using network selections and network selection routes. You create these network selections and network selection routes using these operations:

- Operation: addNetworkSelection
- Operation: addNetworkSelectionRoute

Figure 10–1 illustrates how the plug-in routes terminal status requests with the correct SSN.

Figure 10–1 Terminal Status/MAP Request-to-Node Flow



The **addNetworkSelectionRoute** operation offers the option of associating one or more regular expressions, or the string **default**, with each of your service providers. When a request arrives, the plug-in attempts to match the address against all network service route regular expressions. If it finds a match, it then the passes the request to the network service with the matching network selection ID. The network service then matches the network selection ID with an SSN and passes the terminal status request to that SSN. If the network service route could not match the address with a regular expression, it passes the request to the **default** network selection ID if one is defined.

The plug-in does not do any overlap testing of the regular expressions you create in network service routes.

The **addNetworkSelection** operation specifies an SSN to use for a terminal status request for a network selection ID. It also specifies details about the type and format of the request.

At least one network selection and network selection route must be defined for the plug-in to function. Typically you set up at least one pair per service provider.

After network selections and network selection routes are set up, manage them with these operations:

- Operation: addNetworkSelection
- Operation: addNetworkSelectionRoute
- Operation: listAllNetworkSelectionRoutes
- Operation: listNetworkSelectionRoutes
- Operation: listNetworkSelections
- Operation: removeNetworkSelection
- Operation: removeNetworkSelectionRoute

Reference: Attributes and Operations for Parlay X 2.1 Terminal Status/MAP, and SLA Usage Restrictions

This section describes the attributes, operations, and SLA usage restrictions for configuration and maintenance:

- Attribute: AllowATM
- Attribute: ATITimeout
- Attribute: CpUserId
- Attribute: GroupRequestTimeout
- Attribute: GsmSCFGT
- Attribute: JCPQueueSize
- Attribute: LocalSpc
- Attribute: LocalSpc
- Attribute: NoOfPluginInstances
- Attribute: PluginInstanceId
- Attribute: Ss7Host
- Attribute: Ss7Instances
- Attribute: Ss7Instances
- Operation: addNetworkSelection
- Operation: addNetworkSelectionRoute
- Operation: bindToStack
- Operation: getNetworkSelection
- Operation: addNetworkSelectionRoute
- Operation: listAllNetworkSelectionRoutes
- Operation: listNetworkSelectionRoutes
- Operation: listNetworkSelections
- Operation: removeNetworkSelection
- Operation: removeNetworkSelectionRoute
- Operation: removeNotifications
- SLA Usage Restriction: BusyAvailable
- SLA Usage Restriction: MaximumCount
- SLA Usage Restriction: MaximumNotificationAddresses
- SLA Usage Restriction: MaximumNotificationDuration
- SLA Usage Restriction: MaximumNotificationFrequency
- SLA Usage Restriction: UnlimitedCountAllowed

Attribute: AllowATM

Scope: Cluster

Unit: Not applicable

Format: Boolean

Determines whether the StatusNotification interface uses **AnyTimeModification** (instead of the default **AnyTimeInterrogation**) to set up the terminal status notification. The default value is `true`.

Attribute: ATITimeout

Scope: Cluster

Unit: Milliseconds

Format: Integer

The timeout period used by each **AnyTimeInterrogation** call. The default value is **5000**.

Attribute: CpUserId

Scope: Cluster

Unit: Not applicable

Format: String

The CommonParts user ID that you map to a CommonParts User ID instance in the SS7 stack configuration file. The default value is **USER01**.

Attribute: GroupRequestTimeout

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies a time limit for the **getStatusForGroup** and **startNotification** (with **AnyTimeInterrogation**) interfaces. If all addresses are not processed within this time period, the Terminal Status/MAP plug-in aborts the query process, reports any retrieved terminal statuses, and marks the rest as NotRetrieved. The default value is **30**.

Attribute: GsmSCFGT

Scope: Local

Unit: Not applicable

Format: Integer

The local global title, used in originating addresses.

Attribute: JCPQueueSize

Scope: Cluster

Unit: Number of JCP queue messages

Format: Integer

Sets the JCP (TE stack attribute) message queue size. This queue stores the most recent MAP primitive messages in the JCP attribute of the TE stack, which the JTCAP provider then retrieves. A large queue size allows you to store more messages which helps ensure that none are abandoned before the JTCP provider can retrieve them. A large queue size is useful during high workload periods, but it also requires more memory. The default value is **90**.

Attribute: LocalSpc

Scope: Local

Unit: Not applicable

Format: Integer [**0–16383**] or [**0–16777215**], depending on the standard used.

The SS7 network address for the plug-in instance (the local SCCP Signaling Point Code (SPC) served by the local SS7 stack). The Terminal Status/MAP plug-in uses this as the originating SPC.

You must correlate this value with the property SCCP Local SSN in the SS7 stack back-end configuration file.

Attribute: LocalSsn

Scope: Local

Unit: Not applicable

Format: Integer [**2–254**]

The local SCCP Sub System Number to bind to this plug-in instance.

You must correlate this value with the property SCCP Local SSN in the SS7 stack back-end configuration file.

Attribute: NoOfPluginInstances

Scope: Local

Unit: Not applicable

Format: Integer

Total number of Terminal Status plug-in instances that use the Attribute: LocalSsn. The default value is **1**.

Attribute: PluginInstanceId

Scope: Local

Unit: Not applicable

Format: Integer

Must be 0 or less than the value for Attribute: NoOfPluginInstances. Used to create a unique dialogue reference for every plug-in instance using a different **LocalSsn**. The SS7 stack uses this unique reference to identify the correct Terminal Status/MAP plug-in to reply to. The default value is **0**.

Attribute: Ss7PortNumber

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the SS7 port number for the Terminal Status/MAP plug-in instance to use. The default value is **7001**.

Attribute: Ss7Host

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the SS7 host IP address for the Terminal Status/MAP plug-in instance to use. The default value is **127.0.0.1**.

Attribute: Ss7Instances

Scope: Cluster

Unit: Not applicable

Format: String

The SS7 BackEnd for the Terminal Status/MAP plug-in instance to use, encoded as a comma-separated byte array, for example, 1, 2, 3. Set this value to 0 if you only use one active BackEnd. The default value is **0**.

Operation: addNetworkSelection

Scope: Local

Adds a network selection.

Maps a service provider to an SSN (that presumably has terminal status information). This operation associates a network section ID with a specific SSN, and includes terminal status format details.

The values specified in this operation depend on the SSN configuration and the MAP message's target node. Network selection determines how MAP messages are encoded for a specific address, for example whether it uses GT+SSN or SPC+SSN. Network selection also specifies how **ApplicationContext** and **DialogueAS** appear in the ATI message. **NetworkSelectionRoutes** is then used to determine which network selection to use for a specific address.

This operation throws an exception if any parameter values are outside of their valid ranges, or if there is a problem adding the configuration to the database.

See Setting Up Network Selection Routes and Network Selections for more information.

Signature:

```
addNetworkSelection(NetworkSelectionId: String, MAPVersion: String,
MAPApplicationContext: String, MAPApplicationHandlingContext: String,
MAPApplicationReportingContext: String, MAPDialogueAS: String, SSN: Integer, DPC:
Integer, GTI: Integer, TT: Integer, Na: Integer, Np: Integer)
```

Table 10–4 addNetworkSelection Parameters

Parameter	**Data Type**	**Description**
NetworkSelectionId	String	Identifies each service provider this operation is creating a network selection for. Used by **addNetworkSelectionRoute** to map a terminal addresses with a service provider. This operation then specifies an SSN to query for the address. To avoid confusion, use a separate string than the Services Gatekeeper service provider ID.
MAPVersion	String	Identifies the MAP version that the PDU uses. Currently only 3GPP TS 29.002 V4.18.0 (2007-09) is supported.
MAPApplicationContext	String	The application context name to use, encoded as a comma separated byte array, for example: 4, 0, 0, 1, 0, 29, 3. Where: ■ 4: ccitt/identified-organization ■ 0: etsi ■ 0: mobileDomain ■ 1: gsm-Network ■ 0: map-acany ■ 29: TimeInfoEnquiry 29; ■ 3: version3
MAPApplicatonHandling Context	String	Application context name of ATM message to use, encoded as a comma separated byte array, for example: 4, 0, 0, 1, 0, 43, 3 Where: 4 ccitt/identified-organization 0 etsi 0 mobileDomain 1 gsm-Network 0 map-ac 43 anyTimeInfoHandling 3 version3
MAPApplicationReportin gContext	String	Application context name of ATM message to use, encoded as a comma separated byte array, for example: 4, 0, 0, 1, 0, 42, 3 Where: ■ 4 ccitt/identified-organization ■ 0 etsi ■ 0 mobileDomain ■ 1 gsm-Network ■ 0 map-ac ■ 43 mm-EventReporting ■ 3 version3

Table 10–4 (Cont.) addNetworkSelection Parameters

Parameter	Data Type	Description
MAPDialogueAS	String	The object identifier to use for `DialogueAS`, encoded as a comma separated byte array, for example: 4, 0, 0, 1, 1, 1, 1 Where: 4: ccitt/identified-organization 0: etsi 0: mobileDomain 1: gsm-Network 1: as-Id 1: map-DialoguePDU 1: version1
SSN	Integer	The destination SSN.
DPC	Integer	(Optional) The destination SPC. A value of -1 directs this operation to use GT+SSN instead of SPC+SSN in the destination address.
GTI	Integer	The global title indicator. Controls how the SCCP address for both called and calling addresses is built. Suitable values would be something like (international E.164 number): GTI=4 TT=1 NP=1 NA=4. Can be one of these values: 0: GT included 1: GT includes nature of address indicator only 2: GT includes translation type only 3: GT includes translation type, numbering plan, and encoding scheme 4: GT includes translation type, numbering plan, encoding scheme, and nature of address indication
TT	Integer	The translation type that directs the message to the appropriate GT translation function. Must be in the range 0-254.
Na	Integer	Nature of Address indicator (GSMSCF-Address). Can be one of: 0: unknown 1: international number 2: national significant number 3: network specific number 4: subscriber number 5: reserved 6: abbreviated number 7: reserved for extension

Table 10–4 (Cont.) addNetworkSelection Parameters

Parameter	Data Type	Description
Np	Integer	Address Numbering Plan indicator (GSMSCF-Address). Can be one of: 0: unknown 1: ISDN/Telephony Numbering Plan (Rec ITU-T E.164) 2: spare 3: data numbering plan (ITU-T Rec X.121) 4: telex numbering plan (ITU-T Rec F.69) 5: spare 6: land mobile numbering plan (ITU-T Rec E.212) 7: spare 8: national numbering plan 9: private numbering plan 15: reserved for extension

Operation: addNetworkSelectionRoute

Scope: Local

Maps a terminal address with a service provider ID. The plug-in uses the service provider ID to determine the SSN to probe for terminal status information.

Each network selection route needs to be associated with a network selection (using **addNetworkSelection**) that maps a service provider with the SSN with terminal status information. The plug-in is only active if at least one **NetworkSelectionRoute** / **NetworkSelection** pair is configured.

Note: The network selection ID string and your Services Gateway service provider ID identify the same service providers, but these are used for different purposes. To avoid confusion use different values for these strings.

When the terminal status request enters the plug-in, its MSISDN address(es) are matched against the regular expressions of the network selection routes. If a match is found, the request is sent to the SSN specified by the network selection. If the network selection route uses the **default** expression, all requests are sent to the default SSN.

Wildcards are allowed in the regular expression. For example, the value **^46730.*** matches all MSISDN addresses that start with **46730**. However this plug-in does not check for wildcard overlapping.

An exception is thrown if no corresponding network selection exists for the network selection route, if another route already exists for the service provider ID, or if there is a problem adding the route to the database.

See Setting Up Network Selection Routes and Network Selections for more information.

Signature:

```
addNetworkSelectionRoute(NetworkSelectionId: String, Expression: String)
```

Table 10–5 addNetworkSelectionRoute Parameters

Parameter	Data Type	Description
NetworkSelectionId	String	Identifies each service provider this operation is creating a network selection route for. To avoid confusion, use a different string than the Services Gatekeeper service provider ID.
Expression	String	[**default** | *`msisdn_address`*] The routing expression. A unique regular expression to match against the target MSISDN address or the string **default** (use the default route if no match exists). For example ^46730.*, There is no overlap control for the regular expressions.

Operation: bindToStack

Scope: Local

Binds the Terminal Server/MAP plug-in to the SS7 stack. This operation is required if the configuration has changed. If **bindToStack** is successful, the Terminal Status/MAP plug-in not yet connected; it has only started to connect and bind to the stack. Use the **Bound** parameter to confirm whether the stack is actually bound. This is operation is performed implicitly at startup, but must be manually invoked if any connection attributes were changed.

Signature:

```
bindToStack()
```

Operation: getNetworkSelection

Scope: Local

Retrieves the configured values for a specific network selection and the network selection route it maps to.

Signature:

```
getNetworkSelection(NetworkSelectionId: String)
```

Table 10–6 getNetworkSelection Parameters

Parameter	Data Type	Description
NetworkSelectionId	String	Identifies the service provider this operation is retrieving a network selection for.

Operation: getNetworkSelectionRoute

Scope: Local

Retrieves the network selection route and the network selections it maps to.

Signature:

```
getNetworkSelectionRoute(Expression: String)
```

Table 10–7 getNetworkSelectionRoute Parameters

Parameter	Data Type	Description
Expression	String	A unique regular expression to match against the target MSISDN address, or the string **DEFAULT** (use the default route if no match exists). For example ^46730.*, Note that there is no overlap control for the regular expressions.

Operation: listAllNetworkSelectionRoutes

Scope: Local

List all the configured NetworkSelectionRoutes and the NetworkSelection IDs each is mapped to.

Signature:

```
listAllNetworkSelectionRoutes()
```

Operation: listNetworkSelectionRoutes

Scope: Local

Lists all network selection routes mapped to a specific network selection.

Signature:

```
listNetworkSelectionRoutes(NetworkSelectionId: String)
```

Table 10–8 listNetworkSelectionRoutes Parameters

Parameter	Data Type	Description
NetworkSelectionId	String	Identifies the service provider this operation is listing network selection routes for.

Operation: listNetworkSelections

Scope: Local

Lists all network selections, their configured values, and their corresponding network selection routes.

Signature:

```
listNetworkSelections()
```

Operation: removeNetworkSelection

Scope: Local

Removes a network selection. Returns the removed network selection or `null` if it cannot be removed.

Signature:

```
removeNetworkSelection(NetworkSelectionId: String)
```

Table 10–9 removeNetworkSelection Parameters

Parameter	Data Type	Description
NetworkSelectionId	String	Identifies the service provider this operation is removing a network selection for.

Operation: removeNetworkSelectionRoute

Scope: Local

Removes the specified network selection route. Returns the removed network selection, or `null` if the network selection could not be removed.

Signature:

```
removeNetworkSelectionRoute(NetworkSelectionId: String)
```

Table 10–10 removeNetworkSelectionRoute Parameters

Parameter	Data Type	Description
NetworkSelectionId	String	Identifies the service provider that this operation is removing a network selection route for.

Operation: removeNotifications

Scope: Local

Explicitly removes notifications from the database. After a server crash old (inactive) notifications are removed implicitly.

Signature:

```
removeNotifications(IncludeActiveNotifications: Boolean)
```

Table 10–11 removeNotifications Parameters

Parameter	Data Type	Description
IncludeActiveNotifications	Boolean	`true`: Removes all current notifications `false`: Removes only inactive notifications.

SLA Usage Restriction: BusyAvailable

Determines whether the Terminal Status/MAP plug-in treats a status of busy as a terminal status or terminal status change trigger.

Data type: Boolean

Allowed values: `true` and `false`

Default Value: `true`

Attribute name:

```
oracle.ocsg.plugin.terminal_status.map.policy.BusyAvailable
```

SLA Usage Restriction: MaximumNotificationAddresses

Specifies the maximum number of terminal addresses that a single Terminal Status/MAP plug-in request can contain.

Data type: Integer

Allowed Values: **1 ~ 2147483647 (2^31-1)**

Default Value: **2147483647**

Attribute name:

`oracle.ocsg.plugin.terminal_status.map.policy.MaximumNotificationAddresses`

SLA Usage Restriction: MaximumNotificationFrequency

Sets the maximum number of seconds allowed between Terminal Status/MAP plug-in requests.

Data type: Integer (represents seconds)

Allowed Values: **1~2147483647** (**2^31-1**)

Default Value: **2147483647**

Attribute name:

`oracle.ocsg.plugin.terminal_status.map.policy.MaximumNotificationFrequency`

SLA Usage Restriction: MaximumNotificationDuration

Sets the maximum time limit that a Terminal Status/MAP plug-in request can last. Cannot be used with SLA Usage Restriction: UnlimitedCountAllowed.

Data type: Integer (represents seconds)

Default Value: 600

Attribute name:

`oracle.ocsg.plugin.terminal_status.map.policy.MaximumNotificationDuration`

SLA Usage Restriction: MaximumCount

Specifies the maximum number of notifications that a single Terminal Status/MAP plug-in request can return. Cannot be used with SLA Usage Restriction: UnlimitedCountAllowed.

Data type: Integer

Allowed Values: **1~2147483647** (**2^31-1**)

Default Value: **2147483647**

Attribute name:

`oracle.ocsg.plugin.terminal_status.map.policy.MaximumCount`

SLA Usage Restriction: UnlimitedCountAllowed

Directs the Terminal Server/MAP plug-in to allow an unlimited number of notifications for each request. If this attribute is set to `true`, neither SLA Usage Restriction: MaximumNotificationDuration nor SLA Usage Restriction: MaximumCount should also be specified.

Data type: Boolean

Allowed Values: `true` and `false`

Default Value: `false`

Attribute name:

`oracle.ocsg.plugin.terminal_status.map.policy.UnlimitedCountAllowed`

11 Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

This chapter describes the Parlay X 3.0 Audio Call/Parlay 3.3 User Interaction and MultiParty Call Control communication service in detail.

Overview of the Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC Communication Service

The Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC communication service exposes the Parlay X 3.0 Audio Call set of application interfaces.

The communication service acts as an Open Services Architecture (OSA) Parlay application to an internal Services Gatekeeper OSA/Parlay Gateway. It uses this gateway to access the Call User Interaction and MultiParty Call Control SCSs. For information about the gateway, see "Managing OSA/Parlay Gateway Connections using Parlay_Access" in *System Administrator's Guide*.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using the Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC communication service, an application can:

- Play audio to one or more call participants in an existing call session set up by the communication service.
- Find out if the audio is currently being played or has not yet started to play.
- Explicitly end playing of the audio.
- Collect digits from call participants in response to an audio message that has been played to them and, in conjunction with the Parlay X 3.0 Call Notification/MPCC communication service, return the information to the application.
- Interrupt an ongoing interaction, such as on-hold music.

How It Works

The Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC communication service can be used by applications to play audio messages to one or more call participants in an existing call. The existing call is identified by the Call Session Identifier returned to the application at the time the call session is set up. If desired, applications can receive digits collected from those participants in response to the audio message using a notification set up using the communication service.

The audio message content to be played must be defined in a binary format such as WAV stored at a URL available to the network and rendered by an audio player. Services Gatekeeper does not actually render the message. This is the responsibility of equipment that must be present on the target telecom network, such as Interactive Voice Response (IVR) systems.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 3.0 Audio Call communication service, see the discussion of Parlay X 3.0 Interfaces in *Application Developer's Guide*.

Events and Statistics

The Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 11–1 lists the IDs of the EDRs created by the Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC communication service. This does not include EDRs created when exceptions are thrown.

Table 11–1 Event Types Generated by Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

EDR ID	Method Called
11100	playAudioMessage
11101	getMessageStatus
11102	endMessage
11103	startPlayAndCollectInteraction
11104	stopMediaInteraction
11105	sendInfoRes
11106	SendInfoErr
11107	sendInfoAndCollectRes
11108	SendInfoAndCollectErr
11109	attachMediaRes
111010	attachMediaErr
111011	detachMediaRes
111012	detachMediaErr
111013	abortActionRes
111014	abortActionErr

Charging Data Records

Audio Call/Parlay 3.3 UI-MPCC-specific CDRs are generated under the following conditions:

- When **sendInfoRes** is sent from the network to Services Gatekeeper, indicating that the audio message has completed playing, if this is not the result of an explicit request to stop from the application.
- When **sendInfoAndCollectRes** is sent from the network to Services Gatekeeper, indicating that the audio message has completed playing and the call participant's response has been collected in the form of digits.

Statistics

Table 11–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 11–2 Methods and Transaction Types for Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

Method	Transaction type
startPlayAndCollectInteractions	TRANSACTION_TYPE_CALL_CONTROL_SERVICE_INITIATED
playAudioMessage	TRANSACTION_TYPE_CALL_CONTROL_SERVICE_INITIATED

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Tunneled Parameters for Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

This section lists the parameters that can be tunneled or defined in the `<requestContext>` element of an SLA.

ac.parlay.sendInfoReq.repeatIndicator

Description

Specifies how many times an audio announcement is played to a call participant.

This setting overrides the value set in the OAM **RepeatIndicator** attribute.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format

String with a non-negative integer value

Value

A value of zero (0) indicates either that the announcement will be repeated until the call or call leg is released, either by an application or the network, or that the announcement will be ended by an application.

A positive value specifies the exact number of times that the announcements will be played.

Managing Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

This section describes the properties and workflow for the Parlay X 3.0 Audio Call/Parlay MultiParty Call Control and Call User Interaction plug-in instance.

Most of the configuration is done in the Open Services Architecture (OSA) Access module, but with configuration parameters for Parlay MultiParty Call Control. See "Managing OSA/Parlay Gateway Connections using Parlay_Access" in *System Administrator's Guide*.

This plug-in service is requires Orbacus, which is not installed by default. For information about installing Orbacus, see *Installation Guide*.

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one-to-one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

Properties for Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

Table 11–3 lists the technical specifications for the communication service.

Table 11–3 Properties for Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > Plugin_px30_audio_call_parlay_mpcc_cui

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=Plugin_px30_audio_call_parlay_sip
	Type=com.bea.wlcp.wlng.plugin.ac.parlay.management.AudioCallManagementMBean

Network protocol plug-in service ID	Plugin_px30_audio_call_parlay_mpcc_cui
Network protocol plug-in instance ID	Plugin_px30_audio_call_parlay_mpcc_cui
Supported Address Scheme	tel
Application-facing interfaces	com.bea.wlcp.wlng.px30.plugin.AudioCallPlayMediaPlugin com.bea.wlcp.wlng.px30.plugin.AudioCallCaptureMediaPlugin
Service type	AudioCall
Exposes to the service communication layer a Java representation of:	Parlay X 3.0 Part 11: Audio Call
Interfaces with the network nodes using:	Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control SCF; Subpart 7: MultiParty Call Control Service Open Service Access (OSA); Application Programming Interface (API); Part 5: User Interaction SCF
Deployment artifacts	osa_access.jar, Plugin_px30_audio_call_parlay_mpcc_cui.jar, and px30_audio_call_service.jar, packaged in wlng_nt_audio_call_px30.ear px30_audio_call.war, packaged in wlng_at_audio_call_px30.ear

Configuration Workflow for Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Select the MBean listed in the "Properties for Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC" section.
2. Configure the behavior of the plug-in instance using the MBean attributes. See "Reference: Attributes for Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC" for a list of the attributes and their settings.
3. Obtain the following information from your OSA Gateway administrator and configure the MultiParty Call Control part of the protocol translator (Parlay_Access service) accordingly:
   - OSA/Parlay SCS type to be used in the lookup (service discovery) phase when requesting the MultiParty Call Control service (OSA/Parlay SCS) from the OSA/Parlay Gateway. Typically this is P_MULTI_PARTY_CALL_CONTROL.
   - OSA/Parlay service properties to be used in the look up (service discovery) phase. This information is used to request a service (OSA/Parlay SCS) from the OSA/Parlay Gateway. These properties are specific to the OSA Gateway implementation.
   - Authentication type used by the OSA/Parlay Framework.
   - Encryption method used for the connection with the OSA Gateway.
   - The signing algorithm used when signing the service level agreement with the OSA/Parlay Framework.
4. Set up the OSA Client and the OSA Client Mappings for the MultiParty Call Control part of the plug-in instance. For information on how to do this, see "Creating an OSA client" and "Mapping the OSA client to an OSA Gateway and an OSA/Parlay SCS" in "Managing OSA/Parlay Gateway Connections using Parlay_Access" in *System Administrator's Guide*.
5. Gather the following information from your OSA Gateway administrator and configure the Call User Interaction part of the protocol translator (OSA Access service) accordingly:
   - OSA/Parlay SCS type to be used in the lookup (service discovery) phase when requesting the Generic User interaction service (OSA/Parlay SCS) from the OSA/Parlay Gateway. Typically this is P_USER_INTERACTION.
   - OSA/Parlay service properties to be used in the lookup (service discovery) phase when requesting a service (OSA/Parlay SCS) from the OSA/Parlay Gateway. This depends on the OSA Gateway implementation.
   - Authentication type used by the OSA/Parlay Framework.
   - Encryption method used for the connection with the OSA Gateway.
   - Signing algorithm used when signing the service level agreement with the OSA/Parlay Framework.
6. Set up the OSA Client and the OSA Client Mappings for the Generic User Interaction part of the plug-in instance. For information on how to do this, see "Creating an OSA client" and "Mapping the OSA client to an OSA Gateway and an OSA/Parlay SCS" in "Managing OSA/Parlay Gateway Connections using Parlay_Access" in *System Administrator's Guide*.
7. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.

   It is not necessary to set up routing rules to the plug-in instance.

Reference: Attributes for Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC

This section lists the attributes for configuration and maintenance.

- Attribute: CollectStartTimeout
- Attribute: CollectInterCharTimeout
- Attribute: Language
- Attribute: ChargingAllowed
- Attribute: RepeatIndicator
- Attribute: ResponseRequested
- Attribute: RetensionTime
- Attribute: ShutdownTimerInterval
- Attribute: MaxDigits
- Attribute: MinDigits
- Attribute: EndSequence

Attribute: ChargingAllowed

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether charging is allowed.

- `true` if an application is allowed to specify charging information when playing a message (Parlay X operation **playAudioMessage**).
- `false` if not.

Attribute: CollectInterCharTimeout

Scope: Cluster

Unit: Seconds

The inter-character timeout when collecting user input. Also known as the value for the inter-character timeout timer.

This value corresponds to the **TpUICollectCriteria.InterCharTimeOut** parameter in **sendInfoAndCollectReq** requests to the Generic User Interaction SCS.

Attribute: CollectStartTimeout

Scope: Cluster

Unit: Seconds

The start timeout period for collecting user input. Specifies how much time is allowed for the user to enter the first character.

The value corresponds to the **TpUICollectCriteria.StartTimeout** parameter in **sendInfoAndCollectReq** requests to the Generic User Interaction SCS.

Attribute: EndSequence

Scope: Server

Unit: Not applicable

Format: String

The digit to be used for ending collection of data of various lengths from a call participant.

The value corresponds to the **TpUICollectCriteria.EndSequence** parameter in **sendInfoAndCollectReq** requests to the Generic User Interaction SCS.

Attribute: Language

Scope: Cluster

Unit: Not applicable

Format: String according to valid language strings as defined in ISO 639.

The language of the message to be played for the call participant.

Attribute: MaxDigits

Scope: Server

Unit: Not applicable

Format: Positive integer

The maximum number of digits that can be collected from the call participant.

The value corresponds to the **TpUICollectCriteria.MaxLength** parameter in **sendInfoAndCollectReq** requests to the Generic User Interaction SCS.

Valid range is **1–65535**.

Attribute: MinDigits

Scope: Server

Unit: Not applicable

Format: Positive integer

The minimum number of digits that can be collected from the call participant.

The value corresponds to the **TpUICollectCriteria.MinLength** parameter in **sendInfoAndCollectReq** requests to the Generic User Interaction SCS.

Valid range is **1–65535**.

Attribute: RepeatIndicator

Scope: Cluster

Unit: Not applicable

Format: Integer

The number of times a message should be played to the call participant.

The value corresponds to the **repeatIndicator** parameter in **sendInfoReq** requests to the Generic User Interaction SCS.

Attribute: ResponseRequested

Scope: Cluster

Unit: Not applicable

Format: Integer [**1**,**2**,**4**]

Specifies whether a response is required from the Generic User Interaction SCS, and what, if any, action the service should take.

The value corresponds to the **responseRequested** parameter in **sendInfoReq** requests to the Generic User Interaction SCS.

The valid values are:

- **1** for P_UI_RESPONSE_REQUIRED
- **2** for P_UI_LAST_ANNOUNCEMENT_IN_A_ROW
- **4** for P_UI_FINAL_REQUEST

Attribute: RetensionTime

Scope: Cluster

Unit: Seconds

Format: Integer

The time interval for which status information is retained after a message is played or an error occurs.

Attribute: ShutdownTimerInterval

Scope: Server

Unit: Seconds

Format: Integer

The time interval to wait for call sessions to end before terminating. Used for performing a graceful shutdown.

12 Parlay X 3.0 Call Notification/Parlay 3.3 MPCC

This chapter describes the Parlay X 3.0 Call Notification/Parlay 3.3 Multi-Party Call Control (MPCC) communication service in detail.

Overview of the Parlay X 3.0 Call Notification/Parlay 3.3 MPCC Communication Service

Parlay X 3.0 Call Notification/Parlay 3.3 MPCC communication service exposes the Parlay X 3.0 Call Notification set of application interfaces.

The communication service acts as an Open Services Architecture (OSA) Parlay application to an internal Services Gatekeeper OSA/Parlay Gateway. It uses this gateway to access the MultiParty Call Control SCS. For information about the gateway, see "Managing OSA/Parlay Gateway Connections using Parlay_Access" in *System Administrator's Guide*.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using the Parlay X 3.0 Call Notification/Parlay 3.3 MPCC communication service, an application can:

- Set up and tear down notifications on call events for a given combination of caller and callee.
- Receive notifications on call events related to established notifications.
- Interact with the functionality of other communication services, including Audio Call to play audio to call participants or to collect data from them or Third Party Call to reroute the call or to set up additional call legs.
- End the call.

The operations made available by this communication service are concerned only with monitoring (and, in some cases, making certain changes to) calls during the setup phase. By itself, this communication service is not used to set up new calls, only to reroute or terminate calls already in progress.

How It Works

For an application to receive notifications about call setup attempts from the network, it must register its interest in these notifications by setting up a subscription in Services

Gatekeeper. A subscription, or a notification, is defined by a set of addresses and a set of criteria. The criteria define the events in which the application is interested.

The addresses may be translated by some mechanism in the telecom network prior to reaching Oracle Communications Services Gatekeeper

Two types of notifications exist:

- Monitoring
- Monitoring and rerouting

Monitoring

An application can register to be notified about the following events as the call between the caller and the callee is set up:

- The callee is busy.
- The callee is not reachable.
- The callee does not answer.
- The caller is attempting to call the callee.
- The callee has answered the call.

Note: These notifications may include a Call Session Identifier identifying the call session in the network, if available, to allow interactions with other Parlay X Web Services, such as Third Party Call and Audio Call. These interactions tend to be asynchronous.

- A call participant has interacted with a play-and-collect-media event. The notification contains the results of the interaction, including the digits collected.
- A call participant has interacted with a play-and-record-media event. The notification contains the results of the interaction, including the location of the recorded information.

Setting up a notification for a play and record media event is supported, but setting up the play and record interaction is not supported in the Parlay X 3.0 Audio Call/Parlay 3.3 MPCC communication service in this version.

Monitoring and rerouting

In addition to monitoring the state of call setup, an application can also choose to make certain changes to the call under certain conditions, in a synchronous manner. In the case of certain monitored events (busy, not reachable, no answer, call attempt), an application can specify how to handle them, including:

- Continue to let the call be managed by the network in the normal manner, by, for example, playing a busy tone.
- End the call.
- Intercept the call setup attempt between the caller and the callee and reroute the call to another callee (C-party) without making an attempt to connect with the callee (B-party). An example might be a general technical support number that is routed to the appropriate call center based on time of day.

If the call is rerouted, the media type is always negotiated by the underlying network. The **MediaInfo** parameter is not currently used by the communication service.

Because this communication service handles traffic in two directions (from the application to the network and from the network to the application) its functionality has some aspects of both the application-initiated and the network-triggered types. The communication service itself manages only the signalling, or controlling, aspect of the call. The call itself, the media, or audio, channel, is completely handled by the underlying telecom network.

Because the communication service manages only the signalling aspect of the call, only parties residing on the same network can be controlled, unless:

- The network plug-in connects to a media gateway controller.
- One of the participants is connected to a signalling gateway so that, from a signalling point of view, all parties reside on the same network.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 3.0 Call Notification communication service, the discussion of Parlay X 3.0 Interfaces in *Application Developer's Guide*.

Events and Statistics

The Parlay X 3.0 Call Notification/Parlay 3.3 MPCC communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 12–1 lists IDs of the EDRs created by the Parlay X 3.0 Call Notification/Parlay 3.3 MPCC communication service. This does not include EDRs created when exceptions are thrown.

Table 12–1 Event Types Generated by Parlay X 3.0 Call Notification/Parlay 3.3 MPCC

EDR ID	Method Called
11000	startCallDirectionNotification
11001	stopCallDirectionNotification
11002	startCallNotification
11003	stopCallNotification
11004	startPlayAndCollectNotification
11006	stopMediaInteractionNotification
11007	reportNotification
11008	deleteNotification
11009	createNotification
11011	sendInfoAndCollectRes

Charging Data Records

Parlay X 3.0 Call Notification/Parlay 3.3 MPCC-specific CDRs are generated under the following conditions:

- After a **reportNotification** is sent from the Parlay gateway to Services Gatekeeper, indicating that a call event defined by the notification has occurred and (in appropriate cases) needs to be handled
- After a **sendInfoandCollectRes** has been sent from the Parlay gateway to Services Gatekeeper, indicating that a call participant has interacted with a play-and-collect operation; the response includes the digits collected.

Statistics

Table 12–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 12–2 Methods and Transaction Types for Parlay X 3.0 Call Notification/Parlay 3.3 MPCC

Method	Transaction Type
reportNotification (both CallNotification and CallDirection)	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED
sendInfoAndCollectRes (callNotification only)	TRANSACTION_TYPE_CALL_CONTROL_ NETWORK_INITIATED

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing Parlay X 3.0 Call Notification/Parlay 3.3 MPCC

This section describes the properties and workflow for the Parlay X 2.1 Call Notification/Parlay 3.3 MPCC plug-in instance.

Most of the configuration is done in the OSA Access module, but with configuration parameters for Parlay MultiParty Call Control. See "Managing OSA/Parlay Gateway Connections using Parlay_Access" in S*ystem Administrator's Guide*.

Two different types of notifications are managed:

- Regular notifications, started by an application invoking **startCallNotification** or **startCallDirectionNotification**
- Media notifications, started by an application invoking **startPlayAndCollectNotification**

This plug-in service is requires Orbacus, which is not installed by default. For information about installing Orbacus, see *Installation Guide*.

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one-to-one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

Properties for Parlay X 3.0 Call Notification/Parlay 3.3 MPCC

Table 12–3 lists the technical specifications for the communication service.

Table 12–3 Properties for Parlay X 3.0 Call Notification/Parlay 3.3 MPCC

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services->Plugin_px30_call_notification_parlay_mpcc

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=Plugin_px30_cn_parlay
	Type=com.bea.wlcp.wlng.px30.plugin.callnotification.parlay.management.mbean.CallNotificationMBean

Network protocol plug-in service ID	Plugin_px30_call_notification_parlay_mpcc
Network protocol plug-in instance ID	Plugin_px30_call_notification_parlay_mpcc
Supported Address Scheme	tel
Application-facing interfaces	com.bea.wlcp.wlng.px30.plugin.CallNotificationManagerPlugin com.bea.wlcp.wlng.px30.plugin.CallDirectionManagerPlugin com.bea.wlcp.wlng.px30.callback.CallDirectionCallback com.bea.wlcp.wlng.px30.callback.CallNotificationCallback
Service type	CallNotification
Exposes to the service communication layer a Java representation of:	Parlay X 3.0 Part 3: Call Notification
Interfaces with the network nodes using:	Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control SCF; Subpart 7: MultiParty Call Control Service
Deployment artifacts	px30_callnotification_parlay.jar, packaged in wlng_nt_call_notification_px30.ear px30_call_notification.war and px30_call_notification_callback.jar, packaged in wlng_at_call_notification_px30.ear

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one-to-one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

Configuration Workflow for Parlay X 3.0 Call Notification/Parlay 3.3 MPCC

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Using the Administration Console or an MBean browser, select the MBean listed in the "Properties for Parlay X 3.0 Call Notification/Parlay 3.3 MPCC" section.
2. Gather information about the OSA/Parlay Gateway and configure the protocol translator accordingly. The following information needs to be obtained from the OSA/Parlay Gateway administrator and configured in the Parlay Access service:
   - OSA/Parlay SCS type to be used in the look up (service discovery) phase when requesting the MultiParty Call Control service (OSA/Parlay SCS) from

the OSA/Parlay Gateway. Typically this is P_MULTI_PARTY_CALL_CONTROL.

- OSA/Parlay service properties to be used in the look up (service discovery) phase when requesting a service (OSA/Parlay SCS) from the OSA/Parlay Gateway. This depends on the OSA Gateway implementation.
- Authentication type used by the OSA/Parlay Framework.
- Encryption method used for the connection with the OSA Gateway.
- Signing algorithm used when signing the service level agreement (SLA) with the OSA/Parlay Framework.

3. Set up the OSA/Parlay Client and the OSA/Parlay Client Mappings. See "Managing OSA/Parlay Gateway Connections using Parlay_Access" in *System Administrator's Guide*.
4. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 3.0 Call Notification/Parlay 3.3 MPCC" section.
5. If desired, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
6. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Reference: Operations for Parlay X 3.0 Call Notification/Parlay 3.3 MPCC

This section describes operations for configuration and maintenance:

- Operation: deleteMediaNotification
- Operation: deleteNotification
- Operation: getMediaNotification
- Operation: getNotification
- Operation: listNotifications
- Operation: listMediaNotifications

Operation: deleteMediaNotification

Scope: Cluster

Deletes a media notification.

Signature:

```
deleteMediaNotification(correlator: String)
```

Table 12–4 deleteMediaNotification Parameters

Parameter	Description
correlator	ID for the subscription. Given by an application when the subscription was started.

Operation: deleteNotification

Scope: Cluster

Deletes a notification from the storage and removes it from OSA Gateway.

Signature:

```
deleteNotification(correlator: String)
```

Table 12–5 deleteNotification Parameters

Parameter	Description
correlator	ID for the subscription. Given by an application when the subscription is started.

Operation: getMediaNotification

Scope: Cluster

Displays information about a media notification. The information includes:

- Parlay X correlator
- Parlay X callSessionIdentifier
- needNotify This is an internal field. If `true`, the plug-in instance needs to invoke the Call Notification callback method **sendInfoAndCollectRes**.
- Parlay IpAppUICallRef (CORBA IOR)
- Parlay X endPoint to where the notification is sent
- Data about the owner of the notification:
  - Service provider account ID
  - Application account ID
  - Application instance ID
  - notifMode, used to distinguish which kind of media notification the notification belongs to. This is the Parlay X operation that created the notification:
    - **1** = The notification was created using **startPlayAndCollectInteraction**.
    - **2** = The notification was created using **startPlayAndRecordInteraction**.

Signature:

```
getMediaNotification(correlator: String)
```

Table 12–6 getMediaNotification Parameters

Parameter	Description
correlator	ID for the subscription. Given by an application when the subscription is started.

Operation: getNotification

Scope: Cluster

Displays information about a notification. The information includes:

- Parlay X Correlator

- Parlay X Endpoint where notifications are sent
- List of Parlay notification IDs associated with the Parlay X notification. There is one for each called party address for which notifications should be triggered supplied in the Parlay X operations **startCallNotification** or **startCallDirectionNotification.**
- Data about the owner of the notification:
  - Service provider account ID
  - Application account ID
  - Application instance ID

Signature:

```
getNotification(correlator: String)
```

Table 12–7 getNotification Parameters

Parameter	Description
correlator	ID of the notification. Given by an application when the notification is started.

Operation: listMediaNotifications

Scope: Cluster

Displays all active media notifications. These are notifications that have been registered by an application using:

- **startPlayAndCollectNotification**
- **startPlayAndRecordNotification**

Returns a list of correlators that uniquely identify each notification.

Signature:

```
listMediaNotifications()
```

Operation: listNotifications

Scope: Cluster

Displays all active call notifications. These are notifications registered by an application using:

- **startCallDirectionNotification**
- **startCallNotification**

Returns a list of correlators that uniquely identify each notification.

Signature:

```
listNotifications()
```

13 Parlay X 3.0 Device Capabilities/LDAPv3

This chapter describes the Parlay X 3.0 Device Capabilities/Lightweight Directory Access Protocol version 3 (LDAPv3) communication service in detail.

Overview of the Parlay X 3.0 Device Capabilities/LDAPv3 Communication Service

The Device Capabilities/LDAPv3 communication service exposes the Parlay X 3.0 Device Capabilities and Configuration set of application interfaces.

The communication service acts as an LDAP client to a directory service, connecting to the directory service using the LDAPv3.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

The Parlay X 3.0 Device Capabilities/LDAPv3 communication service sends requests to any LDAPv3-compliant directory server with a device's address (usually a phone number), and in return receives one of the following device identifiers:

- The device's unique device ID, device or model name, and a link to the User Agent Profile XML file.
- The device's equipment identifier (for example, its IMEI)

Application Interfaces

For information about the SOAP-based interface for the Parlay X 3.0 Device Capabilities communication service, see the discussion of Parlay X 3.0 Interfaces in *Application Developer's Guide*.

For information about the RESTful Call Notification interface, see the discussion of Device Capabilities in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on, they are the same.

Events and Statistics

The Parlay X 3.0 Device Capabilities/LDAPv3 communication service generates Event Data Records (EDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 13–1 lists the IDs of the EDRs created by the Device Capabilities/LDAPv3 communication service. This list does not include EDRs created when exceptions are thrown.

Table 13–1 EDRs Generated by Parlay X 3.0 Device Capabilities/LDAPv3

EDR ID	Method Called
403001	getCapabilities
403002	getDeviceId

Charging Data Records

The Device Capabilities/LDAPv3 communication service does not generate any CDRs by default.

Statistics

Table 13–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 13–2 Methods and Transaction Types for Parlay X 3.0 Device Capabilities/LDAPv3

Method	Transaction Type
getCapabilities	TRANSACTION_TYPE_OTHER
getDeviceId	TRANSACTION_TYPE_OTHER

Managing Parlay X 3.0 Device Capabilities/LDAPv3

This section describes the properties and workflow for the Parlay X 3.0 Device Capabilities/LDAPv3 plug-in instance.

It also includes a description of how to create an LDAP-to-XML mapping file.

Properties for Parlay X 3.0 Device Capabilities/LDAPv3 Plug-in

Table 13–3 lists the technical specifications for the communication service.

Table 13–3 Properties for Parlay X 3.0 Device Capabilities/LDAPv3

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plug-in_instance_id*

MBean	Domain=oracle.ocsg.plugin.dc.ldap.management
	Name=wlng_nt_device_capabilities_px30
	InstanceName=Device_cap
	Type=oracle.ocsg.plugin.dc.ldap.management.DeviceCapabilitiesLdapMBean

Network protocol plug-in service ID	Plugin_px30_decvice_capabilities_ldap

Table 13–3 (Cont.) Properties for Parlay X 3.0 Device Capabilities/LDAPv3

Property	**Description**
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
Supported Address Formats	tel, id, imsi, ipv4
Application-facing interface	com.bea.wlcp.wlng.px30.plugin.DeviceCapabilitiesPlugin
Service type	DeviceCapabilities
Exposes to the service communication layer a Java representation of:	Device Capabilities/LDAP
Interfaces with the network nodes using:	LDAP
Deployment artifact NT EAR wlng_nt_device_capabilities_px30.ear	px30_device_capabilities.jar and Plugin_px30_device_capabilities_ldap.jar.
Deployment artifact AT EAR: SOAP Only wlng_at_device_capabilities_px30_soap.ear	Ipx30_device_capabilities.war
Deployment artifact AT EAR: wlng_at_device_capabilities_px30.ear	px30_device_capabilities.jar and Plugin_px30_device_capabilities_ldap.jar

Configuration Workflow for Device Capabilities/LDAPv3 Plug-in

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID as listed in the "Properties for Parlay X 3.0 Device Capabilities/LDAPv3 Plug-in" section.
2. Using the Administration Console or an MBean browser, select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID given when the plug-in instance was created.
3. Define the characteristics of the LDAP server to connect to using these attributes:
   - Attribute: Port
   - Attribute: BaseDN
   - Attribute: AuthDN
   - Attribute: AuthPassword
4. Using "Attribute: Schema", define the XML schema.

See "Creating an LDAP-to-XML Mapping File" for a description of the schema and "Configuration Workflow for Device Capabilities/LDAPv3 Plug-in" for a description of the mappings.

5. Define the connection pool characteristics for the connection:
   - Attribute: MinConnections
   - Attribute: MaxConnections
   - Attribute: ConnTimeout
6. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 3.0 Device Capabilities/LDAPv3 Plug-in" section.
7. If required, create and load a node SLA. For details, see “Defining Global Node and Service Provider Group Node SLAs” and “Managing SLAs” in *Accounts and SLAs Guide*.
8. Provision the service provider and application accounts. For information, see *Accounts and SLAs Guide*.

Creating an LDAP-to-XML Mapping File

You can create multiple Device Capabilities/LDAPv3 plug-in instances, each with a different LDAP configuration. Each plug-in instance could point to a different LDAP tree or even a different LDAP server.

Each Device Capabilities/LDAPv3 plug-in instance routes requests to an LDAP stack (LDAPJDK 4.1). The LDAP library (physical connection) is specified using the **instanceId** field. The LDAP stack is included as a library in the network tier EAR package.

The LDAP library must have the device capabilities (**Name**, **agentProfileRef**, and **deviceId** (IMEI)) stored as attributes in a single LDAP entry indexed by address. You can redirect a plug-in to a different LDAPv3 library by specifying a new Distinguished Name (DN) and schema as long as the device capabilities are all available from a single LDAP entry.

An XSD schema that you create maps the URI format (for example, tel: or imsi:) to an associated query string; this file does not affect the LDAP database.

You need to map the Device Capabilities communication service SOAP request data to an LDAP query string that matches the subscriber information in your LDAP directory. You do this by defining an XML file to map the data and an XSD schema to validate the XML.

Example 13–1 shows a sample LDAP query XSD schema for the sample XML data shown in Example 13–2. This XML file maps the tel:1234 address to msisdn=1234,domainName=msisdnD. The resulting LDAP query for this example is:

(&(msisdn=1234)(objectClass=*))

in domainName=msisdnD,%Base DN% .

The Base DN is configured using Attribute: BaseDN.

Example 13–1 LDAP Query XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<xs:element name="LdapConfig">
<xs:complexType>
<xs:sequence>
<xs:element name="Keys" type="KeySet" minOccurs="1" maxOccurs="unbounded"/>
<xs:element name="LdapObject" type="LdapObject" minOccurs="1"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType name="KeyObject">
<xs:sequence>
<xs:element name="uriScheme" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="addressKeyName" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="objectKeyName" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="objectKeyValue" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="optional"/>
</xs:complexType>

<xs:complexType name="KeySet">
<xs:sequence>
<xs:element name="Key" type="KeyObject" minOccurs="1" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="LdapObject">
<xs:sequence>
<xs:element name="ObjectKeySet" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="keyName" type="xs:string" use="required"/>
<xs:attribute name="keyValue" type="xs:string" use="required"/>
</xs:complexType>
</xs:schema>
```

Example 13–2 shows sample XML data that matches the LDAP query XSD file in Example 13–1.

Example 13–2 Sample XML Data

```
<?xml version="1.0" encoding="UTF-8"?>
<LdapConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation='sp_config.xsd'>
    <Keys id="sample">
        <Key>
            <uriScheme>tel</uriScheme>
            <addressKeyName>msisdn</addressKeyName>
            <objectKeyName>domainName</objectKeyName>
            <objectKeyValue>msisdnD</objectKeyValue>
        </Key>
    </Keys>
   </LdapConfig>
```

You need to create your own LDAP query XSD file to map your LDAP SOAP request elements to your LDAP database elements. The LDAP query XSD file must define the following objects based on their elements, listed in Table 13–4:

- LdapObject: A KeySet holder.

- KeySet: A collection of KeyObjects. Sets of keys are used because there may be several ways to reach a certain node in the tree. One LDAP plug-in instance can be configured with several KeySets and can provide the link between the search key in the Extended Web Services interface and the LDAP tree.
- KeyObject: An entry point to the LDAP tree that provides the link between the search key in the Extended Web Services interface and the LDAP tree.

Table 13–4 LDAP Server Schema

Object	Element	Description
LdapObject	ObjectKeySet	Defines the KeySet through which it can be reached. Refers to theID attribute of a defined KeySet.
LdapObject	id	The identity of the LdapObject. Can be referenced from other LdapObjects through the ParentObjectId field.
LdapObject	keyName	The name of the key through which the LdapObject can be reached.
LdapObject	keyValue	The value of the key through which the LdapObject can be reached.
KeyObject	uriScheme	Defines the URI scheme of the address for which this key applies.
KeyObject	addressKeyName	Defines the key name with which the address value is associated.
KeyObject	objectKeyName	Provides the possibility of defining the addressing key of a possible tree node above the node that is reached by the address key (that is, like the domain object in the 3DS directory information tree).
KeyObject	objectKeyValue	See objectKeyName. Defines the value of the key.
KeyObject	id	The identity of the key. Used only for descriptive purposes.
KeySet	Key	All keys in the KeySet
KeySet	id	The identity of the KeySet. Used when associating an LdapObject with a KeySet.

Reference: Attributes and Operations for Device Capabilities/LDAPv3

This section describes the attributes and operations for configuration and maintenance:

- Attribute: AuthDN
- Attribute: AuthPassword
- Attribute: BaseDN
- Attribute: ConnTimeout
- Attribute: DeviceIdAttributeName
- Attribute: DeviceNameAttributeName
- Attribute: DeviceProfileURLAttributeName
- Attribute: Host

Attribute: AuthDN

Scope: Cluster

Format: String

Specifies a Distinguished Name (DN) in the LDAP server.

Use "Operation: apply" to make changes to this attribute take effect.

Example:

```
cn=admin,o=acompany,c=uk
```

Attribute: AuthPassword

Scope: Cluster

Format: String

Specifies the password associated with theAttribute: AuthDN.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: BaseDN

Scope: Cluster

Format: String

Specifies the base Distinguished Name (DN) for the LDAP database in use.

Use "Operation: apply" to make changes to this attribute take effect.

Example:

```
o=acompany,c=uk
```

Attribute: ConnTimeout

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the maximum time to wait for an LDAP connection to be established. If the related timer expires, a retry is performed. See "Attribute: recoverTimerInterval" for more information.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: DeviceIdAttributeName

Scope: Cluster

Format: String

Specifies the **DeviceId** of the target LDAP entry.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: DeviceNameAttributeName

Scope: Cluster

Format: String

Specifies the **DeviceName** of the target LDAP entry.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: DeviceProfileURLAttributeName

Scope: Cluster

Format: String

Specifies the **DeviceProfileURL** of the target LDAP entry.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: Host

Scope: Cluster

Format: String

Specifies the host name or IP address of the LDAP server to connect to.

Use "Operation: apply" to make changes to this attribute take effect.

Examples:

```
myldapserver.mycompany.org
192.168.0.14
```

Attribute: LDAPConnectionStatus

Read-only.

Scope: Local

Unit: Not applicable

Values: **active**, **update_pending**, or **deactive**. Table 13–5 describes each of these values and their implications.

Format: String

Table 13–5 LDAP Server Connection Status

Status	Description
active	The connection is active. The plug-in instance accepts requests.
update_pending	The connection is temporarily unavailable due to an update of the configuration settings. The plug-in instance does not accept requests.

Table 13–5 (Cont.) LDAP Server Connection Status

Status	Description
deactive	The connection is inactive. The plug-in instance does not accept requests. Reasons for this entering this state include: ■ Missing or incorrect configuration ■ LDAP server is unreachable ■ Internal errors

Use Operation: apply to make changes to this attribute take effect.

Attribute: MaxConnections

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the maximum number of connections in the LDAP connection pool.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: MinConnections

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the minimum number of connections to establish using connections from the LDAP connection pool.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: Port

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the port number of the LDAP server to connect to.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: recoverTimerInterval

Scope: Cluster

Format: Integer

Unit: Seconds

Default Value: 300

Specifies the time to wait before performing an LDAP connection retry after an LDAP connection error. Should be at least twice the time defined in the **ConnTimeout** attribute. See "Attribute: ConnTimeout" for more information.

Use "Operation: apply" to make changes to this attribute take effect.

Attribute: Schema

Scope: Cluster

Unit: Not applicable

Format: String

The LDAP schema to use.

Use "Operation: apply" to make changes to this attribute take effect.

Operation: apply

Scope: Cluster

Applies attribute changes.

Signature:

```
apply()
```

Operation: updateSchemaURL

Scope: Cluster

Format: String

Updates the schema URL to use when performing lookups in the LDAP database.

During the update, the LDAP connection is temporarily unavailable and the connection status is **update_pending**. See Table 13–5, " LDAP Server Connection Status" for more information.

Signature:

```
pdateSchemaURL(SchemaURL:String)
```

Table 13–6 explains that the **schemaURL** parameter is the LDAP database schema URL to use.

Table 13–6 updateSchemaURL Parameters

Parameter	Description
SchemaURL	The LDAP database schema URL. Examples: Windows: file:///d:/ldap/schema.xml UNIX: file://ldap/schema.xml

14 Parlay X 3.0 Payment/Diameter

This chapter describes the Parlay X 3.0 Payment/Diameter communication service in detail.

Overview of the Parlay X 3.0 Payment Communication Service

The Parlay X 3.0 Payment/Diameter communication service exposes the Parlay X 3.0 Payment set of application interfaces.

The communication service acts as a credit control client to a credit control server using the Diameter protocol.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using a Payment communication service, an application can:

- Charge and refund accounts directly.
- Operate on reservations, which includes:
  - Make reservations.
  - Charge reservations.
  - Release reservations.
- Charge multiple accounts concurrently.

All charging is done on accounts. The unit of charge is specified as a given currency or a charging code.

A reservation expires after a given time. An expiration mechanism provided by the Storage Service is used. If the store entry expires, the reservation is cancelled.

Some Diameter servers, for example Oracle Billing and Revenue Management, mandate that a refund operation be correlated with a previous charge operation. The Payment interface does not provide any correlation between charge operations and refund operations. The **session-id** tunneled parameter has been added in order to correlate these requests. When an application calls **chargeAmount**, the tunneled parameter **session-id** is returned in the SOAP header. An application should use this **session-id** in subsequent **refundAmount** requests to correlate the two requests. If the application does not provided the tunneled parameter, it is the responsibility of the Diameter server to either accept or deny the request. If the request is denied, the application receives a ServiceException. See "session-id" for more information.

Processing Direct Queries/Application-initiated Requests

If an application makes a request to interact directly with an account, Services Gatekeeper sends the request to the network node capable of handling the request. The request does not return until the targeted account has been updated.

Processing Notifications/Network-triggered Requests

There are no notifications or other network-triggered requests for this communications service.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 3.0 Payment communication service, see the discussion of Parlay X 3.0 Interfaces in *Application Developer's Guide*.

For information about the RESTful Call Notification interface, see the discussion of Payment in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on., they are the same.

Events and Statistics

The Parlay X 3.0 Payment/Diameter communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 14–1 lists Event Data Record (EDR) IDs created by the Payment/Diameter communication service.

Table 14–1 Event Types Generated by Parlay X 3.0 Payment/Diameter

EDR ID	Method Called
15001	chargeAmount
15002	refundAmount
15003	chargeSplitAmount
15004	reserveAmount
15005	reserveAdditionalAmount
15006	chargeReservation
15007	releaseReservation

Charging Data Records

Payment/Diameter communication service-specific CDRs are generated when the response to a request is successfully delivered to the application. All of the following operations trigger the generation of CDRs:

- **chargeAmount**
- **refundAmount**
- **chargeSplitAmount**
- **reserveAmount**
- **reserveAdditionalAmount**
- **chargeReservation**
- **releaseReservation**

Statistics

Table 14–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counter.

Table 14–2 Methods and Transaction Types for Parlay X 3.0 Payment/Diameter

Method	Transaction type
chargeAmount	TRANSACTION_TYPE_CHARGING_DIRECT
chargeSplitAmount	TRANSACTION_TYPE_CHARGING_DIRECT
refundAmount	TRANSACTION_TYPE_CHARGING_DIRECT
reserveAmount	TRANSACTION_TYPE_CHARGING_RESERVED_STRING
reserveAdditionalAmount	TRANSACTION_TYPE_CHARGING_RESERVED_STRING
chargeReservation	TRANSACTION_TYPE_CHARGING_RESERVED_STRING

Tunneled Parameters for Parlay X 3.0 Payment / Diameter

This section lists the parameters that can be tunneled.

session-id

Description

Correlates a **refundAmount** operation with a **chargeAmount** operation.

Some billing systems, including Oracle Billing and Revenue Management, allow refund operations only on previously charged amounts. Parlay X does not have the ability to correlate charge and refund operations. This parameter provides that functionality.

The key and the value are available in the return message from a **chargeAmount** operation. It is the responsibility of the application to provide the key and the value in subsequent **refundAmount** operations to correlate the two.

If no session-id is provided in the request to the Diameter node, the Diameter node can either accept or deny the request. If the node denies the request, a ServiceException is sent back to the application.

Format

String

Example

This is an example in a SOAP header:

```
<xparams> <param key=" session-id value="12233187769"/>  </xparams>
```

Managing Parlay X 3.0 Payment /Diameter

This section describes the properties and workflow for the Parlay X 3.0 Payment/Diameter plug-in instance.

Properties for Parlay X 3.0 Payment/Diameter

Table 14–3 lists the technical specifications for the communication service.

Table 14–3 Properties for Parlay X 3.0 Payment/Diameter

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plugin_instance_id*

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=same as the network protocol *instance_id* assigned when the plug-in instance is created
	Type=com.bea.wlcp.wlng.plugin.payment.diameter.management .PaymentMBean

Network protocol plug-in service ID	Plugin_px30_payment_diameter
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
Supported Address Scheme	tel
Application-facing interfaces	com.bea.wlcp.wlng.px30.plugin.AmountChargingPlugin com.bea.wlcp.wlng.px30.plugin.ReserveAmountChargingPlugin
Service type	Payment
Exposes to the service communication layer a Java representation of:	Parlay X 3.0 Part 6: Payment
Interfaces with the network nodes using:	Diameter RFC3588 and RFC 4006
Deployment artifact NT EAR wlng_nt_payment_ px30.ear	Plugin_px30_payment_diameter.jar and px30_payment_ service.jar
Deployment artifact AT EAR: Normal wlng_at_payment_ px30.ear	rest_payment.war and px30_payment.war

Table 14–3 (Cont.) Properties for Parlay X 3.0 Payment/Diameter

Property	Description
Deployment artifact AT EAR: SOAP Only wlng_at_payment_px30_soap.ear	px30_payment.war

Configuration Workflow for Parlay X 3.0 Payment/Diameter

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
1. Use the plug-in service ID as listed in the "Properties for Parlay X 3.0 Payment/Diameter" section.
2. Using the Administration Console or an MBean browser, select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID given when the plug-in instance was created.
3. Configure the behavior of the plug-in instance:
   - Attribute: Connected
   - Attribute: Domain
   - Attribute: DestinationHost
   - Attribute: DestinationPort
   - Attribute: DestinationRealm
   - Attribute: OriginHost
   - Attribute: OriginPort
   - Attribute: OriginRealm
4. Use "Operation: connect" to connect to the Diameter server.
5. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Parlay X 3.0 Payment/Diameter" section.
6. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
7. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Provisioning Workflow for Parlay X 3.0 Payment/Diameter

The Parlay X 3.0 Payment/Diameter plug-in instance can be explicitly connected to the Diameter server. It does not connect to the server by default. The service has a connection status that will be preserved after service redeployment and server restart.

Use:

- Operation: connect

- Operation: disconnect

Use "Operation: connect" after any changes to the configuration attributes. Changes does not take affect until this operation is invoked.

Reference: Attributes and Operations for Parlay X 3.0 Payment/Diameter

This section describes the attributes and operations for configuration and maintenance:

- Attribute: Connected
- Attribute: DestinationHost
- Attribute: DestinationPort
- Attribute: DestinationRealm
- Attribute: Domain
- Attribute: Domain
- Attribute: OriginHost
- Attribute: OriginRealm
- Attribute: Service-Context-Id
- Operation: connect
- Operation: disconnect

Attribute: Connected (read-only)

Scope: Server

Format: Boolean

Unit: Not applicable

Displays the status of the connection to the Diameter server.

Displays:

- `true`, if connected.
- `false`, if not connected.

Attribute: Connected

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the value of the Destination-Host AVP in Diameter requests.

Example:

host.destination.com

Attribute: DestinationHost

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the host name of the Diameter server to connect to.

Example:

host.destination.com

Attribute: DestinationPort

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the port on the Diameter server to connect to.

Example:

3588

Attribute: DestinationRealm

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the Destination-Realm AVP in Diameter requests.

Example:

destination.com

Attribute: Domain

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the Service-Context ID AVP.

The default is **oracle.com**.

Attribute: OriginHost

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the Origin-Host AVP in Diameter requests.

Example:

host.oracle.com

Attribute: OriginPort

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the local originator port to be used for the connection to the Diameter server.

Example:

7002

Attribute: OriginRealm

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the Origin-Realm AVP in Diameter requests.

Example:

oracle.com

Attribute: Service-Context-Id

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the value of the Service-Context-Id AVP in Diameter requests. Used to override the oracle.com suffix in the ServiceContextId value to work with third-party charging applications.

Example:

S**CAP_V.2.0@xyz.com**

Operation: connect

Scope: Cluster

Connects to the Diameter server.

Once connected, the service will try to reconnect to the Diameter server if the server is restarted or the plug-in is redeployed.

Signature:

```
connect()
```

Operation: disconnect

Scope: Cluster

Disconnects from the Diameter server.

Once disconnected, the plug-in will not try to reconnect to the Diameter server if the server is restarted or the plug-in is redeployed.

Signature:

```
disconnect()
```

15

Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC

This chapter describes the Parlay X 3.0 Third Party Call/Parlay 3.3 Multi-Party Call Control (MPCC) communication service in detail.

Overview of the Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC Communication Service

The Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC communication service exposes the Parlay X 3.0 Third Party Call set of application interfaces.

The communication service acts as an Open Services Architecture (OSA) Parlay application to an internal Services Gatekeeper OSA/Parlay Gateway. It uses this gateway to access the MultiParty Call Control SCS. For information about the gateway, see "Managing OSA/Parlay Gateway Connections using Parlay_Access" in System Administrator's Guide.

For the exact version of the standards this communication service supports, see the appendix on standards and specifications in *Concepts Guide*.

Using the Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC communication service, an application can:

- Set up a uniquely identified call between one or more participants.
- Add further participants to an established call.
- Delete participants from an established call.
- Transfer participants to other established calls.
- Indicate charging information to be associated with the call session.
- Indicate information on any media to be used in association with the call.
- Interact with the functionality of other communication services, such as Audio Call to play audio to call participants or Call Notification to respond to previously established notifications.
- Query Services Gatekeeper for the status of an established call or particular call participants.
- Terminate an ongoing call it created.

How It Works

The Parlay X 3.0 Third Party Call communication service can be used by applications that need to set up calls to one or more participants, as, for example, in establishing a conference call. It can also be used to set up calls that also use the capabilities of other communication services (Audio Call or Call Notification).

The application first sets up the call session using the **makeCallSession** operation, passing in the address of at least one (the A-party) participant. In the most common case, the address of a second participant, the B-party, is also passed in.

Services Gatekeeper sends a request to establish the first call leg to the network and returns a unique identifier (**callSessionIdentifier**) to the application synchronously. This identifier allows the application to perform further administrative tasks on the call and provides any other communication services (Audio Call or Call Notification) access to the call at any point during the call session.

The call session identifier is returned to the application before the A-party goes off hook (answers). To receive information on the ongoing status of the call session, the application polls Services Gatekeeper, using the identifier and the **getCallSessionInformation** operation.

While the call is underway, the application can add additional parties, delete one or more parties, or transfer parties to and from other established call sessions, using the identifier returned during the call setup phase.

A call is terminated either by the application-facing interface or when the call participants hang up.

The Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC communication service can use the same call session identifier to access the functionality of the Parlay X 3.0 Audio Call/Parlay 3.3 UI-MPCC communication service to play media to one or more call participants. It can access the functionality of the Parlay X 3.0 Call Notification/Parlay 3.3 MPCC communication service for third party call service, such as rerouting a "busy" address to a second predefined one.

Requests using the Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC communication service flow only in one direction, from the application to the network. By itself this communication service supports only application-initiated functionality. Mobile-originated scenarios can be supported when this communication service is used in concert with Call Notification.

The Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC communication service manages only the signalling, or controlling, aspect of a call. The call itself takes place in the underlying telecom network. Only parties residing on the same network can be controlled, unless:

- The network plug-in connects to a media gateway controller.
- One of the participants is connected to a signalling gateway so that, from a signalling point of view, all parties reside on the same network.

Application Interfaces

For information about the SOAP-based interface for the Parlay X 3.0 Third Party Call communication service, see the discussion of Parlay X 3.0 Interfaces in *Application Developer's Guide*.

Events and Statistics

The Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 15–1 listsIDs of the EDRs created by the Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC communication service. This does not include EDRs created when exceptions are thrown.

Table 15–1 Event Types Generated by Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC

EDR ID	Method Called
10000	addCallParticipant
10001	deleteCallParticipant
10002	endCallSession
10003	getCallParticipantInformation
10004	getCallSessionInformation
10005	makeCallSession
10006	transferCallParticipant
10007	eventReportRes
10008	eventReportErr
10009	callLegEnded
10010	callEnded
10011	createAndRouteCallLegErr
10012	getInfoRes
10013	getInfoErr

Charging Data Records

Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC-specific CDRs are generated under the following conditions:

- After Services Gatekeeper has created the first call leg of a call session. This is not dependent on whether the participant has answered.
- After a call participant has been added to a call session, deleted from a session, or transferred to or from another session.
- When call information or call participant information has been successfully delivered to the application.
- When the call is ended by the application.

Statistics

Table 15–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 15–2 Methods and Transaction Types for Parlay X 3.0 Third Party Call /Parlay 3.3 MPCC

Method	Transaction type
makeCallSession	TRANSACTION_TYPE_CALL_CONTROL_SERVICE_INITIATED
transferCallParticipant	TRANSACTION_TYPE_CALL_CONTROL_SERVICE_INITIATED
addCallParticipant	TRANSACTION_TYPE_CALL_CONTROL_SERVICE_INITIATED

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Tunneled Parameters for Parlay X 3.0 Third Party Call / Parlay 3.3 MPCC

This section lists, by parameter key, the parameters that can be tunneled or defined in the `<requestContext>` element of an SLA.

tpc.parlay.makecallsession.first.party.anonymous

Description
Anonymous call flag.

Specifies whether the originating address is presented to the first participant in a call.

When an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation, this triggers a set of invocations of the **routeReq** operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **Presentation** data element in the **originatingAddress** parameter on the first invocation of **routeReq**.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, the **Presentation** data element is set to P_ADDRESS_PRESENTATION_RESTRICTED.

If `false`, the **Presentation** data element is set to P_ADDRESS_PRESENTATION_ALLOWED.

tpc.parlay.makecallsession.second.party.anonymous

Description
Anonymous call flag.

Specifies whether the originating address is presented to the second participant in a call.

When an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation, this triggers a set of invocations of the **routeReq** operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **Presentation** data element in the **originatingAddress** parameter on the second invocation of **routeReq**.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, the **Presentation** data element is set to P_ADDRESS_PRESENTATION_ RESTRICTED.

If `false`, the **Presentation** data element is set to P_ADDRESS_PRESENTATION_ ALLOWED.

tpc.parlay.addcallparticipant.anonymous

Description
Anonymous call flag.

Specifies whether the originating address is presented when a participant is added to an existing call.

When an application invokes the Parlay X 3.0 Third Party Call **addCallParticipant** operation, this triggers an invocation of the `routeReq` operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **Presentation** data element in the **originatingAddress** parameter on the invocation of **routeReq**.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, the **Presentation** data element is set to P_ADDRESS_PRESENTATION_ RESTRICTED.

If `false`, the **Presentation** data element is set to P_ADDRESS_PRESENTATION_ ALLOWED.

tpc.parlay.transfercallparticipant.anonymous

Description
Anonymous call flag.

Specifies whether the originating address is presented when a participant is transferred from one call session to another.

When an application invokes the Parlay X 3.0 Third Party Call **transferCallParticipant** operation, this triggers an invocation of the **routeReq** operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **Presentation** data element in the **originatingAddress** parameter on the invocation of **routeReq**.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, the **Presentation** data element is set to P_ADDRESS_PRESENTATION_ RESTRICTED.

If `false`, the **Presentation** data element is set to P_ADDRESS_PRESENTATION_ ALLOWED.

tpc.parlay.makecallsession.first.party.media.attach.explicitly

Description
Media attach flag.

Specifies whether the media is attached explicitly when a call is set up to the first participant.

When an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation, this triggers a set of invocations of the **routeReq** operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **connectionProperties** parameter on the first invocation of **routeReq**.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, the **AttachMechanism** data element is set to P_CALLLEG_ATTACH_ EXPLICITLY. The **attachMediaReq** operation in the Parlay 3.3. MPCC interface IpCallLeg is also invoked.

If `false`, the **AttachMechanism** data element is set to P_CALLLEG_ATTACH_ IMPLICITLY.

tpc.parlay.makecallsession.second.party.media.attach.explicitly

Description
Media attach flag.

Specifies whether the media is attached explicitly when a call is set up to the second participant.

When an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation, this triggers a set of invocations of the **routeReq** operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **connectionProperties** parameter on the second invocation of **routeReq**.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, the **AttachMechanism** data element is set to P_CALLLEG_ATTACH_EXPLICITLY. The **attachMediaReq** operation in the Parlay 3.3. MPCC interface IpCallLeg is also invoked.

If `false`, the **AttachMechanism** data element is set to P_CALLLEG_ATTACH_IMPLICITLY.

tpc.parlay.addcallparticipant.media.attach.explicitly

Description
Media attach flag.

Specifies whether the media is attached explicitly when a participant is added to an existing call.

When an application invokes the Parlay X 3.0 Third Party Call **addCallParticipant** operation, this triggers an invocation of the `routeReq` operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **connectionProperties** parameter on the second invocation of **routeReq**.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, the **AttachMechanism** data element is set to P_CALLLEG_ATTACH_EXPLICITLY. The **attachMediaReq** operation in the Parlay 3.3. MPCC interface IpCallLeg is also invoked.

If `false`, the **AttachMechanism** data element is set to P_CALLLEG_ATTACH_IMPLICITLY.

tpc.parlay.transfercallparticipant.media.attach.explicitly

Description
Media attach flag.

Specifies whether the media is attached explicitly when a participant is transferred from one call session to another.

When an application invokes the Parlay X 3.0 Third Party Call **transferCallParticipant** operation, this triggers an invocation of the **routeReq** operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **connectionProperties** parameter on the second invocation of **routeReq**.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
Boolean

Value
If `true`, the **AttachMechanism** data element is set to P_CALLLEG_ATTACH_ EXPLICITLY. The **attachMediaReq** operation in the Parlay 3.3. MPCC interface IpCallLeg is also invoked.

If `false`, the **AttachMechanism** data element is set to P_CALLLEG_ATTACH_ IMPLICITLY.

tpc.parlay.maximum.duration

Description
Specifies the maximum duration of a call, in milleseconds.

Specifies whether the media is attached explicitly when a participant is transferred from one call session to another.

When a call has been in progress for the amount of time set by this parameter, the **release** operation in the Parlay 3.3. MPCC IpMultiPartyCall interface is invoked.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String with Long value

tpc.parlay.makecallsession.first.party.prefix

Description
Prefix to call participant address.

Adds this string as a prefix to the address specified as the first call participant when an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

Value
Valid values are the digits **0– 9**.

tpc.parlay.makecallsession.second.party.prefix

Description
Prefix to call participant address.

Adds this string as a prefix to the address specified as the second call participant when an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

Value
Valid values are the digits **0– 9**.

tpc.parlay.addcallparticipant.prefix

Description
Prefix to call participant address.

Adds this string as a prefix to the address specified as the call participant when an application invokes the Parlay X 3.0 Third Party Call **addCallParticipant** operation.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

Value
Valid values are the digits **0– 9**.

tpc.parlay.transfercallparticipant.prefix

Description
Prefix to call participant address.

Adds this string as a prefix to the address specified as the call participant when an application invokes the Parlay X 3.0 Third Party Call **transferCallParticipant** operation.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

Value
Valid values are the digits **0– 9**.

tpc.parlay.makecallsession.first.party.callappgenericinfo

Description
Call forwarding indicator.

Specifies the maximum number of allowed call forwarding hops for the address specified as the first call participant when an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation.

When an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation, this triggers an invocation of the `routeReq` operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **P_CALL_APP_GENERIC_INFO** element in the **appInfo** parameter of the **routeReq** operation.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

Value
The string must be formatted as follows:

redirectionCounter = *n*

where *n* is the number of allowed call hops.

Value range for *n* is **1–5**.

Example
This example allows two hops:

```
redirectionCounter=2
```

tpc.parlay.makecallsession.second.party.callappgenericinfo

Description
Call forwarding indicator.

Specifies the maximum number of allowed call forwarding hops for the address specified as the second call participant when an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation.

When an application invokes the Parlay X 3.0 Third Party Call **makeCallSession** operation, this triggers an invocation of the `routeReq` operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **P_CALL_APP_GENERIC_INFO** element in the **appInfo** parameter of the **routeReq** operation.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

Value
The string must be formatted as follows:

redirectionCounter = *n*

where *n* is the number of allowed call hops.

Value range for *n* is **1–5**.

Example
This example allows two hops:

```
redirectionCounter=2
```

tpc.parlay.addcallparticipant.callappgenericinfo

Description
Call forwarding indicator.

Specifies the maximum number of allowed call forwarding hops for the call participant when an application invokes the Parlay X 3.0 Third Party Call **addCallParticipant** operation.

When an application invokes the Parlay X 3.0 Third Party Call **addCallParticipant** operation, this triggers an invocation of the **routeReq** operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **P_CALL_APP_GENERIC_INFO** element in the **appInfo** parameter of the **routeReq** operation.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

Value
The string must be formatted as follows:

redirectionCounter = *n*

where *n* is the number of allowed call hops.

Value range for *n* is **1–5**.

Example
This example allows two hops:

```
redirectionCounter=2
```

tpc.parlay.transfercallparticipant.callappgenericinfo

Description
Call forwarding indicator.

Specifies the maximum number of allowed call forwarding hops for the call participant when an application invokes the Parlay X 3.0 Third Party Call **transferCallParticipant** operation.

When an application invokes the Parlay X 3.0 Third Party Call **transferCallParticipant** operation, this triggers an invocation of the **routeReq** operation in the Parlay 3.3. MPCC IpCallLeg interface.

This setting affects the value of the **P_CALL_APP_GENERIC_INFO** element in the **appInfo** parameter of the `routeReq` operation.

Can be set using SLAs or parameter tunneling. An SLA setting overrides a tunneled parameter.

Format
String

Value
The string must be formatted as follows:

redirectionCounter = *n*

where *n* is the number of allowed call hops.

Value range for *n* is **1–5**.

Example
This example allows two hops:

```
redirectionCounter=2
```

Managing Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC

This section describes the properties and workflow for the Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC plug-in instance.

This plug-in service is requires Orbacus, which is not installed by default. For information about installing Orbacus, see *Installation Guide*.

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one-to-one mapping between plug-in service and plug-in instance. The plug-in instance is created when the plug-in service is started.

Properties for Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC

Table 15–3 lists the technical specifications for the communication service.

Table 15–3 Properties for Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > Plugin_px30_third_party_call_parlay_mpcc

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=Plugin_px30_third_party_call_parlay_mpcc
	Type=com.bea.wlcp.wlng.plugin.tpc.parlay.management.ThirdPartyCallMBean

Network protocol plug-in service ID	Plugin_px30_third_party_call_parlay_mpcc
Network protocol plug-in instance ID	Plugin_px30_third_party_call_parlay_mpcc
Supported Address Scheme	tel
Application-facing interface	com.bea.wlcp.wlng.px30.plugin.ThirdPartyCallPlugin
Service type	ThirdPartyCall
Exposes to the service communication layer a Java representation of:	Parlay X 3.0 Part 2: Third Party Call
Interfaces with the network nodes using:	Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control SCF; Subpart 7: MultiParty Call Control Service
Deployment artifacts	Plugin_px30_third_party_call_parlay_mpcc.jar, packaged in wlng_nt_third_party_call_px30.ear px30_third_party_call.war, packaged in wlng_at_third_party_call_px30.ear

This plug-in service does not support multiple instantiation using the Plug-in Manager. There is a one-to-one mapping between the plug-in service and the plug-in instance. The plug-in instance is created when the plug-in service is started.

Configuration Workflow for Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Select the MBean detailed in Properties for Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC.
2. Configure the attributes of the plug-in instance:
   - Attribute: CallingParticipantNameMandantory
   - Attribute: MaximumDurationEnforced
   - Attribute: MultiMediaSupported
   - Attribute: ChargingAllowed
   - Attribute: StatusRetentionTime
   - Attribute: ChangeMediaAllowed
   - Attribute: MaximumParticipants
   - Operation: configCallGetInfoReq
   - Operation: configLegGetInfoReq
3. Gather information about the OSA Gateway and configure the plug-in instance accordingly. The following information needs to be obtained from the OSA Gateway administrator and configured in the OSA Access service:
   - OSA/Parlay SCS type to be used in the lookup (service discovery) phase when requesting the service (OSA/Parlay SCS) from the OSA/Parlay Gateway. Typically this is P_MULTI_PARTY_CALL_CONTROL.
   - OSA/Parlay service properties to be used in the lookup (service discovery) phase when requesting a service (OSA/Parlay SCS) from the OSA/Parlay Gateway. This depends on the OSA Gateway implementation.
   - Authentication type used by the OSA/Parlay Framework.
   - Encryption method used for the connection with the OSA Gateway.
   - Signing algorithm used when signing the Service Level Agreement (SLA) with the OSA/Parlay Framework.
4. Set up the OSA Client and the OSA Client Mappings. See "Creating an OSA Client" and "Mapping the OSA client to an OSA Gateway and an OSA/Parlay SCS" in "Managing OSA/Parlay Gateway Connections using Parlay_Access" in *System Administrator's Guide*.
5. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes detailed in the "Properties for Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC" section.
6. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
7. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Management Operations for Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC

The following operations are related to management:

- Operation: getCallLegs
- Operation: getCallSessionInfo

- Operation: getCallLegSessionInfo
- Operation: listCallSessionIds
- Operation: countPendingCallSession

Reference: Attributes and Operations for Parlay X 3.0 Third Party Call/Parlay 3.3 MPCC

This section describes the attributes and operations for configuration and maintenance:

- Attribute: CallGetInfoReqConfig (read-only)
- Attribute: CallingParticipantNameMandantory
- Attribute: ChangeMediaAllowed
- Attribute: ChargingAllowed
- Attribute: LegGetInfoReqConfig (read-only)
- Attribute: MaximumDurationEnforced
- Attribute: MaximumParticipants
- Attribute: MultiMediaSupported
- Attribute: StatusRetentionTime
- Operation: configCallGetInfoReq
- Operation: configLegGetInfoReq
- Operation: getCallLegs
- Operation: getCallSessionInfo
- Operation: getCallLegSessionInfo
- Operation: listCallSessionIds
- Operation: countPendingCallSession

Attribute: CallGetInfoReqConfig (read-only)

Scope: Cluster

Unit: Not applicable

Format: String

Indicates the current configuration for the **getInfoReq** operation in the IpMultiPartyCall interface. The information includes:

- **Supported**: Boolean that indicates if the operation is supported.
- **PCallInfoTimes**: Boolean that indicates if the P_CALL_INFO_TIMES tag is present in the operation.
- **PCallInfoReleaseCause**: Boolean that indicates if the P_CALL_INFO_RELEASE_CAUSE tag is present in the operation.

Use "Operation: configCallGetInfoReq" to change these settings.

Attribute: CallingParticipantNameMandantory

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if the **callingParticipantName** parameter in the **makeCallSession** operation should be used as the original address.

If **callingParticipantName** is required, it must be in the form of a string that can be translated to a URI; for example: tel:123456.

Enter:

- `true` if **callingParticipantName** should be used as the originating address.
- `false` otherwise.

Attribute: ChangeMediaAllowed

Scope: Server

Unit: Not applicable

Format: Boolean

Specifies if an end user (a call participant) is allowed to change the media used in the call.

- `true` if an end user is allowed to change media for an existing call session.

  For an end user to change the media type for a given call session the following conditions are required:

  – This attribute must be `true`.

  – The application must have allowed the end user to change media when the call session was established by the **makeCallSession** operation.

  – "Attribute: MultiMediaSupported" must also be `true`.
- `false` if an end user is not allowed to change media for an existing call session.

Attribute: ChargingAllowed

Scope: Server

Unit: Not applicable

Format: Boolean

Specifies whether charging is allowed.

- `true` if an application is allowed to specify charging information when creating a call session by the **makeCallSession** operation.
- `false` otherwise.

Attribute: LegGetInfoReqConfig (read-only)

Scope: Cluster

Unit: Not applicable

Format: String

Indicates the current configuration for the **getInfoReq** operation in the IpCallLeg interface. The information includes:

- **Supported:** Boolean that indicates if the operation is supported.
- **PCallLegInfoTimes:** Boolean that indicates if the P_CALL_LEG_INFO_TIMES tag is present in the operation.
- **PCallLegInfoReleaseCause**: Boolean that indicates if the P_CALL_LEG_INFO_RELEASE_CAUSE tag is present in the operation.
- **PCallLegInfoAddress**: Boolean that indicates if the P_CALL_LEG_INFO_ADDRESS tag is present in the operation.
- **PCallLegInfoAppInfo**: Boolean that indicates if the P_CALL_LEG_INFO_APPINFO tag is present in the operation.

Use "Operation: configLegGetInfoReq" to change these settings.

Attribute: MaximumDurationEnforced

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if a call whose duration exceeds the maximum value will be terminated.

Enter:

- `true` to terminate the call.
- `false` to allow the call to continue.

Attribute: MultiMediaSupported

Scope: Cluster

Unit: Not applicable

Format: Boolean

Indicates if multimedia is supported.

- `true` if multimedia is supported.
- `false` if multimedia is not supported.

Attribute: MaximumParticipants

Scope: Server

Unit: Not applicable

Format: Integer

Specifies the maximum number of participants in a call.

Valid values are **2–65535**.

Attribute: StatusRetentionTime

Scope: Server

Unit: Seconds

Format: Integer

Specifies the length of time information about a call is stored after the call is terminated.

Valid values are **0–65535**.

Operation: configCallGetInfoReq

Scope: Cluster

Configures the parameters in the **getInfoReq** operation in the IpMultiPartyCall interface.

Signature:

```
configCallGetInfoReq(Supported: Boolean, PCallInfoTimes: Boolean,
PCallInfoReleaseCause: Boolean)
```

Table 15–4 configCallGetInfoReq Parameters

Parameter	Description
Supported	Specifies if the operation is supported by the OSA/Parlay Gateway.
PCallInfoTimes	Specifies if the P_CALL_INFO_TIMES tag should be present in TpCallInfoType. Use: ■ `true` to add it to TpCallInfoType. ■ `false` otherwise.
PCallInfoReleaseCause	Specifies if the P_CALL_INFO_RELEASE_CAUSE tag should be present in TpCallInfoType. Use: ■ `true` to add it to TpCallInfoType. ■ `false` otherwise.

Operation: configLegGetInfoReq

Scope: Cluster

Configures the parameters in the **getInfoReq** operation in the IpCallLeg interface.

Signature:

```
configLegGetInfoReq(Supported : Boolean, PCallLegInfoTimes : Boolean,
PCallLegInfoReleaseCause : Boolean, PCallLegInfoAddress : Boolean,
PCallLegInfoAppInfo : Boolean)
```

Table 15–5 configLegGetInfoReq Parameters

Parameter	Description
Supported	Specifies if the operation is supported by the OSA/Parlay Gateway.

Table 15–5 (Cont.) configLegGetInfoReq Parameters

Parameter	Description
PCallLegInfoTimes	Specifies if the P_CALL_LEG_INFO_TIMES tag should be present in TpCallLegInfoType. Use: ■ `true` to add it to TpCallLegInfoType. ■ `false` otherwise.
PCallLegInfoReleaseCaus e	Specifies if the P_CALL_LEG_INFO_RELEASE_CAUSE tag should be present in TpCallLegInfoType. Use: ■ `true` to add it to TpCallLegInfoType. ■ `false` otherwise.
PCallLegInfoAddress	Specifies if the P_CALL_LEG_INFO_ADDRESS tag should be present in TpCallLegInfoType. Use: ■ `true` to add it to TpCallLegInfoTyp.e ■ `false` otherwise.
PCallLegInfoAppInfo	Specifies if the P_CALL_LEG_INFO_APPINFO tag should be present in TpCallLegInfoType. Use: ■ `true` to add it to TpCallLegInfoType. ■ `false` otherwise.

Operation: getCallLegs

Scope: Cluster

Displays a list of IDs for all call legs in a call session.

Signature:

```
getCallLegs(CallSessionId: String)
```

Table 15–6 getCallLegs Parameters

Parameter	Description
CallSessionId	ID of the call session to list call legs for

Operation: getCallSessionInfo

Scope: Cluster

Displays information about a call session. This includes:

- **callSessionId**: The ID of the call session.
- **callStatus**: The current status of the call. The status is one of:
  - **Established**, during call duration.
  - **Terminated**, when the call has terminated but information is still present in the internal storage: see "Attribute: StatusRetentionTime" for more information.
  - **Expired**, when the call is terminated and information is no longer available.

- **originalAddress**: The originator (a-party) of the call.
- **appInstGrpId**: The application instance ID associated with the application that created the call session.
- **callRef**: The CORBA reference to the call object in the Parlay Gateway (IpMultiPartyCall).
- **srcPlugin**: the type of plug-in instance that initiated the call. The type is one of:
  - ThirdPartyCall
  - CallNotification
  - CallDirection

Signature:

```
getCallSessionInfo(CallSessionId: String)
```

Table 15–7 getCallSessionInfo Parameters

Parameter	Description
CallSessionId	ID of the call session to get information about.

Operation: getCallLegSessionInfo

Scope: Cluster

Displays information about a call leg in a call session.

- **id**: The CORBA reference to the call leg object in the Parlay Gateway (IpCallLeg).
- **callSessionId**: The ID of the call session.
- **callParticipantIdentifier**: The URI identifying the terminal associated with the call leg.
- **callParticipantStatus**: The status of the call participant. The status is one of:
  – **Initial**, during call setup.
  – **Connected**, during call duration.
  – **Terminated**, the participant has left the call.
- **callParticipantStartTime**: The time the call participant was connected to the call.
- **callParticipantEndTime**: The time the call participant was disconnected from the call.
- **callParticipantDuration**: The duration of the call for the participant. Given in seconds.
- **callParticipantTerminationCause**: The cause of the participant leaving the call. One of:
  – **Noanswer**, no answer from the participant.
  – **Busy**, the participant was busy (off-hook)
  – **Hangup**, the participant went on-hook.
  – **Notreachable**, could not reach the participant.
  – **Aborted**, the call was terminated for a reason other than Hangup.

- **appInstGrpId**: the application instance associated with the application that created the call session.
- **callRef**: The CORBA reference to the call object in the Parlay Gateway (IpMultiPartyCall).
- **srcPlugin**: The type of plug-in instance that initiated the call. The type is one of:
  - ThirdPartyCall
  - CallNotification
  - CallDirection

Signature:

```
getCallLegSessionInfo(CallLegSessionId: String)
```

Table 15–8 getCallLegSessionInfo Parameters

Parameter	Description
CallLegSessionId	ID of the call leg session to get information about.

Operation: listCallSessionIds

Scope: Cluster

Displays a list of IDs for ongoing call sessions. These are the Parlay X 3.0 call session IDs.

Signature:

```
listCallSessionIds()
```

Operation: countPendingCallSession

Scope: Server

Displays the number of ongoing call sessions for this plug-in instance.

Signature:

```
countPendingCallSession()
```

16 Extended Web Services Binary SMS/SMPP

This chapter describes the Extended Web Services (EWS) Binary SMS/Short Message Peer to Peer (SMPP) Communication Service in detail.

Overview of the EWS Binary SMS/SMPP

The EWS Binary SMS/SMPP communication service allows applications to send and receive generic binary object attachment, such as vCards. It exposes the Oracle Extended Web Services Binary SMS interface.

The communication service acts as an External Short Message Entity (ESME) that connects to a Short Messaging Service Center (SMSC) over TCP/IP.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using the EWS Binary SMS/SMPP communication service an application can:

- Send short messages with binary attachments to one or more destination addresses.
- Subscribe and unsubscribe for network-triggered binary short messages with binary attachments.
- Receive network-triggered short messages with binary attachments.

The actual message element is made up of an array of UDH and message parts, encoded in Base64. See "3rd Generation Partnership Project; Technical Specification Group Terminals; Technical realization of the short message service (SMS); (Release 6) 3GPP 23.040 Version 6.5.0" at:

`http://www.3gpp.org/ftp/Specs/html-info/23040.htm`

The send message operation gives an application the flexibility to manipulate the SMPP UDH and message data. Both the UDH and message data elements are optional, but the overall element, **binaryMessage**, is required. The contents of the UDH and the message can be of any binary data, although any byte array should be less than 140 bytes due to SMPP limitations, and the number of BinaryMessage arrays should be less than the **SegmentsLimit** specified in OAM. The default value is 1024. see Attribute: SegmentsLimit.

The notification operation gives the application access to an array of SMPP UDHs, the SMPP DCS, the protocol identifier according to 3GPP 23.040 Version 6.5.0, and other data such as sender address, destination address and timestamp of the message.

SMPP expects the sender name value to be in ASCII characters. The use of non-ASCII characters may cause the request to become garbled or even be removed at the SMSCS

Services Gatekeeper provides support for the billing identification identifier, **smpp_ billing_id**, defined in SMPP Specification 5.0, through the use of a tunneled parameter. It also supports the **ussd_service_operation**, which was added as an optional parameter to the **deliverSM** operation as a tunneled parameter in SMPP v 5.0. See the descriptions of the **smpp_billing_id** and **ussd_service_operation** tunneled parameters in Chapter 6, "Parlay X 2.1 Short Messaging/SMPP" for more information.

Send Receipts

Send receipts are acknowledgements that the network node has received the short message from the application by Services Gatekeeper. Although a single short message may be sent to multiple destination addresses, normally only one send receipt is returned to the application by Services Gatekeeper. The receipt is returned synchronously in the response message to the **sendBinarySms** operation.

Delivery Receipts

Delivery receipt notifications can be set up using the **sendBinarySms** operation, but the actual asynchronous delivery of receipts is accomplished using the Parlay X 2.1 Short Messaging interface. See "Delivery Receipts" in Chapter 6, "Parlay X 2.1 Short Messaging/SMPP" for information on delivery receipts.

Connection Handling and Provisioning

The EWS Binary SMS/SMPP communication service uses the Services Gatekeeper SMPP Server Service to establish and manage southbound connections between Services Gatekeeper and Short Message Service Centers (SMSCs). The SMPP Server Service is deployed as an Oracle WebLogic Server Service.

The SMPP Server Service provides these services for the Parlay X 2.1 Short Messaging and Native SMPP plug-ins as well as for EWS Binary SMS/SMPP.

For information about configuration options pertaining to these client connections, see the System Properties for SMPP Server Service and Reference: Attributes and Operations for SMPP Server Service sections.

The client connection ID is created on the plug-in's successful bind with the SMSC. The connection ID changes on a successful rebind.

For information about connection handling and provisioning, multiple connections and multiple plug-in instances, windowing, and load balancing/high availability, see the applicable sections in Chapter 6, "Parlay X 2.1 Short Messaging/SMPP."

Application Interfaces

For information about the application interface for the EWS Binary SMS/SMPP communication service, see the discussion of Extended Web Services Binary SMS in *Application Developer's Guide*.

For information about the RESTful Call Notification interface, see the discussion of Binary Short Messaging in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the application interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on, they are the same.

Events and Statistics

For general information, see Appendix A, "Events, Alarms, and Charging."

The Extended Web Services Binary SMS/SMPP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service.

Event Data

Table 16–1 lists the IDs of the EDRs created by the EWS Binary SMS/SMPP communication service. This list does not include EDRs created when exceptions are thrown.

Table 16–1 Event Types Generated by EWS Binary SMS /SMPP

EDRID	Method Called
7101	sendBinarySms
7201	startBinarySmsNotification
7202	stopBinarySmsNotification
7204	notifyBinarySmsDeliveryReceipt
7205	notifyBinarySmsReception

For the list of EDRs generated by the SMPP Server Service, see Table 20–2, " Event Types Generated by the SMPP Server Service".

Charging Data Records

EWS Binary SMS/SMPP-specific CDRs are generated under the following conditions:

- After a mobile-terminated **sendBinarySms** request is sent from Services Gatekeeper to the network.
- After a a network-triggered binary SMS message has been successfully delivered to the application.

Statistics

Table 16–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 16–2 Methods and Transaction Types for EWS Binary SMS/SMPP

Method	Transaction type
sendBinarySMS	TRANSACTION_TYPE_MESSAGING_SEND
receivedMobileOriginatedBinarySMS	TRANSACTION_TYPE_MESSAGING_RECEIVE

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing EWS Binary SMS/SMPP

The properties, workflow, tunneled parameters and management operations for the EWS Binary SMS/SMPP communication service are identical to those provided for the Parlay X 2.1 Short Messaging/SMPP communication service.

For details, see:

- Managing Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP
- Properties for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP
- Configuration Workflow for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP
- Management Operations in the SMPP Server Service
- Reference: Attributes and Operations for Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP
- Tunneled Parameters for Parlay X 2.1 Short Messaging / SMPP

17 Extended Web Services Subscriber Profile/LDAPv3

This chapter describes the Extended Web Services (EWS) Subscriber Profile/Lightweight Directory Access Protocol (LDAPv3) communication service in detail.

Overview of the EWS Subscriber Profile/LDAPv3 Communication Service

The EWS Subscriber Profile/LDAPv3 communication service exposes Oracle's Extended Web Services Subscriber Profile application interface.

The communication service acts as an LDAP client to a directory service, connecting to the directory service using LDAPv3.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using the EWS Subscriber Profile/LDAPv3 communication service, an application can:

- Retrieve the specific value for a particular property belonging to a subscriber profile stored in an LDAP data source.
- Retrieve an entire subscriber profile from an LDAP data source, subject to SLA filtering.

Application Interfaces

For information about the application interface for the Extended Web Services Subscriber Profile communication service, see the discussion of Extended Web Services Subscriber Profile in *Application Developer's Guide*.

For information about the RESTful Call Notification interface, see the discussion of Subscriber Profile in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on, they are the same.

Events and Statistics

The EWS Subscriber Profile/LDAPv3 communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 17–1 lists IDs of the EDRS created by the EWS Subscriber Profile/LDAPv3 communication service. This list does not include EDRs created when exceptions are thrown

Table 17–1 Event Types Generated by EWS Subscriber Profile/LDAPv3

EDR ID	Method Called
13001	get
13002	getProfile

Charging Data Records

EWS Subscriber Profile/LDAPv3-specific CDRs are generated under the following conditions:

- After Services Gatekeeper has returned a full or partial subscriber profile to an application based on one or more attributes requested by that application.
- After Services Gatekeeper has returned a subscriber profile to an application based on the ID of the profile.

Statistics

Table 17–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 17–2 Methods and Transaction Types for EWS Subscriber Profile/LDAPv3

Method	Transaction Type
get	TRANSACTION_TYPE_SUBSCRIBER_PROFILE
getProfile	TRANSACTION_TYPE_SUBSCRIBER_PROFILE

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing EWS Subscriber Profile/LDAPv3

This section describes the properties and workflow for the EWS Subscriber Profile/LDAPv3 plug-in instance.

It includes an LDAP server schema to use in constructing LDAP queries.

A connection pool is used for connections to the LDAP server. The connection pool is shared among all plug-in instances, and any configuration settings related to this pool or schema updates are broadcast to all plug-in instances in the cluster.

Use "Operation: updateLDAPSettings" to force configuration changes to take effect.

Properties for EWS Subscriber Profile/LDAPv3

Table 17–3 lists the technical specifications for the communication service.

Table 17–3 Properties for EWS Subscriber Profile/LDAPv3

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plugin_instance_id*

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=same as the network protocol *instance_id* assigned when the plug-in instance is created.
	Type=com.bea.wlcp.wlng.plugin.subscriberprofile.ldap.managedplugin.management.SubscriberProfileMBean

Network protocol plug-in service ID	Plugin_ews_subscriber_profile_ldap
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
Supported Address Scheme	tel, id, imsi, ipv4
Application-facing interface	com.bea.wlcp.wlng.ews.plugin.SubscriberProfilePlugin
Service type	SubscriberProfile
Exposes to the service communication layer a Java representation of:	Extended Web Services Subscriber Profile
Interfaces with the network nodes using:	LDAP
Deployment artifact NT EAR wlng_nt_subscriber_profile_ews.ear	ews_subscriber_profile_service.jar and Plugin_ews_subscriber_profile_ldap.jar
Deployment artifact AT EAR: Normal wlng_at_subscriber_profile_ews.ear	ews_subscriber_profile.war and rest_subscriber_profile.war
Deployment artifact AT EAR: SOAP Only wlng_at_subscriber_profile_ews_soap.ear	ews_subscriber_profile.war

LDAP Server Schema

All subscriber-profile-related operations are handed off to network nodes that accept LDAP queries according to LDAPv3. The decision concerning which node in the LDAP directory should be used to perform the query is decided at run time based on configuration settings. The data that is handed back to the application that initiated the Subscriber Profile query is filtered using the result filter mechanism in the service provider group and application group SLAs. For more information, see

`<resultRestrictions>` in "Defining Service Provider Group and Application Group SLAs" in the *Accounts and SLAs Guide*.

A schema is used for constructing queries. See Example 17–1.

Example 17–1 LDAP Query schema XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="LdapConfig">
<xs:complexType>
<xs:sequence>
<xs:element name="Keys" type="KeySet" minOccurs="1" maxOccurs="unbounded"/>
<xs:element name="LdapObject" type="LdapObject" minOccurs="1"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType name="KeyObject">
<xs:sequence>
<xs:element name="uriScheme" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="addressKeyName" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="objectKeyName" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xs:element name="objectKeyValue" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="optional"/>
</xs:complexType>

<xs:complexType name="KeySet">
<xs:sequence>
<xs:element name="Key" type="KeyObject" minOccurs="1" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="LdapObject">
<xs:sequence>
<xs:element name="ObjectKeySet" type="xs:string" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="keyName" type="xs:string" use="required"/>
<xs:attribute name="keyValue" type="xs:string" use="required"/>
</xs:complexType>
</xs:schema>
```

The LDAP server schema describes the following elements:

- **LdapObject**: Holder of a KeySet
- **KeySet**: Defines a collection of KeyObjects. Sets of keys are used because there may be several ways to reach a certain node in the tree. One LDAP plug-in instance can be configured with several KeySets and can provide the link between the search key in the Extended Web Services interface and the LDAP tree.
- **KeyObject**: Defines an entry point to the LDAP tree and provides the link between the search key in the Extended Web Services interface and the LDAP tree.

Table 17–4 describes the schema objects in detail.

Table 17–4 LDAP Server Schema

Object	Element	Description
LdapObject	ObjectKeySet	Defines the KeySet through which it can be reached. Refers to theID attribute of a defined KeySet.
LdapObject	id	The identity of the LdapObject. Can be referenced from other LdapObjects through the ParentObjectId field.
LdapObject	keyName	The name of the key through which the LdapObject can be reached.
LdapObject	keyValue	The value of the key through which the LdapObject can be reached.
KeyObject	uriScheme	Defines the URI scheme of the address for which this key applies.
KeyObject	addressKeyName	Defines the key name with which the address value is associated.
KeyObject	objectKeyName	Provides the possibility of defining the addressing key of a possible tree node above the node that is reached by the address key (that is, like the domain object in the 3DS directory information tree).
KeyObject	objectKeyValue	See objectKeyName. Defines the value of the key.
KeyObject	id	The identity of the key. Used only for descriptive purposes.
KeySet	Key	All keys in the KeySet
KeySet	id	The identity of the KeySet. Used when associating an LdapObject with a KeySet.

Example 17–2 shows a directory information tree built using the schema described in Table 17–4.

Example 17–2 Example of LDAP server schema

```
<?xml version="1.0" encoding="UTF-8"?>
<LdapConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation='sp_config.xsd'>
<Keys id="myKeys">
<Key id="misidnKey">
<uriScheme>tel</uriScheme>
<addressKeyName>msisdn</addressKeyName>
<objectKeyName>domainName</objectKeyName>
<objectKeyValue>msisdnD</objectKeyValue>
</Key>
<Key id="imsiKey">
<uriScheme>imsi</uriScheme>
<addressKeyName>imsi</addressKeyName>
<objectKeyName>domainName</objectKeyName>
<objectKeyValue>imsiD</objectKeyValue>
</Key>
<Key id="subscriberIdKey">
<uriScheme>id</uriScheme>
<addressKeyName>id</addressKeyName>
<objectKeyName>domainName</objectKeyName>
```

```
<objectKeyValue>subsD</objectKeyValue>
</Key>
<Key id="ipv4Key">
<uriScheme>ipv4</uriScheme>
<addressKeyName>ipv4Addr</addressKeyName>
<objectKeyName>domainName</objectKeyName>
<objectKeyValue>ipv4D</objectKeyValue>
</Key>
</Keys>
<LdapObject id="mySchema" keyName="serviceName" keyValue="mySchema">
<ObjectKeySet>myKeys</ObjectKeySet>
</LdapObject>
</LdapConfig>
```

Configuration Workflow for EWS Subscriber Profile/LDAPv3

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID as listed in the "Properties for EWS Subscriber Profile/LDAPv3" section.
2. Select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID given when the plug-in instance was created.
3. Define the characteristics of the LDAP server to connect to:
   - Attribute: Port
   - Attribute: AuthDN
   - Attribute: BaseDN
   - Attribute: AuthPassword
4. Using either "Attribute: Schema" or "Operation: updateSchemaURL", define the schema.

   See "LDAP Server Schema" for a description of the schema and "Configuration Workflow for EWS Subscriber Profile/LDAPv3" for a description of the mappings.
5. Define the connection pool characteristics for the connection:
   - Attribute: MinConnections
   - Attribute: MaxConnections
   - Attribute: ConnTimeout
   - Attribute: RecoverTimerInterval
6. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for EWS Subscriber Profile/LDAPv3" section.
7. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
8. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Management Operations for EWS Subscriber Profile/LDAPv3

There are no specific management operations, except for "Operation: updateLDAPSettings", used to update the LDAP connection pool after changing any of the following attributes:

- Attribute: MinConnections
- Attribute: MaxConnections
- Attribute: ConnTimeout
- Attribute: RecoverTimerInterval

Provisioning for EWS Subscriber Profile/LDAPv3

If the results from the LDAP query should be filtered, use the service provider group and application group SLAs. See `<resultRestriction>` in "Defining Service Provider Group and Application Group SLAs" in the *Accounts and SLAs Guide*.

Reference: Attributes and Operations for EWS Subscriber Profile/LDAPv3

This section describes the attributes and operations for configuration and maintenance:

- Attribute: AuthDN
- Attribute: AuthPassword
- Attribute: BaseDN
- Attribute: ConnTimeout
- Attribute: Host
- Attribute: LDAPConnectionStatus
- Attribute: MaxConnections
- Attribute: MinConnections
- Attribute: RecoverTimerInterval
- Attribute: Port
- Attribute: Schema
- Operation: updateLDAPSettings
- Operation: updateSchemaURL

Attribute: AuthDN

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the authentication Distinguished Name (DN) for the LDAP server.

Example:

```
cn=admin,o=acompany,c=uk
```

Attribute: AuthPassword

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the password associated with "Attribute: AuthDN".

Attribute: BaseDN

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the base Distinguished Name (DN) for the LDAP database in use.

Example:

```
o=acompany,c=uk
```

Attribute: ConnTimeout

Scope: Cluster

Unit: Seconds

Specifies the maximum time to wait for an LDAP connection to be established. If the related timer expires, a retry is performed. See "Attribute: RecoverTimerInterval" for more information.

Any change to this setting must be followed by "Operation: updateLDAPSettings".

Attribute: Host

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the host name or IP address of the LDAP server to connect to.

Examples:

```
myldapserver.mycompany.org
192.168.0.14
```

Attribute: LDAPConnectionStatus

Read-only.

Scope: Cluster

Unit: Not applicable

Format: String enumeration listed in Table 17–5.

Table 17–5 Status of the connection to the LDAP server

Status	Description
active	The connection is active. The plug-in instance accepts requests.

Table 17–5 (Cont.) Status of the connection to the LDAP server

Status	Description
update_pending	The connection is temporarily unavailable due to an update of the configuration settings. The plug-in instance does not accept requests.
deactive	The connection is inactive. The plug-in instance does not accept requests. Reasons for this entering this state include: ▪ Missing or incorrect configuration ▪ LDAP server is unreachable ▪ Internal errors

Attribute: MaxConnections

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the maximum number of connections in the LDAP connection pool.

Any change to this setting must be followed by "Operation: updateLDAPSettings".

Attribute: MinConnections

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the minimum number of connections to establish using connections from the LDAP connection pool.

Any change to this setting must be followed by "Operation: updateLDAPSettings".

Attribute: Port

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the port number of the LDAP server to connect to.

Attribute: RecoverTimerInterval

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the time to wait before performing an LDAP connection retry after an LDAP connection error. Should be at least twice the time defined in "Attribute: ConnTimeout".

Any change to this setting must be followed by "Operation: updateLDAPSettings".

Attribute: Schema

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the LDAP schema to use.

Operation: updateLDAPSettings

Scope: Cluster

Refreshes the LDAP connection pool to use the new configuration.

During the update, the LDAP connection is temporarily unavailable and the connection status is **update_pending**. See Table 17–5, " Status of the connection to the LDAP server" for status values.

Signature:

```
updateLDAPSettings()
```

Operation: updateSchemaURL

Scope: Cluster

Updates the schema to use when performing lookups in the LDAP database.

During the update, the LDAP connection is temporarily unavailable and the connection status is **update_pending**. See Table 17–5, " Status of the connection to the LDAP server" for status values.

Signature:

```
updateSchemaURL(SchemaURL:String)
```

Table 17–6 updateSchemaURL Parameters

Parameter	Description
SchemaURL	URL to the LDAP database schema. Examples: `file:///d:/ldap/schema.xml` (Windows systems) `file://ldap/schema.xml` (UNIX systems)

18 Extended Web Services WAP Push/PAP

This chapter describes the Extended Web Services (EWS) WAP Push/Push Access Protocol (PAP) communication service in detail.

Overview of the EWS WAP Push/PAP Communication Service

The EWS WAP Push/PAP communication service exposes the Oracle Extended Web Services WAP Push interface.

The communication service connects to a Push Proxy Gateway (PPG) using Push Access Protocol (PAP) 2.0. See "Push Access Protocol (PAP) 2.0" for information about this network protocol.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using the EWS WAP Push/PAP communication service, an application can:

- Send a WAP Push message.
- Send a replacement WAP Push message.
- Ask to be notified asynchronously of the status of WAP Push messages that have been sent. The possible values returned include:
  - Rejected: The message was not accepted.
  - Pending: The message is in process.
  - Delivered: The message was successfully delivered to the end-user.
  - Undeliverable: The message could not be delivered because of a problem.
  - Expired: The message reached the maximum age allowed by server policy or could not be delivered by the time specified in the push submission.
  - Aborted: The mobile device aborted the message.
  - Timeout: The delivery process timed out.
  - Cancelled: The message was cancelled through the cancel operation.
  - Unknown: The server does not know the state of the message.
- Send a result notification message. This occurs only if the initial push submission was accepted for processing. One result notification message is sent per destination address.

Push Access Protocol (PAP) 2.0

EWS WAP Push/PAP supports a subset of the PAP 2.0 operations. These include:

- push-message: Submits a message to be delivered. This operation is also used to send a replacement message.
- push-response: The response to the push-message operation. This response includes a code specifying the immediate status of the message submission, of the following general types:
  - 1xxx Success: The action was successfully received, understood, and accepted
  - 2xxx Client Error: The request contains bad syntax or cannot be fulfilled
  - 3xxx Server Error: The server failed to fulfil an apparently valid request
  - 4xxx: Service Failure: The service could not be performed. The operation may be retried
- resultnotification-message: Specifies the final outcome of a specific message for a specific recipient. Sent only if the initial request includes the URL to which this notification is to be delivered. Includes both textual indication of state and a status code including the following general types:
  - 1xxx Success: The action was successfully received, understood, and accepted
  - 2xxx Client Error: The request contains bad syntax or cannot be fulfilled
  - 3xxx Server Error: The telecom network node failed to fulfil an apparently valid request
  - 4xxx: Service Failure: The service could not be performed. The operation may be retried
  - 5xxx: Mobile Device Abort: The mobile device aborted the operation.
- resultnotification-response: The response to the result notification. This response includes a code specifying the status of the notification
  - 1xxx Success: The action was successfully received, understood, and accepted
  - 2xxx Client Error: The request contains bad syntax or cannot be fulfilled
- badmessage-response: A response indicating that request is unrecognizable or is of a protocol version that is not supported. This response contains either a 3002 code (Version not supported) or a 2000 code (Bad Request). In the case of Bad Request, a fragment of the unrecognizable message is included in the response

See the appendix on standards and specifications in *Concepts Guide* for the exact version of the protocol standard Services Gatekeeper supports.

Application Interfaces

For information about the application interface for the Extended Web Services WAP Push communication service, see the discussion of Extended Web Services WAP Push in *Application Developer's Guide*.

For information about the RESTful Call Notification interface, see the discussion of WAP Push in *RESTful Application Developer's Guide*.

The RESTful Service Call Notification interfaces provide RESTful access to the same functionality as the SOAP-based interfaces. The internal representations are identical, and for the purposes of creating SLAs and reading CDRs, and so on, they are the same.

Events and Statistics

The EWS WAP Push/PAP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service

For general information, see Appendix A, "Events, Alarms, and Charging."

Charging Data Records

EWS WAP Push/PAP-specific CDRs are generated under the following conditions:

- When the **sendPushMessage** response returns from the network.
- When a **sendResultNotificationMessage** response returns from the application.

Event Data Records

Table 18–1 lists the IDs of the EDRs created by the EWS WAP Push communication service.

Table 18–1 Event Types Generated by EWS WAP Push/PAP

EDRID	Method Called
14001	sendPushMessage
14002	sendResultNotificationMessage

Statistics

Table 18–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 18–2 Methods and Transaction Types for EWS WAP Push/PAP

Method	Transaction type
sendPushMessage	TRANSACTION_TYPE_MESSAGE_SENDER_SEND
sendResultNotificationMessage	TRANSACTION_TYPE_MESSAGE_SENDER_NOTIFY

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing the EWS WAP Push/PAP Communication Service

This section describes the properties and workflow for the EWS WAP Push/PAP plug-in instance.

Properties for EWS WAP Push/PAP

Table 18–3 lists the technical specifications for the communication service.

Table 18–3 Properties for EWS WAP Push/PAP

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plugin_instance_id*

Table 18–3 (Cont.) Properties for EWS WAP Push/PAP

Property	Description

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=same as the network protocol *instance_id* assigned when the plug-in instance is created
	Type=com.bea.wlcp.wlng.plugin.pushmessage.pap.management.PushMessagePAPMBean

Network protocol plug-in service ID	Plugin_ews_push_message_pap
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide.*
Supported Address Scheme	tel, wapuser See "WAP User Address Scheme" for information on the wapuser address scheme.
Application-facing interface	com.bea.wlcp.wlng.ews.plugin.PushMessagePlugin com.bea.wlcp.wlng.ews.callback.PushMessageNotificationCallback
Service type	PushMessage
Exposes to the service communication layer a Java representation of:	Extended Web Services WAP Push
Interfaces with the network nodes using:	Push Access Protocol (PAP), 2.0. WAP-247-PAP-20010429-a
Deployment artifact: NT EAR wlng_nt_push_message_ews.ear	ews_push_message_service.jar, Plugin_ews_push_message_pap.jar, and ews_push_message_pap.war
Deployment artifact: AT EAR: Normal wlng_at_push_message_ews.ear	ews_push_message.war, ews_push_message_callback.jar, and rest_push_message.war
Deployment artifact: AT EAR: SOAP Only wlng_at_push_message_ews_soap.ear	ews_push_message.war and ews_push_message_callback.jar

WAP User Address Scheme

The wapuser address scheme supports the client address formats defined in the *Wireless Application Protocol Push Proxy Gateway Service Specification*.

To use the wapuser address scheme, the application should set the WAPPUSH and TYPE values in the **destinationAddress**. For example, given the address:

WAPPUSH=+155519990730

TYPE=PLMN@ppg.carrier.com

set the destinationAddress to:

WAPPUSH=+155519990730/TYPE=PLMN@ppg.carrier.com

Given the address:

WAPPUSH=john.doe%40wapforum.org

TYPE=USER@ppg.carrier.com

set the **destinationAddress** to:

WAPPUSH=john.doe%40wapforum.org/ TYPE=USER@ppg.carrier.com

Configuration Workflow for EWS WAP Push/PAP

Following is an outline for configuring the plug-in using the Administration Console.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID as listed in the "Properties for EWS WAP Push/PAP" section.
2. Using the Administration Console or an MBean browser, select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID assigned when the plug-in instance was created.
3. Define the characteristics of the PPG server to connect to:

   Attribute: PPGNotificationURL

   Attribute: PPGURL
4. Specify heartbeat behavior. See "Configuring Heartbeats" in *System Administrator's Guide*.
5. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for EWS WAP Push/PAP" section.
6. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
7. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Reference: Attributes for WAP Push/PAP

This section describes the attributes for configuration and maintenance:

- Attribute: BasicAuthentication
- Attribute: BAPassword
- Attribute: BAUser
- Attribute: PPGNotificationURL
- Attribute: PPGURL
- Attribute: ResultNotificationEndpoint

Attribute: BasicAuthentication

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether basic authentication is used.

Attribute: BAPassword

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the password used for basic authentication.

Attribute: BAUser

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the user used for basic authentication.

Attribute: PPGNotificationURL

Scope: Server

Unit: Not applicable

Format: String

Specifies the URL of the plug-in instance. Used by the Push Proxy Gateway (PPG) to send notifications of results to the plug-in instance.

Attribute: PPGURL

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the URL of the Push Proxy Gateway (PPG) the plug-in instance uses.

Attribute: ResultNotificationEndpoint

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether delivery reports are sent to the application. Set to `true` if delivery reports are sent, `false` if they are not.

19 Native MM7

This chapter describes the Native MM7 communication service in detail.

Overview of the Native MM7 Communication Service

The Native MM7 communication service exposes the 3GPP MM7 standard interfaces.

From the point of view of an application, the communication service acts as an MMS relay server. From the point of view of the network, it acts as an MMS VAS application.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

Using the Native MM7 communication service, an application can:

- Send a multimedia message to one or many destination addresses.

  The payload in these multimedia messages can be any type that can be specified using Multipurpose Internet Mail Extensions (MIME), including multipart messages. If a subscription for notifications has been previously set up, the request can also specify that a delivery report or a read report should be returned later in relation to this message.
- Receive delivery reports on sent multimedia messages that have arrived from the network.
- Receive read-reply reports on sent multimedia messages that have arrived from the network.
- Receive multimedia messages from the network.

Requests can flow in two directions using the Native MM7 communication service: from the application to the network and from the network to the application.

Status Reports

There are two types of status reports that can be returned to the application from the network via Services Gatekeeper. Both are returned asynchronously, using callback information provided when the notification is set up. If the network sends a report but no notification has been set up, Services Gatekeeper sends the network an error code indicating permanent failure.

- Delivery Reports
- Read-Reply Report

Delivery Reports

Delivery reports are acknowledgements that the network node has handled the message from the application that was submitted. The report indicates the status of the message: for example, Forwarded, Expired, or Rejected. There is one delivery report per destination address. If a connection error occurs within Services Gatekeeper or between Services Gatekeeper and the application, an error code is returned to the network, which resends the message.

Read-Reply Report

Read-reply reports contain the final delivery status of the multimedia message. The final delivery status reports whether the message has actually been delivered by the network to the mobile terminal. It also includes the status of the message at that terminal; for example, Read or Deleted without being read.

Because a recipient can request that read-reply reports not be generated, lack of a read-reply report does not necessarily mean that the message has not been rendered on the recipient’s terminal.

There is one read-reply report per destination address. If a connection error occurs within Services Gatekeeper or between Services Gatekeeper and the application, an error code is returned to the network, which resends the message.

Network-triggered Multimedia Messages

For an application to receive multimedia messages from the network, it must register its interest in these messages by setting up a subscription. A subscription, or notification, is defined by a destination address. For the message to be accepted by Services Gatekeeper, the destination address must match the subscription. Each registered subscription must be unique, and subscription attempts with overlapping criteria are rejected. If a message with several destination addresses arrives, Services Gatekeeper iterates through the list until it reaches a match or until the list is exhausted.

Application Interfaces

For information about the application interface for the Native MM7 communication service, see the discussion of Native Interfaces in *Application Developer’s Guide*.

Events and Statistics

The Native MM7 communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 19–1 lists the IDs of the EDRs created by the Native MM7 communication service.

Table 19–1 Event Types Generated by Native MM7

EDR ID	Description
401000	An application-initiated message has entered the plug-in.

Table 19–1 (Cont.) Event Types Generated by Native MM7

EDR ID	Description
401001	An application-initiated message has exited the plug-in.
401002	A network-triggered message sent via v.1.0 has entered the plug-in.
401003	A network-triggered message has exited the plug-in. It is formatted according to v. 1.2.
401004	A delivery report using v. 1.0 has entered the plug-in.
401005	A delivery report has exited the plug-in. It is formatted according to v 1.2
401006	A read-reply report using v. 1.0 has entered the plug-in.
401007	A read-reply report has exited the plug-in. It is formatted according to v. 1.2

Charging Data Records

Native MM7 -specific CDRs are generated under the following conditions:

- After an MMS message has been successfully sent from the application to the network.
- After an MMS message has been successfully sent from the network to the application.
- After a delivery report has been successfully delivered to the application.
- After a read-reply report has been successfully delivered to the application.

Statistics

Table 19–2 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 19–2 Methods and Transaction Types for Native MM7

Method	Transaction type
submit	TRANSACTION_TYPE_MESSAGING_MMS_SEND
deliver	TRANSACTION_TYPE_MESSAGING_MMS_RECEIVE

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing Native MM7

This section describes the properties and workflow for the Native MM7 communication service.

Properties for Native MM7

Table 19–3 lists the technical specifications for the communication service.

Table 19–3 Properties for Native MM7

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plugin_instance_id*

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=same as the network protocol *instance_id* assigned when the plug-in instance is created
	Type=com.bea.wlcp.wlng.plugin.mm7.management.Mm7MBean

Network protocol plug-in service ID	Plugin_multimedia_messaging_mm7
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*.
Supported Address Scheme	tel, mailto, short
Application-facing interfaces	com.bea.wlcp.wlng.mm7.plugin.MmsPlugin com.bea.wlcp.wlng.mm7.callback.MmsVaspCallback
Service type	Mm7
Exposes to the service communication layer a Java representation of:	3GPP TS 23.140 V5.3.0 (REL-5-MM7-1-2.xsd)
Interfaces with the network nodes using:	MM7 (REL-5-MM7-1-0 or REL-5-MM7-1-2)
Deployment artifacts	Plugin_multimedia_messaging_mm7.jar, mm7_service.jar, 1_0_mm7_vasp.war, 1_2_mm7_vasp.war packaged in wlng_nt_multimedia_messaging_mm7.ear mm7.war, mm7_callback_client.jar, packaged in wlng_at_multimedia_messaging_mm7.ear

Configuration Workflow for Native MM7

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID as listed in the "Properties for Native MM7" section.
2. Using the Administration Console or an MBean browser, select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID assigned when the plug-in instance was created.
3. Configure the behavior of the plug-in instance:
   - Attribute: Mm7RelayServerAddress
   - Attribute: HTTPBasicAuthentication. If using HTTP basic authentication also define:
     – Attribute: HTTPBasicAuthenticationUsername
     – Attribute: HTTPBasicAuthenticationPassword
   - Attribute: XSDVersion

4. Specify heartbeat behavior. See "Configuring Heartbeats" in *System Administrator's Guide*.
5. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Native MM7" section.
6. Provide the administrator of the MM7 server with the URL to which the MM7 server should deliver mobile-originated messages and delivery reports:
   - For REL-5-MM7-1-0 the default URL is:

     ```
     http://IP_Address_of_NT_ Server:port/1_0_mm7_vasp/mms_vasp
     ```

   - For REL-5-MM7-1-2 the default URL is:

     ```
     http://IP_Address_of_NT Server:port/1_2_mm7_vasp/mms_vasp
     ```

7. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
8. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Provisioning Workflow for Native MM7

Following is an outline for provisioning Native MM7.

1. Register offline notifications. This means that mobile-originated messages should not result in notifications to an application, but instead be stored in Services Gatekeeper for polling. Use "Operation: addVASPIDMapping" to register offline notifications. Use the following operations to manage the offline registrations:
   - Operation: listAllVASIDMapping
   - Operation: addVASPIDMapping
   - Operation: removeReceiveMmsNotification
2. Register online notifications. This means that registrations for mobile-originated messages are managed on behalf of an application. Use "Operation: listVASIDMapping" to register online notifications. Use the following operations to manage the online registrations:
   - Operation: listAllVASPIDMapping
   - Operation: enableReceiveMmsNotification
   - Operation: removeStatusReporting

Reference: Attributes and Operations for Native MM7

This section describes the attributes and operations for configuration and maintenance:

- Attribute: HTTPBasicAuthentication
- Attribute: HTTPBasicAuthenticationUsername
- Attribute: HTTPBasicAuthenticationPassword
- Attribute: Mm7RelayServerAddress
- Attribute: XSDVersion

- Operation: addVASIDMapping
- Operation: addVASPIDMapping
- Operation: enableReceiveMmsNotification
- Operation: listAllVASIDMapping
- Operation: listAllVASPIDMapping
- Operation: removeReceiveMmsNotification
- Operation: removeStatusReporting
- Operation: removeVASIDMapping
- Operation: removeVASPIDMapping

Attribute: HTTPBasicAuthentication

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if HTTP basic authentication shall be used for authentication with the MM7 server.

Set to `true` if HTTP basic authentication shall be used, otherwise `false`.

If `true`, "Attribute: HTTPBasicAuthenticationUsername" and "Attribute: HTTPBasicAuthenticationPassword" must be specified.

Attribute: HTTPBasicAuthenticationUsername

Scope: Cluster

Unit: Not applicable

Format: String

The user name to use for HTTP basic authentication towards the MM7 server. This is equivalent to the Application Instance ID.

Attribute: HTTPBasicAuthenticationPassword

Scope: Cluster

Unit: Not applicable

Format: String

The password to use for HTTP basic authentication towards the MM7 server.

Attribute: Mm7RelayServerAddress

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the address of the MM7 Relay Server. The address is an HTTP URL.

Attribute: XSDVersion

Scope: Cluster

Unit: Not applicable

Format: String [REL-5-MM7-1-0, REL-5-MM7-1-2]

Specifies the xsd version that should be used for requests towards the MMSC.

Enter one of the following:

- **REL-5-MM7-1-0** to use an altered version of the **REL-5-MM7-1-0.xsd**. The altered version allows the use of delivery notifications when the MMC-S requires this version of the xsd. This is a requirement when connecting to, among others, Comverse MMSCs.
- **REL-5-MM7-1-2** to use **REL-5-MM7-1-2.xsd**.

Operation: addVASIDMapping

Scope: Cluster

Adds the service provider VAS ID and the Services Gatekeeper VAS ID mapping for use with the submit request.

Signature:

```
addVASIDMapping(spVasID: String, wlngVasID: String)
```

Table 19–4 addVASIDMapping Parameters

Parameter	Description
spVasID	The service provider's VAS ID.
wlngVasID	The VAS ID that is to be sent to the MMSC.

Operation: addVASPIDMapping

Scope: Cluster

Adds the service provider VASP ID and the Oracle Communications Services Gatekeeper VASP ID mapping for use with the **SubmitReq** operation.

Signature:

```
addVASPIDMapping(spVaspID: String, wlngVaspID: String)
```

Table 19–5 addVASPIDMapping Parameters

Parameter	Description
spVaspID	The service provider's VASP ID.
wlngVaspID	The VASP ID that is to be sent to the MMSC.

Operation: enableReceiveMmsNotification

Scope: Cluster

Sets up delivery notification for network-triggered message delivery. Messages matching this configuration result in a callback to the application by the **DeliverReq** operation.

Signature:

```
enableReceiveMmsNotification(Address: String, appInstGroupID: String,
applicationURI: String)
```

Table 19–6 enableReceiveMmsNotification Parameters

Parameter	Description
Address	The destination address of the MMS message
appInstGroupID	The application instance group ID associated with this notification
applicationURI	The URI where the application can be reached

Operation: enableStatusReporting

Scope: Cluster

Sets up status report notification for delivery status and read-reply status for application-initiated messages.

Signature:

```
enableStatusReporting(appInstGroupID: String, applicationURI: String)
```

Table 19–7 enableStatusReporting Parameters

Parameter	Description
appInstGroupID	The application instance group ID associated with this notification
applicationURI	The URI where the application can be reached

Operation: getReceiveMmsNotificationForAddress

Scope: Cluster

Checks to see if a notification is already set up for this exact address. If no entry is returned, the address can be used to set up a new notification

Signature:

```
getReceiveMmsNotificationForAddress(Address: String)
```

Table 19–8 getReceiveMmsNotificationForAddress Parameters

Parameter	Description
Address	The destination address of the MMS message

Operation: getReceiveMmsNotificationMatches

Scope: Cluster

Searches existing configurations for Receive Mms notifications for an address pattern, using regular expressions. If there are no matching configurations, null is returned. This can be used to find out which application is registered for notifications for a specific address.

This operation returns at most 1 match. If there are multiple entries, only the first will be returned, although in normal operation overlapping entries should not be possible

Signature:

```
getReceiveMmsNotificationMatches(Address: String)
```

Table 19–9 getReceiveMmsNotificationMatches Parameters

Parameter	Description
Address	The destination address pattern of the MMS: for example, 1234 or .@oracle.com

Operation: listAllVASIDMapping

Scope: Cluster

Lists all VAS ID mappings

Signature:

```
listAllVASIDMapping()
```

Operation: listAllVASPIDMapping

Scope: Cluster

Lists all VASP ID mappings

Signature:

```
listAllVASPIDMapping()
```

Operation: listReceiveMmsNotifications

Scope: Cluster

Lists all established delivery notifications

Signature:

```
listReceiveMmsNotifications()
```

Operation: listStatusReportingNotifications

Scope: Cluster

Lists all established status report notifications

Signature:

```
listStatusReportingNotifications()
```

Operation: listVASIDMapping

Scope: Cluster

Lists the VAS ID mapping that corresponds to the specified **spVasID**

Signature:

```
listVASIDMapping(spVasID: String)
```

Table 19–10 listVASIDMapping Parameters

Parameter	Description
spVasID	The service provider VAS ID to be matched.

Operation: listVASPIDMapping

Scope: Cluster

Lists the VASP ID mapping that corresponds to the specified **spVaspID**

Signature:

```
listVASPIDMapping(spVaspID: String)
```

Table 19–11 listVASPIDMapping Parameters

Parameter	Description
spVaspID	The service provider VASP ID to be matched.

Operation: removeReceiveMmsNotification

Scope: Cluster

Removes an existing delivery notification

Signature:

```
removeReceiveMmsNotification(address: String)
```

Table 19–12 removeReceiveMmsNotification Parameters

Parameter	Description
address	The destination address of the MMS message

Operation: removeStatusReporting

Scope: Cluster

Removes an existing status report notification.

Signature:

```
removeStatusReporting(applicationInstanceGroupID: String)
```

Table 19–13 removeStatusReporting Parameters

Parameter	Description
applicationInstanceGroupID	The application instance group ID associated with the notification.

Operation: removeVASIDMapping

Scope: Cluster

Removes an established VAS ID mapping.

Signature:

```
removeVASIDMapping(spVasID: String)
```

Table 19–14 removeVASIDMapping Parameters

Parameter	Description
spVasID	The service provider VAS ID whose mapping is to be removed

Operation: removeVASPIDMapping

Scope: Cluster

Removes an established VASP ID mapping.

Signature:

```
removeVASIPDMapping(spVasID: String)
```

Table 19–15 removeVASPIDMapping Parameters

Parameter	Description
spVaspID	The service provider VASP ID whose mapping is to be removed

20 Native SMPP

This chapter describes the Native SMPP communication service in detail.

Overview of the Native SMPP Communication Service

The Native SMPP communication service exposes the SMPP v. 3.4 standard interfaces.

The communication service acts as an External Short Message Entity (ESME) that connects to a Short Messaging Service Center (SMSC) over TCP/IP.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, seethe appendix on standards and specifications in *Concepts Guide*.

The Native SMPP communication service access the network using the following network protocols:

- SMPP v 3.4
- SMPP v 5.0

  SMPP v5.0 supports the billing identification parameter and ussd service_ operation as optional parameters to **Deliver_SM**. See "smpp_billing_id" and "ussd_service_operation" for more information.

Using the Native SMPP communication service, an application can:

- Send a short message to one or many destination addresses.
- Cancel a previously sent message that has not yet been delivered.
- Replace a previously sent message that has not yet been delivered.
- Query the delivery status of a previously sent message.
- Receive short messages arrived from the network.
- Receive the delivery status of a previously sent message.

Requests flow in two directions: from the application to the network and from the network to the application.

All Native SMPP components are deployed in the network tier.

SMPP Server Service

The core module of the Native SMPP communication service is an SMPP Server Service deployed as an Oracle WebLogic Server Service. It provides connection services for the Native SMPP and Parlay X 2.1 Short Messaging plug-ins. The SMPP Server Service:

- Receives SMPP data from the socket.
- Constructs the SMPP protocol data unit (PDU).
- Associates the current PDU with the correct application instance.
- Invokes the plug-in.
- Manages connections between Services Gatekeeper and applications.
- Manages connections between Services Gatekeeper and Short Message Service Centers (SMSCs).

Because the SMPP Server Service is deployed in the network tier, applications using the Native SMPP Native communication service must be able to connect directly to the network tier. Firewalls must be configured to allow connection to the ports defined for the SMPP Server Service.

Connection Handling and Provisioning

Plug-in instances establish connections to Services Gatekeeper using facilities provided by the SMPP Server Service.

The connection ID with the application is created on a successful bind with the application. The connection ID with the SMSC is created on a successful bind with the SMSC. The connection ID changes on a successful rebind.

Error handling for establishing connections is as follows:

- If at least one plug-in instance successfully binds to the SMSC, Services Gatekeeper sends a successful bind response to the application and establishes a client connection. All client connections that failed to bind attempt to reconnect periodically.

  While a client connection is attempting to reconnect with the SMSC, its corresponding server connection continues to receive requests from the application. The plug-in will not be able to process these requests. In this case, the server connection sends an error response to the application. The requests are not stored and not re-sent to the SMSC.
- If all of the plug-in instances fail to bind, Services Gatekeeper sends a failure bind response to the application and closes and removes the server connection.
- When a client connection is successfully established, the connection is verified periodically using **ENQUIRE_LINK** requests (heartbeats). If the **ENQUIRE_LINK** requests fail a configurable number of times, Services Gatekeeper attempts to reconnect with the SMSC. If the reconnect attempts fail a configurable number of times, the client connection is closed and removed. See Attribute: EnquireLinkMaxFailureTimes and Attribute: RetryTimesBeforeGiveUp.

Applications can bind to Services Gatekeeper as a transmitter, a receiver, or a transceiver. An application can establish several parallel sessions by issuing multiple bind operations.

The number of concurrent connections is provisioned for each Native SMPP plug-in, if connection-based routing is not enabled. See Attribute: BindType, Attribute: NumberReceiverConnections, Attribute: NumberTransceiverConnections, and Attribute: NumberTransmitterConnections.

The SMPP Server Service should be provisioned with the following data about the application instance:

- The port number to bind to.

- The maximum number of concurrent sessions allowed.
- Whether subsequent operations should be allowed to target a previously sent short message.
- Whether network-triggered short messages and delivery reports should be forwarded to the application.
- The address range that, when matched with the destination address of a network-triggered short message, forwards the message to the application.

See "Operation: addApplicationSpecificSettings" for details about configuring these settings.

Authentication

Authentication credentials are configured in the Native SMPP plug-in instance MBean. See "Reference: Attributes and Operations for Native SMPP Plug-in"for more information.

Applications use an application instance ID as the ESME system_id and the related password when binding to Services Gatekeeper.

Connection Pooling

The SMPP Server Service maintains server and client connection pools.

Server Connection Pools

The SMPP Server Service maintains a server connection pool for application-facing (northbound) connections. The pool is created when the SMPP Server Service is started.

The plug-in obtains connections from this pool to send messages to the application.

The server connections are used to:

- Invoke the plug-in.
- Send messages to and receive messages from the application.
- Manage the application-facing SMPP timers.
- Manage windowing toward the application.
- Cache transaction mapping information for transactions between Services Gatekeeper and the application.

Client Connection Pools

The SMPP Server Service maintains a client connection pool for network-facing (southbound) connections.

The plug-in sends **BIND** and **UNBIND** requests to the client pool and obtains a client connection ID from the pool to perform SMPP transactions.

The client connections are used to:

- Invoke the plug-in.
- Send messages to and receive messages from the SMSC.
- Manage the network-facing SMPP timers.
- Manage windowing toward the SMSC.

- Cache transaction mapping information between Services Gatekeeper and the SMSC.

Timeouts

You can configure timers for both application-facing and network-facing connections. Some of the timers for application-facing and network-facing connections have the same names, but they are configured in different MBeans.

SMPP Server Service Timers

The SMPP Server Service provides the following configurable timers for connections between Services Gatekeeper and applications:

- Initiation timer: This timer ensures that when an application initiates a connection, the **BIND** occurs within a specified period after the connection is established to Services Gatekeeper. See "Attribute: InitiationTimerValue" for more information.
- Inactivity timer: This timer establishes a period of inactivity after which, if no SMPP messages are exchanged with the application, Services Gatekeeper closes the connection. See "Attribute: InactivityTimerValue" for more information.
- Connection timer: This timer sets the heartbeat interval that Services Gatekeeper uses to request the connection status on the server connection. If the **ENQUIRE_LINK** requests fail, Services Gatekeeper closes the connection and attempts to reconnect. See "Attribute: EnquireLinkTimerValue" in "Reference: Attributes and Operations for SMPP Server Service" for more information.
- Transaction timer: This timer establishes the interval between an SMPP request to the application and the corresponding SMPP response. If the interval is reached, Services Gatekeeper does not re-send the request. In this case, Services Gatekeeper removes the transaction information and discards the PDU response. See "Attribute: RequestTimerValue" in "Reference: Attributes and Operations for SMPP Server Service" for more information.

You can disable any of these timers by setting their values to 0.

Plug-in Instance Timers

The plug-in instance MBean provides the following configurable timers for connections between Services Gatekeeper and SMSCs:

- Connection timer: This timer sets the heartbeat interval that Services Gatekeeper uses to request the connection status on the client connection. If the **ENQUIRE_LINK** requests fail, Services Gatekeeper closes the connection and attempts to reconnect. See "Attribute: EnquireLinkTimerValue" in "Reference: Attributes and Operations for Native SMPP Plug-in" for more information.
- Transaction timer: This timer establishes the interval between an SMPP request to the SMSC and the corresponding SMPP response. If the interval is reached, Services Gatekeeper does not re-send the request. In this case, Services Gatekeeper removes the transaction information and discards the PDU response. See the "Attribute: RequestTimerValue" in "Reference: Attributes and Operations for Native SMPP Plug-in" for more information.

Windowing

To maximize throughput, Native SMPP supports windowing on both the application-facing and network-facing interfaces. Windowing provides a way to specify the amount of data that can be transmitted without receiving an acknowledgment.

Requests wait in a windowing queue until they can be submitted. Two values apply to the windowing queue. The windowing maximum queue size is the size of the queue, specifying the maximum number of requests that can wait in the queue at one time. The windowing maximum wait time value specifies the maximum amount of time that a single request can wait in the windowing queue.

The windowing size value is the number of unacknowledged requests that can be sent simultaneously.

Windowing for mobile-originated requests toward the application is configured in the following parameters in the SMPP Server Service's **addApplicationSpecificSettings** operation:

- **windowingSize**
- **windowingMaxQueueSize**
- **windowingMaxWaitTime**

See "Operation: addApplicationSpecificSettings" for more information.

Windowing for mobile-terminated requests toward the SMSC is configured in the following plug-in instance MBean attributes:

- Attribute: WindowingSize
- Attribute: WindowingMaxQueueSize
- Attribute: WindowingMaxWaitTime

A request moves from the windowing queue to the window. From the window it is submitted for processing. A submitted request remains in the window until its response is received. When the response is received, the request is released and another request can be moved from the windowing queue to the window.

If any one of these three windowing parameters is set to a value less than zero, windowing is turned off. If all of these three parameters are greater than zero, windowing is turned on.

In both directions, if the windowing request queue is full or the timer has expired, the request is not sent and an error code is returned to the plug-in instance.

Connection-Based Routing

Connection-based routing lets network operators configure geo-redundant sites to allow applications to send mobile-originated (MO), mobile-terminated (MT,) and delivery receipt (DR) traffic to and from any of the redundant sites. For example, a DR can be sent to a site other than the one through which the original message was submitted.

Enable Connection-Based Routing

To use this feature, set the **ConnectionBaseRouting** attribute in the SMPP Server Service to `true.` By default this attribute is `false`. See "Attribute: ConnectionBasedRouting" for more information.

When connection-based routing is enabled, messages from the network are routed to the application that caused or that could have caused the connection in the plug-in to be established to the SMSC. This works both for delivering a short message with a new message and delivering a short message containing a delivery receipt. This means that

DELIVER_SM with a new message is not routed based on the destination address, and **DELIVER_SM** containing a delivery receipt is not routed based on the message identifier.

Limitations

The following are some limitations and issues pertaining to connection-based routing:

- If an application is configured to support subsequent operations (**CANCEL_SM**, **QUERY_SM** and **REPLACE_SM**), those requests must be sent to the same geographic site as the original submit requests. They will not be accepted if sent to the other site. When Services Gatekeeper checks the subsequent operations, it returns an error response if it cannot find the original **SUBMIT_SM** request in the store.
- If subsequent operations are enabled and a submit request is sent through site 1 but delivery receipt arrives on site 2, the data stored about the message in the database on site 1 is not deleted until the information is considered to be too old. The consequence is that an application can continue sending subsequent operations related to the message through site 1 even after the message was delivered.
- If connection-based routing is enabled, the **NumberReceiverConnections**, **NumberTransceiverConnections**, and **NumberTransmitterConnections** attributes in the plug-in instance are ignored, because connections to the SMSC cannot be shared among different application instances.

Short Code Translation

The Native SMPP communication service does not offer short code translations.

USSD Support

Native SMPP provides Unstructured Supplementary Services Data (USSD) through the **its_session_info**, **service_type**, and **ussd_service_operation** optional parameters.

its_session_info

Required parameter for the CDMA Interactive Teleservice as defined by the Korean PCS carriers [KORITS]. Contains control information for the interactive session between an MS and an ESME.

See Section 5.3.2.43 of the *Short Message Peer to Peer Protocol Specification v3.4* for the formal definition of the parameter and the appropriate subsections of Section 4 for its specification as an optional parameter for **SUBMIT_SM**, **DELIVER_SM**, and **DATA_SM**.

Format

Octet String

Following is a description of the octet string.

Bits 7...............0

SSSS SSSS (octet 1)

NNNN NNNE (octet 2)

Octet 1 contains the session number (0 -255) encoded in binary. The session number remains constant for each session.

The sequence number of the dialog unit (as assigned by the ESME) within the session is encoded in bits [7. . . 1] of octet 2.

The End of Session Indicator indicates the message is the end of the conversation session and is encoded in bit 0 of octet 2 as follows:

- **0** = End of Session Indicator inactive
- **1** = End of Session Indicator active

service_type

Indicates the SMS application service associated with the message. Allows the ESME to use enhanced messaging services such as "replace_if_present" (generic) and to control the teleservice used on the air interface (for example, ANSI-136/TDMA, IS-95/CDMA).

Used to support USSD (Unstructured Supplementary Service Data 3G TS 23.090 version 3.0.0) messages through the SMPP protocol.

See Section 5.2.11 of the *Short Message Peer to Peer Protocol Specification v3.4*. for the formal definition of the parameter and the appropriate subsections of Section 4 for its specification as a mandatory parameter for **SUBMIT_SM**, **SUBMIT_MULTI**, **DELIVER_SM**, **DATA_SM**, and **CANCEL_SM**.

Format
Octet String

Value
The pre-defined generic service type value for USSD is **USSD**.

ussd_service_operation

Defines the USSD service operation that is required when SMPP is used as an interface to a (GSM) USSD system.

Used to support tunneling USSD (Unstructured Supplementary Service Data 3G TS 23.090 version 3.0.0) messages through the SMPP protocol.

Used as an optional parameter to SMPP **SUBMIT_SM**.

Defined in section Section 5.3.2.44 of the *Short Message Peer to Peer Protocol Specification v3.4*.

Added to **DELIVER_SM** in the SMPP 5.0 specification. See *Short Message Peer to Peer Protocol Specification Version 5.0*.

Format
Octet String

Value
Valid values are:

- **0** = PSSD indication
- **1** = PSSR indication
- **2** = USSR request
- **3** = USSN request
- **4** to **15** Reserved
- **16** = PSSD response

- **17** = PSSR response
- **18** = USSR confirm
- **19** = USSN confirm
- **20** to **31** Reserved
- **32** to **255** Reserved for vendor-specific USSD operations

Billing Identification

The native SMPP communication service supports the **billing_identification** parameter in the format in the SMPP Specification 5.0 through an optional parameter named **smpp_billing_id**.

The parameter works with SMPP 5.0 SMSCs, but with not with SMPP 3.4 SMSCs.

smpp_billing_id

Defines the billing information according to the format in the SMPP Specification 5.0, section 4.8.4.3 titled "billing_identification".

Format

Hexadecimal string

Table 20–1 describes the format.

Table 20–1 Format for smpp_bliing_id Value

Field	Size (octets)	Type	Description
parameter tag	2	Integer	0x060B
length	2	Integer	Length of value part in octets
value	1 - 1024	Octet String	Bits 7......0 0XXXXXXX (Reserved) 1XXXXXXX (Vendor Specific) The first octet represents the Billing Format tag and indicates the format of the billing information contained in the remaining octets.

If the value is not sent as a hexadecimal string, it is ignored and a warning is logged.

Here is sample code for encoding the string.

```
private String getHexEncodedString(String normalString) {
  byte[] bHexStr = normalString.getBytes();
  String retVal = "";
..String sOctet = null;
  for (int i = 0; i < bHexStr.length; i++) {
    sOctet = Integer.toHexString((int) (bHexStr[i] & 0xFF));
    if (sOctet.length() == 1) {
      sOctet = "0" + sOctet;
    }
    retVal = retVal.concat(sOctet);
  }
  return retVal.toUpperCase();
}
```

Load Balancing, High Availability and Fail-Over

To optimize system utilization, applications should load-balance application-triggered requests among all network tier servers.

The SMSC should load-balance network-triggered requests among all network tier servers.

Load balancing is supported only among plug-in instances that are located in same network tier server and share same large account. When a request is sent to a plug-in instance, the plug-in instances use the SMPP Server Service in the same server to forward the request to the applications. When a request is sent to the SMPP Server Service, the SMPP Server Service uses a plug-in instance in the same server to process the request.

High availability and fail-over is supported between Services Gatekeeper and the SMSC. High availability between the application and Services Gatekeeper must be handled by each application.

A prerequisite for high-availability for the Native SMPP communication service is redundant network tier servers, redundant network interface cards in each network tier server, and a redundant set of SMPP servers to connect to. High availability between Services Gatekeeper and the network is achieved by using at least two different plug-in instances per network tier server and having the plug-in instances connect to different SMPP servers.

Between SMPP applications and Services Gatekeeper, the applications handle high availability and fail-over for application-initiated requests by binding to two or more network tier servers. For network-triggered requests, the same requirement that the applications bind to two or more network tier servers applies.

High availability behavior is as follows:

- In a Services Gatekeeper cluster, if the server becomes unavailable after sending a Submit SM request to and receiving the **SUBMIT_SM_RESP** from the SMSC, the SMSC routes the subsequent delivery receipt to another server. This other server retrieves the message information from cluster-level storage and processes it.
- In a Services Gatekeeper cluster, if a server becomes unavailable after sending a **SUBMIT_ SM** request to and receiving the **SUBMIT_SM_RESP** from an application, the application routes the subsequent **CANCEL_SM**, **QUERY_SM** or **REPLACE_SM** request to another server. This other server retrieves the message information from cluster-level storage and processes it.
- In a geo-redundant configuration, all sites are connected to the SMSC. If a site becomes unavailable after sending a **SUBMIT_ SM** request to and receiving the **SUBMIT_SM_RESP** from the SMSC, the SMSC routes the subsequent delivery receipt to another site. This other site uses connection-based routing to process the delivery receipt.
- In a geo-redundant configuration, if an application is configured to support subsequent operations (**CANCEL_SM**, **QUERY_SM**, and **REPLACE_SM**) through the **subsequentOperationsAllowed** parameter to the **addApplicationSpecificSettings** operation, those requests must be sent to the same geographic site from which the original submit requests were sent. They will not be accepted if they are sent to another site.
- In a geo-redundant configuration, if an application is configured to support subsequent operations and a submit request is sent through site 1 but delivery receipt arrives on site 2, the data stored about the message in the database on site 1 is not deleted until the information is considered to be too old. The consequence is

that an application can continue sending subsequent operations related to the message through site 1 even after the message was delivered.

The Native SMPP communication service can be provisioned for applications to share the same large account in the SMPP server, so that they share the same bind. However, his configuration is not recommended since it impacts high availability for network-triggered requests. When there is only one bind between Services Gatekeeper and the SMPP server, and more than one application is listening for network-triggered messages, the Native SMPP communication service must listen to incoming messages on behalf of all the applications. The bind between the plug-in and the network node is performed on all network tier servers, so network-triggered messages can be sent to any of these servers. If the network-triggered request ends up in a server that the application has not bound to, the communication service does not try to look up a server that the application has bound to. Instead, it does not see an active bind and treats the request as undeliverable to the application. Because it is common for SMPP servers to load-balance between binds, it is very likely that 50% or more of the requests will fail in this setup. The only way to ensure high availability in this scenario is to mandate that all applications bind to all network tier servers.

Application Interfaces

For information about the application interface for the Native SMPP communication service, see the discussion of Native Interfaces in *Application Developer's Guide*.

Events and Statistics

The Native SMPP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 20–2 lists IDs of the EDRs created by the SMPP Server Service.

Table 20–2 Event Types Generated by the SMPP Server Service

EDR ID	Description
400000	Entering the NorthChannelProcessor recvBind method.
400001	Entering the NorthChannelProcessor recvUnbind method.
400002	Entering the NorthChannelProcessor recvSubmitSM method.
400003	Leaving the NorthChannelProcessor sendSubmitSMResp method.
400004	Entering the NorthChannelProcessor recvSubmitMulti method.
400005	Leaving the NorthChannelProcessor sendSubmitMultiResp method.
400006	Entering the NorthChannelProcessor recvQuerySM method.
400007	Leaving the NorthChannelProcessor sendQuerySMResp method.
400008	Entering the NorthChannelProcessor recvCancelSM method.
400009	Leaving the NorthChannelProcessor sendCancelSMResp method.
400010	Entering the NorthChannelProcessor recvReplaceSM method.
400011	Leaving the NorthChannelProcessor sendReplaceSMResp method.

Table 20–2 (Cont.) Event Types Generated by the SMPP Server Service

EDR ID	Description
400020	Leaving the SouthChannelProcessor sendBind method.
400021	Leaving the SouthChannelProcessor sendUnbind method.
400022	Leaving the SouthChannelProcessor sendSubmitSM method.
400023	Entering the SouthChannelProcessor recvSubmitSMResp method.
400024	Leaving the SouthChannelProcessor sendSubmitMulti method.
400025	Entering the SouthChannelProcessor recvSubmitMultiResp method.
400026	Leaving the SouthChannelProcessor sendQuerySM method.
400027	Entering the SouthChannelProcessor recvQuerySMResp method.
400028	Leaving the SouthChannelProcessor sendCancelSM method.
400029	Entering the SouthChannelProcessor recvCancelSMResp method.
400030	Leaving the SouthChannelProcessor sendReplaceSM method.
400031	Entering the SouthChannelProcessor recvReplaceSMResp method.
400051	Entering the NorthChannelProcessor recvUnbindResp method.
400054	Entering the NorthChannelProcessor recvGenericNack method.
400055	Entering the NorthChannelProcessor sendBindResp method.
400056	Leaving the NorthChannelProcessor sendUnbind method.
400057	Leaving the NorthChannelProcessor sendUnbindResp method.
400060	Leaving the NorthChannelProcessor sendGenericNack method.
400061	Entering the SouthChannelProcessor recvBindResp method.
400062	Entering the SouthChannelProcessor recvUnbind method.
400063	Entering the SouthChannelProcessor recvUnbindResp method.
400066	Entering the SouthChannelProcessor recvGenericNack method.
400067	Leaving the SouthChannelProcessor sendUnbindResp method.
400070	Leaving the SouthChannelProcessor sendGenericNack method.
400100	Leaving the NorthChannelProcessor sendDeliverSM method.
400101	Entering the NorthChannelProcessor recvDeliverSMResp method.
400108	Entering the SouthChannelProcessor recvDeliverSM method.
400109	Leaving the SouthChannelProcessor sendDeliverSMResp method.

The Native SMPP plug-in instance does not exchange events directly with the application or the SMSC, so it does not generate any EDRs.

Charging Data Records

Native SMPP plug-in-specific CDRs are generated under the following conditions:

- After a successful MT **SUBMIT_SM_RESP** operation from the application to the network.
- After a successful MT **SUBMIT_MULTI_RESP** operation from the application to the network.

- After a successful MO **DELIVER_SM_RESPONSE** operation from the network to the application.

Statistics

Table 20–3 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counters.

Table 20–3 Methods and Transaction Types for Native SMPP

Method	Transaction type
submitSm	TRANSACTION_TYPE_MESSAGING_SEND
submitSmMulti	TRANSACTION_TYPE_MESSAGING_SEND
receiveMoReq	TRANSACTION_TYPE_MESSAGING_RECEIVE

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing Native SMPP

This section describes the properties and workflow for the Native SMPP communication service.

Properties for SMPP Server Service

Table 20–4 lists the technical specifications for the SMPP Server Service.

Table 20–4 SMPP Server Service Properties

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Container Services > SMPPService

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng
	InstanceName=SMPPService
	Type=oracle.ocsg.protocol.smpp.management.SMPPServiceMBean

Exposes this interface to applications	Short Message Peer to Peer, Protocol Specification v3.4
Deployment artifacts	oracle.ocsg.protocol.smpp_api_5.0.0.0.jar, oracle.ocsg.protocol.smpp_5.0.0.0.jar

Properties for Native SMPP Plug-in

Table 20–5 lists the technical specifications for the Native SMPP plug-in.

Table 20–5 Native SMPP Plug-in Properties

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > Plugin_sms_smpp#5.0

Table 20–5 (Cont.) Native SMPP Plug-in Properties

Property	Description

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=same as the network protocol *instance_id* assigned when the plug-in instance is created
	Type=oracle.ocsg.plugin.nativesmpp.management.NativeSMPPPluginMBean

Deployment name	wlng_nt_native_smpp_sms#5.0
Network protocol plug-in service ID	Plugin_sms_smpp
Network protocol plug-in instance ID	The ID is assigned when the plug-in instance is created. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide.*
Exposes to the service communication layer a Java representation of:	SMPP v3.4, depends on common SMPP server service Short Message Peer to Peer Protocol Specification v3.4
Interfaces with the network nodes using:	SMPP v3.4, depends on common SMPP server service4 Short Message Peer to Peer Protocol Specification v3.4
Service Type	SMPP
Application-facing interfaces	oracle.ocsg.protocol.smpp.plugin.SMPPPluginNorth oracle.ocsg.protocol.smpp.service.SMPPServiceNorth
Network-facing interfaces	oracle.ocsg.protocol.smpp.plugin.SMPPPluginSouth oracle.ocsg.protocol.smpp.service.SMPPServiceSouth
Supported Address Scheme	tel
Deployment artifact	wlng_nt_native_smpp_sms.ear

Configuration Workflow for Native SMPP Communication Service

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Navigate to **Container Services** and then **SMPPService.**
2. Configure the behavior of the SMPP Server Service.

   See "Reference: Attributes and Operations for SMPP Server Service" for descriptions of the configuration options.
3. Using "Operation: updateAllServerPorts", apply the configuration settings for the Native SMPP Service.
4. Create one or more instances of the plug-in service. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the network protocol plug-in service ID as described in the "Properties for Native SMPP Plug-in" section.
5. Using the console or an MBean browser, select the MBean for the plug-in instance that you want to configure. The MBean display name is the same as the plug-in instance ID assigned when the plug-in instance was created.

6. Configure the behavior of the plug-in instance. See "Reference: Attributes and Operations for Native SMPP Plug-in" for the list of attributes that you can set.
7. Apply the configuration settings for the Native SMPP plug-in instance by restarting the plug-in or using "Operation: resetClientConnection".
8. Set up the routing rules to the plug-in instance. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in instance ID and address schemes listed in the "Properties for Native SMPP Plug-in" section.
9. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
10. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Provisioning Workflow for Native SMPP Communication Service

Following is an outline of tasks for provisioning the communication service.

1. To register application instances to use the Native SMPP communication service, use "Operation: addApplicationSpecificSettings" in the MBean for the SMPP Server Service. Use the following operations to manage the account settings:
   - Operation: addApplicationSpecificSettings
   - Operation: deleteApplicationSpecificSettings
2. Using "Operation: updateAllServerPorts", apply the provisioning settings.

System Properties for SMPP Server Service

The SMPP Server Service has some system properties that cannot be modified at runtime. Set these properties on the Java command line when you start Services Gatekeeper.

These system properties are applicable to both Native SMPP and Parlay X SMS/SMPP plug-ins.

System Property: oracle.ocsg.protocol.smpp.serverservice.max_threads

Format: Integer

Maximum number of threads available to server connections.

The default is **32**.

System Property: oracle.ocsg.protocol.smpp.serverservice.min_threads

Format: Integer

Minimum number of threads available to client and server connections. Each client connection uses one thread. Each server port uses one thread.

The default is **2**.

System Property: wlng.legacy.smpp.PDUManipulationAllowed

Format: Boolean

Specifies whether an interceptor can modify a parameter passed between the SMPP Server Service and a plug-in.

Set to `true` to allow parameter modification, `false` to prohibit it.

The default is `true`.

System Property: wlng.smpp.max_payload_size

Format: Integer

Specifies the maximum number of characters in an SMS message.

The default is the maximum defined by the Parlay X 2.1 SMS specification: 160 GSM 7-bit characters or 70 Unicode characters.

Reference: Attributes and Operations for SMPP Server Service

The attributes listed in this section are used only by the Native SMPP communication service.

All of the operations are used by the Native SMPP communication service, but only the following four are used by the Parlay X 2.1 Short Messaging/SMPP and Extended Web Services Binary SMS/SMPP communication services:

- Operation: closeClientConnection
- Operation: listClientConnections
- Operation: listPluginInstances
- Operation: resetClientConnection

This section describes the attributes and operations for configuration and maintenance.

- Attribute: ConnectionBasedRouting
- Attribute: EnquireLinkMaxFailureTimes
- Attribute: EnquireLinkTimerValue
- Attribute: InactivityTimerValue
- Attribute: InitiationTimerValue
- Attribute: LooseBinding
- Attribute: OfflineMO
- Attribute: RequestTimerValue
- Attribute: ServerAddress
- Attribute: ServerPort
- Attribute: SmscSystemId
- Operation: addApplicationSpecificSettings
- Operation: closeClientConnection
- Operation: closeServerConnection
- Operation: closeServerPort
- Operation: deleteApplicationSpecificSettings

- Operation: listApplicationSpecificSettings
- Operation: listClientConnections
- Operation: listClusterServerConnectionsForMOJumping
- Operation: listPluginInstances
- Operation: listServerConnections
- Operation: listServerPorts
- Operation: resetClientConnection
- Operation: resetServerPort
- Operation: updateAllServerPorts

Attribute: ConnectionBasedRouting

Scope: Cluster

Unit: Not applicable

Format: Boolean

Enables and disables connection-based routing for Native SMPP plug-ins.

Connection-based routing lets operators configure geo-redundant sites to allow applications to send MO, MT, and DR traffic to and from either of the sites.

Set to `true` to enable, `false` to disable. The default is `false`.

For more information, see Connection-Based Routing.

This attribute can be modified only when there are no active connections between SMPP applications and the SMPP Server Service.

Attribute: EnquireLinkMaxFailureTimes

Scope: Cluster

Unit: Not applicable

Format: Integer

Maximum number of failed **ENQUIRE_LINK** requests to the application before the connection with the application is closed.

Attribute: EnquireLinkTimerValue

Scope: Cluster

Unit: Seconds

Format: Integer

Minimum interval between the submission of **ENQUIRE_LINK** requests (heartbeats) to an application.

To disable the sending of **ENQUIRE_LINK** requests, set this value to **0** (zero).

Attribute: InactivityTimerValue

Scope: Cluster

Unit: Seconds

Format: Integer

Maximum period of inactivity for an application before the connection with the application is closed.

Use 0 (zero) for no timeout.

Attribute: InitiationTimerValue

Scope: Cluster

Unit: Seconds

Format: Integer

Maximum time between establishment a connection to the application and the **BIND** request.

If the timeout value is reached, the server connection is closed.

Use **0** (zero) for no timeout.

Attribute: LooseBinding

Scope: Server

Unit: Not applicable

Format: Boolean

Controls behavior on application **BIND** and **UNBIND**.

If `true`, the following applies:

- As long as there are transmitting-capable connections (TX, TRX) from applications, TX and TRX connections will be kept open to SMSCs.
- As long as there are receiving-capable connections (RX, TRX) from applications, RX and TRX connections will be kept open to SMSCs.

If `false`, the binding rules are more restrictive:

- As long as there are TX connections from applications, TX connections will be kept open to SMSCs.
- As long as there are RX connections from applications, RX connections will be kept open to SMSCs.
- As long as there are TRX connections from applications, TRX connections will be kept open to SMSCs.

This attribute value cannot be changed while there is an active connection with an application.

The default is `true`.

Attribute: OfflineMO

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether the JMS-based routing functionality for network-triggered messages is enabled. If `true`, a message from the network to an NT server that does not have an active bind to the appropriate application can be placed in a JMS queue from which another server that does have an active bind can fetch it and send it to the application.

The default is `false`.

The time that the message stays alive in the JMS queue is configurable. The default value is **3600000** milliseconds. To change this value, in the administrative console:

1. Select **Services ->Messaging->JMS Modules**.
2. Click **WLNGJMSResource**. The **Settings** page opens.
3. On the **Configuration** tab, click **LegacySMSConnectionFactory**. The **Settings for LegacySMSConnectionFactory** page opens
4. On the **Configuration** tab, select the **Default Delivery** sub-tab.
5. Make your changes to the **Default Time-to-Live** attribute.
6. Click **Save**.
7. Click the **Activate Changes** button in the **Change Center**.

This attribute is not applicable if **ConnectionBasedRouting** is `true`.

Attribute: RequestTimerValue

Scope: Cluster

Unit: Seconds

Format: Integer

Maximum time between the submission of a request to an application and the receipt of the corresponding response, before the connection is closed.

Set to **0** (zero) for no timeout.

Attribute: ServerAddress

Scope: Server

Unit: Not applicable

Format: String

Default host name or IP address that applications use to connect to the SMPP Server Service.

Multiple addresses are supported as a comma-separated list of IP addresses.

Attribute: ServerPort

Scope: Server

Unit: Not applicable

Format: Integer [**1024–65535**]

Default port that applications use to connect to the SMPP Server Service.

Updating this attribute takes effect immediately if the old port and the new port are not in use. In this case, the SMPP Server Service closes the old port and opens the new port.

If the old port is in use when this attribute is set, it is closed after the last application instance that used it is removed by the **deleteApplicationSpecificSettings** operation. See "Operation: deleteApplicationSpecificSettings" for more information.

When a new application instance is added with the **addApplicationSpecificSettings** operation, if the **acceptPort** parameter to that operation is a negative value, all traffic from the application uses the new port. See "Operation: addApplicationSpecificSettings" for more information.

Attribute: SmscSystemId

Scope: Cluster

Unit: Not applicable

Format: String; maximum 16 characters

SMSC system ID. Sent to an ESME client upon a successful **BIND**.

Operation: addApplicationSpecificSettings

Scope: Cluster

Specifies connection details for an application with the specified **applicationInstanceId**.

Required for an application instance to access the SMPP Server Service.

This operation takes effect immediately after it is invoked. The SMPP Server Service closes the old port, if it is not in use, and opens the new port, if it is not in use.

See "Windowing" for more information about the **windowingSize**, **windowingMaxQueueSize**, and **windowingMaxWaitTime** parameters.

Signature:

```
addApplicationSpecificSettings(applicationInstanceId: int, acceptPort: int,
maxSession: int, subsequentOperationsAllowed: boolean, notificationEnabled:
boolean, addressRange: String, windowingSize: int, windowingMaxQueueSize: int,
windowingMaxWaitTime: int)
```

Table 20–6 addApplicationSpecificSettings Parameters

Parameter	Description
applicationInstanceId	ID of the application instance for which the settings are valid.
acceptPort	Port to which the application is allowed to bind. A negative value allows binding to the port specified as the ServerPort. See "Attribute: ServerPort" for more information.
maxSession	Maximum number of concurrent sessions the application is allowed to establish. A negative value allows an unlimited number of concurrent sessions.

Table 20–6 (Cont.) addApplicationSpecificSettings Parameters

Parameter	Description
subsequentOperationsAllowed	Specifies if the application is allowed to perform the following operations on a previously-sent short message: ■ **QUERY_SM** ■ **REPLACE_SM** ■ **CANCEL_SM** Enter: ■ `true` to allow ■ `false` to deny Setting this attribute to `false` reduces the resource utilization by the SMPP Server Service since it does not need to track each request in its store. See "Connection-Based Routing" for details about how subsequent operations are handled in geo-redundant configurations.
notificationEnabled	Specifies if the application is allowed to receive network-triggered messages. If allowed, the application can send the **BIND_TRANSCEIVER** and **BIND_RECEIVER** operations. Enter: ■ `true` to allow ■ `false` to deny
addressRange	If the **notificationEnabled** parameter is `true`, specifies the address range for listening for network-triggered short messages. Only messages that are sent to this address range are forwarded to the application. The address range is expressed as a regular expression. When used for binding a receiver or transceiver, the address range in the bind operation must be in the specified range. Otherwise the bind is rejected. See Appendix A in *SMPP Protocol Specification v3.4*. This setting is valid only if the application is allowed to receive network-triggered messages. Example: **^1234**
windowingSize	Maximum number of concurrent mobile-originated requests.
windowingMaxQueueSize	Maximum number of mobile-originated requests allowed to wait in the windowing queue.
windowingMaxWaitTime	Maximum time in seconds that each mobile-originated request is allowed to wait in the windowing queue.

Operation: closeClientConnection

Scope: Server

Closes the specified client connection between the communication service and the SMSC.

If the **connectionId** parameter is matched, closes just the client connection.

If the **pluginInstanceId** parameter is matched, closes all connections related to the plug-in instance.

Signature:

```
closeClientConnection(connectionId: String, pluginInstanceId: String)
```

Table 20–7 closeClientConnection Parameters

Parameter	Description
connectionId	Id of connection to be closed. Created by a previous **BIND**.
pluginInstanceId	Id of plug-in instance for which related connections are to be closed.

Operation: closeServerConnection

Scope: Server

Closes the specified server connections between the communication service and the application.

If the **connectionId** parameter is matched, closes the connection.

If the **appInstanceId** parameter is matched, closes all connections related to the application instance.

If the **port** parameter is matched, closes all connections to that port.

Signature:

```
closeServerConnection(connectionId: string, appInstanceId: string, port: int)
```

Table 20–8 closeServerConnection Parameters

Parameter	Description
connectionId	Id of connection to be closed. Created by a previous **BIND** operation.
appInstanceId	Id of application instance for which related connections are to be closed.
port	Port for which connections are to be closed.

Operation: closeServerPort

Scope: Server

Closes the specified server port on which the SMPP Server Service is listening. Closes all server and client connections on the specified port.

Signature:

```
closeServerPort(port: int)
```

Table 20–9 closeServerPort Parameters

Parameter	Description
port	Port to close.

Operation: deleteApplicationSpecificSettings

Scope: Cluster

Deletes application-specific settings for an application with the specified **applicationInstanceId**.

The application will no longer be able to access the SMPP Server Service.

Signature:

```
deleteApplicationSpecificSettings(applicationInstanceId: String)
```

Table 20–10 deleteApplicationSpecificSettings Parameters

Parameter	Description
applicationInstanceId	ID of the application instance for which to delete settings

Operation: listApplicationSpecificSettings

Scope: Cluster

Displays all application-specific settings.

Signature:

```
listApplicationSpecificSettings()
```

Operation: listClientConnections

Scope: Server

Displays description and status of all client connections. These are connections between the communication service and the SMSC.

Signature:

```
listClientConnections()
```

Operation: listClusterServerConnectionsForMOJumping

Scope: Cluster

Displays the description and status for cluster server connections for which the MO jumping is enabled.

For information about MO jumping, see "Attribute: OfflineMO" for more information.

Signature:

```
listClusterServerConnectionsForMOJumping()
```

Operation: listPluginInstances

Scope: Server

Displays description and status for all registered plug-in instances.

```
listPluginInstances()
```

Operation: listServerConnections

Scope: Server

Displays description and status of each server connection.

Signature:

```
listServerConnections()
```

Operation: listServerPorts

Scope: Server

Displays description and status of each server port.

Signature:

```
listServerPorts()
```

Operation: resetClientConnection

Scope: Server

Closes and restarts the specified client connection between the communication service and the SMSC.

If the **connectionId** parameter is matched, resets the connection.

If the **pluginInstanceId** parameter is matched, resets all connections related to the plug-in.

Signature:

```
resetClientConnection(connectionId: String, pluginInstanceId: String)
```

Table 20–11 resetClientConnection Parameters

Parameter	Description
connectionId	Id of the connection to be reset. Created by a previous **BIND** operation.
pluginInstanceId	Id of the plug-in instance for which related connections are to be reset.

Operation: resetServerPort

Scope: Server

Closes and restarts the specified application-facing server port on which the SMPP Server Service is listening. This operation resets all server and client connections on the specified port.

Signature:

```
resetServerPort(port: int)
```

Table 20–12 resetServerPort Parameters

Parameter	Description
port	Port to reset.

Operation: updateAllServerPorts

Scope: Server

Closes and restarts all server ports all local server ports in the current configuration.

Signature:

```
updateAllServerPorts()
```

Reference: Attributes and Operations for Native SMPP Plug-in

This section describes the attributes and operations for configuration and maintenance:

- Attribute: BindType

- Attribute: DeliverSmRespCommandStatus
- Attribute: EnableDeleteAfterCancel
- Attribute: EnableDeleteAfterNotify
- Attribute: EnableDeleteAfterQuery
- Attribute: EnquireLinkTimerValue
- Attribute: EsmeAddressRange
- Attribute: EsmeNpi
- Attribute: EsmePassword
- Attribute: EsmeSystemId
- Attribute: EsmeSystemType
- Attribute: EsmeTon
- Attribute: LocalAddress
- Attribute: LocalPort
- Attribute: MessageIdInHexFormat
- Attribute: NumberReceiverConnections
- Attribute: NumberTransceiverConnections
- Attribute: NumberTransmitterConnections
- Attribute: RequestTimerValue
- Attribute: RetryTimesBeforeGiveUp
- Attribute: RetryTimesBeforeReconnect
- Attribute: SmscAddress
- Attribute: SmppVersion
- Attribute: SmscPort
- Attribute: WindowingMaxQueueSize
- Attribute: WindowingMaxWaitTime
- Attribute: WindowingSize

Attribute: BindType

Scope: Server

Unit: Not applicable

Format: Integer

Specifies how the plug-in binds to the SMSC.

Use:

- **1** to bind as Transceiver
- **2** to bind as Transmitter
- **3** to bind as Receiver

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: DeliverSmRespCommandStatus

Scope: Cluster

Unit: Not applicable

Format: Integer

Error code to used in the **command_status** field when the application is unavailable. See section *5.1.3 command_status* in *SMPP Protocol Specification v3.4*.

Specifies how the plug-in responds to an SMSC if a network-triggered short message cannot be delivered to an application that subscribed for notifications on incoming short messages.

The default is **ESME_RINVDSTADR**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: EnableDeleteAfterCancel

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether to delete SMPP session information from storage after receipt of a **CANCEL_SM_RESP.**

The default is `true`.

Attribute: EnableDeleteAfterNotify

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether to delete SMPP session information from storage after receipt of the delivery report with the final message state.

The default is `true`.

Attribute: EnableDeleteAfterQuery

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether to delete SMPP session information from storage after the receipt of the query response with the final message state.

The default is `false`.

Attribute: EnquireLinkTimerValue

Scope: Cluster

Unit: Seconds

Format: Integer

Minimum interval between **ENQUIRE_LINK** requests to the SMSC.

The default is **60**.

To disable the sending of **ENQUIRE_LINK** requests, set this value to **0** (zero).

Attribute: EsmeAddressRange

Scope: Server

Unit: Not applicable

Format: String formatted as a regular expression.

ESME address range. This is the address range of the SMS messages to be sent to the plug-in instance by the SMSC.

The default is **^.*$**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: EsmeNpi

Scope: Server

Unit: Not applicable

Format: Integer

The ESME Numbering Plan Indicator (NPI) used in a **BIND** request.

Used for destination address and as a default for originating address. Also used for both destination address and originating address during bind operation. Use:

- **0** for Unknown
- **1** for ISDN (E163/E164)
- **3** for Data (X.121)
- **4** for Telex (F.69)
- **6** for Land Mobile (E.212)
- **8** for National
- **9** for Private
- **10** for ERMES
- **14** for Internet (IP)
- **18** for WAP Client ID

The default is **0**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: EsmePassword

Scope: Cluster

Unit: Not applicable

Format: String

Password used by the plug-in instance for connecting to the SMSC as an ESME.

Attribute: EsmeSystemId

Scope: Cluster

Unit: Not applicable

Format: String

System ID used by the plug-in instance when connecting to the SMSC as an ESME.

The default is **OCSG**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: EsmeSystemType

Scope: Cluster

Unit: Not applicable

Format: String

System type used by the plug-in instance for connecting to the SMSC as an ESME.

The default is **mess_gateway**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: EsmeTon

Scope: Cluster

Unit: Not applicable

Format: Integer

ESME Type Of Number (TON).

Used for destination address and as a default for originating address. Also used for both destination address and originating address in a **BIND** request. Use:

- **0** for Unknown
- **1** for International
- **2** for National
- **3** for Network
- **4** for Subscriber
- **5** for Alphanumeric
- **6** for Abbreviated
- **7** Reserved

The default is 0.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: LocalAddress

Scope: Server

Unit: Not applicable

Format: String

Local server address used by the plug-in to connect to the SMSC. The address can be expressed as an IP address or host name. The address or host name must resolve to a local address.

Enter **""** to use the default address of the server.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: LocalPort

Scope: Server

Unit: Not applicable

Format: Integer [**1** - (**65535** - *number of connections*)]

Local port used by the plug-in to connect to the SMSC.

The default is **3000**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: MessageIdInHexFormat

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies the message_id format used in **SUBMIT_SM_RESP**, **SUBMIT_MULTI_RESP**, **DATA_SM_RESP** operations.

If `true`, the format is hexadecimal; if `false`, it is decimal.

The default is `false`.

Attribute: NumberReceiverConnections

Scope: Cluster

Unit: Not applicable

Format: Integer

Number of connections used to connect to the SMSC if the bind type is 3.

The default is **1**.

See "Attribute: BindType" for more information.

The connections are established with the first successful **BIND** between the application and Services Gatekeeper, if connection-based routing is disabled.

If connection-based routing is enabled, connections to the SMSC cannot be shared among different application instances, so this attribute is ignored.

Attribute: NumberTransceiverConnections

Scope: Cluster

Unit: Not applicable

Format: Integer

Number of connections used to connect to the SMSC if the bind type is 1.

The default is **1**.

See "Attribute: BindType" for more information.

The connections are established with the first successful **BIND** between the application and Services Gatekeeper, if connection-based routing is disabled.

If connection-based routing is enabled, connections to the SMSC cannot be shared among different application instances, so this attribute is ignored.

Attribute: NumberTransmitterConnections

Scope: Cluster

Unit: Not applicable

Format: Integer

Number of connections used to connect to the SMSC if the bind type is or 2.

The default is **1**.

See "Attribute: BindType" for more information.

The connections are established with the first successful **BIND** between the application and Services Gatekeeper, if connection-based routing is disabled.

If connection-based routing is enabled, connections to the SMSC cannot be shared among different application instances, so this attribute is ignored.

Attribute: RequestTimerValue

Scope: Cluster

Unit: Seconds

Format: Integer

Maximum time between the submission of a request to the SMSC and the receipt of the corresponding response before the connection is terminated.

The default is **20**.

Set to **0** (zero) for no timeout.

Attribute: RetryTimesBeforeGiveUp

Scope: Cluster

Unit: Not applicable

Format: Integer

Maximum number of times for the plug-in to try to reconnect to the server service.

The default is **30**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: RetryTimesBeforeReconnect

Scope: Cluster

Unit: Not applicable

Format: Integer

Maximum number of times for the plug-in to try to connect to the server service before attempting to reconnect.

The default is **3**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: SmscAddress

Scope: Cluster

Unit: Not applicable

Format: String

SMSC address as an IP address or host name.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: SmppVersion

Scope: Cluster

Unit: Not applicable

Format: String

SMPP version of the communication service used between Services Gatekeeper and the SMSC.

Valid values are **3.4** and **5.0**.

3.4 is the fully-supported version.

5.0 is provided to support the **billing identification** parameter and the **ussd_service_operation** parameter for the **DELIVER_SM** operation. See "USSD Support" and "Billing Identification" for information about these parameters

The default is 3.4.

Attribute: SmscPort

Scope: Cluster

Unit: Not applicable

Format: Integer

Listening port used by the SMSC

The default is **5016**.

The setting is not applied until the plug-in is restarted or the SMPP Server Service "Operation: resetClientConnection" is performed.

Attribute: WindowingMaxQueueSize

Scope: Cluster

Unit: Not applicable

Format: Integer

Maximum number of mobile-terminated requests to the SMSC allowed in the windowing queue.

The default is **100**.

If any one of the three windowing attributes (**WindowingMaxQueueSize**, **WindowingMaxWaitTime**, or **WindowingSize**) is set to a value less than zero, windowing is turned off. If all of these three attributes have values greater than zero, windowing is turned on.

See "Windowing" for general information about windowing.

Attribute: WindowingMaxWaitTime

Scope: Cluster

Unit: Seconds

Format: Integer

Maximum time that a mobile-terminated request to the SMSC is allowed to wait in the windowing queue.

The default is **15**.

If any one of the three windowing attributes (**WindowingMaxQueueSize**, **WindowingMaxWaitTime**, or **WindowingSize**) is set to a value less than zero, windowing is turned off. If all of these three attributes have values greater than zero, windowing is turned on.

See "Windowing" for general information about windowing.

Attribute: WindowingSize

Scope: Cluster

Unit: Not applicable

Format: Integer

Maximum number of simultaneous unacknowledged mobile-terminated requests to the SMSC enforced for each connection.

The default is **5**.

If any one of the three windowing attributes (**WindowingMaxQueueSize**, **WindowingMaxWaitTime**, or **WindowingSize**) is set to a value less than zero, windowing is turned off. If all of these three attributes have values greater than zero, windowing is turned on.

See "Windowing" for general information about windowing.

21 Native UCP

This chapter describes the Native UCP communication service.

Overview of the Native UCP Communication Service

The Native UCP communication service exposes the UCP Short Message Service Center EMI-UCP standard interfaces.

The communication service acts as a Short Message Terminal (SMT) that connects to a Short Messaging Service Center (SMSC) over TCP/IP.

For the exact version of the standards that the communication service supports for the application-facing interfaces and the network protocols, see the appendix on standards and specifications in *Concepts Guide*.

The Native UCP service can:

- Connect to a specified SMSC address.
- Open a session with the SMSC.
- Send acknowledgments to the SMSC.
- Send acknowledgments to the application.
- Send a mobile-terminated SMS message to destination addresses.
- Deliver a mobile-originated SMS message.
- Deliver a delivery notification associated with a previously sent mobile-terminated SMS message.

All Native UCP components are deployed in the network tier.

The core module of the Native UCP communication service is a Native UCP Protocol Server Service deployed as an Oracle WebLogic Server Service. The Native UCP Protocol Server Service:

- Receives UCP data from the socket.
- Constructs the UCP protocol data unit (PDU).
- Associates the current PDU with the correct application instance.
- Invokes the plug-in.

There is also a Native UCP managed plug-in module, as well as the Native UCP plug-in instances.

In addition, Native UCP uses the Connection Information Manager service to create and manage a credential map to support each plug-in instance. See "Managing and

Configuring Connection Information" in *System Administrator's Guide* for information about the Connection Information Manager.

The entire Native UCP Service is deployed in the network tier, so applications using it must connect directly to the network tier. The network and any firewall should be configured to allow connection to the ports defined for the Native UCP Service.

There is no failover between network tier servers. Redundant SMSCs and redundant network cards are required to support high-availability features.

To optimize system utilization, the application and the SMSC should load balance the requests among all network tier servers.

Hitless upgrade is not supported for the Native UCP communication service. To upgrade you must restart the server.

Connection and Credential Handling

Plug-in instances establish connections to Services Gatekeeper using facilities provided by the Protocol Server Service. They also use the Protocol Server Service to open a session and to send requests to the SMSC. The Protocol Server Service creates a new socket connection for each session management operation of subtype "open session" that is sent.

The Native UCP Protocol Server Service uses the Connection Information Manager´s **getConnectInfo** operation to get the connection information for a particular plug-in instance. When a plug-in instance sends a Native UCP PDU to the Protocol Server Service passing its plug-in instance ID, a connection ID is returned. This connection ID identifies the SMSC connection on which the request was sent.

A server-side connection connects an application to Services Gatekeeper, which is the server in this context.

A client-side connection connects an SMSC to Services Gatekeeper, which is a client in this context.

Native UCP has no unbind operation. There are no receiver, transceiver, or transmitter connection types. If a connection is lost, the Protocol Server Service automatically closes one connection to the SMSC for the current application instance. See Multiple Connections for more information.

Credentials

The Protocol Server Service performs network credential mappings based on a credential map set up in the Connection Information Manager.

A user/password combination is associated with a credential ID that is stored in the Connection Information Manager. See the Connection Information Manager´s **createOrUpdateUserPasswordCredentialEntry** operation. The credential ID is associated with a plug-in instance and an application instance in an entry in the Connection Information Manager's credential map. See the Connection Information Manager's **createOrUpdateCredentialMap** operation.

For detailed information on how to configure the connection information and the credential map, see the "Managing and Configuring Connection Information" chapter in *System Administrator's Guide*, another document in the set.

Windowing and Transaction Numbers To maximize throughput, Native UCP supports windowing on both the application-facing and network-facing interfaces. This provides a way to specify the amount of data that can be transmitted to and from the network without receiving an acknowledgment.

On the server side, Native UCP creates a transaction number (TRN) allocation table using default values. These are used by server-side connections sending **deliver_ short_message** and **deliver_notification** requests to an application.

On the client side, Native UCP creates a transaction number allocation table using values configured in the Connection Information Manager. Configure the client-side windowing behavior by setting the parameters listed in Table 21–1 using the Connection Information Manager´s **addXParamToCredentialEntry** operation. For information about this operation, see the "Managing and Configuring Connection Information" chapter *System Administrator's Guide.*

Table 21–1 UCP Windowing Parameters in ConnectionInfoManager

Parameter	Description	Default
windowSize	Maximum number of unacknowledged transactions allowed between a plug-in instance and an SMSC	100
maxWaitAcquireTimeout	Maximum time in milliseconds that a request can wait while trying to allocate a transaction number	3000
allocationTimeout	Maximum time in milliseconds that an allocated transaction number can be held while the plug-in or the SMSC is waiting for an acknowledgment	5000
maxQueueSize	Maximum number of threads that can wait for a transaction number to be allocated	5

Behavior When the Window is Exceeded If Services Gatekeeper tries to allocate a TRN when all the TRNs have been allocated and none is old enough to be cleaned up and **maxWaitAcquireTimeout** has expired, an exception is thrown causing Services Gatekeeper to respond with a **NACK**.

Behavior When TRNs Are Not Released When a TRN is allocated, values that have already been allocated are checked to see whether they have expired. Entries older than **allocationTimeout** are cleaned and automatically released. An error is logged, but no alarm is generated. No **NACK** is triggered for a request that was originally associated with the expired TRN.

Multiple Connections

An application instance can establish multiple TCP connections to Services Gatekeeper. Multiple application instances, those with different application instance names, cannot share a connection to the SMSC.

If one connection between an application instance and an SMSC is dropped, Services Gatekeeper does not automatically close associated application instance connections as long as there are other SMSC connections available for that same application instance. If all connections to the SMSC for a particular application instance are dropped, Services Gatekeeper terminates all of that application instance's connections.

Connection Pooling

When an application instance sends an **open session** operation on a new connection, the Protocol Server Service tries to establish a connection to the underlying SMSCs and then to open the session. It does not automatically establish connection pools to the underlying SMSCs. It establishes additional connections only when an application instance establishes multiple connections with Services Gatekeeper.

Because an application may establish multiple connections, a request sent from the Protocol Server Service to a plug-in includes a server-side connection identifier. This identifier is then included when the plug-in uses the Protocol Server Service to send acknowledgments back to the application. Acknowledgments must be sent on the same connection as the corresponding request. Delivery reports can be sent on a different connection.

Windowing and Transaction Numbers

To maximize throughput, Native UCP supports windowing on both the application-facing and network-facing interfaces. This provides a way to specify the amount of data that can be transmitted to and from the network without receiving an acknowledgment.

On the server side, Native UCP creates a transaction number (TRN) allocation table using default values. These are used by server-side connections sending **deliver_ short_message** and **deliver_notification** requests to an application.

On the client side, Native UCP creates a transaction number allocation table using values configured in the Connection Information Manager. Configure the client-side windowing behavior by setting the parameters listed in Table 21–2 using the Connection Information Manager´s **addXParamToCredentialEntry** operation. For information about this operation, see the "Managing and Configuring Connection Information" chapter in *System Administrator's Guide*.

Table 21–2 UCP Windowing Parameters in ConnectionInfoManager

Parameter	Description	Default
windowSize	Maximum number of unacknowledged transactions allowed between a plug-in instance and an SMSC	100
maxWaitAcquireTimeout	Maximum time in milliseconds that a request can wait while trying to allocate a transaction number	3000
allocationTimeout	Maximum time in milliseconds that an allocated transaction number can be held while the plug-in or the SMSC is waiting for an acknowledgment	5000
maxQueueSize	Maximum number of threads that can wait for a transaction number to be allocated	5

Behavior When the Window is Exceeded

If Services Gatekeeper tries to allocate a TRN when all the TRNs have been allocated and none is old enough to be cleaned up and **maxWaitAcquireTimeout** has expired, an exception is thrown causing Services Gatekeeper to respond with a **NACK**.

Behavior When TRNs Are Not Released

When a TRN is allocated, values that have already been allocated are checked to see whether they have expired. Entries older than **allocationTimeout** are cleaned and automatically released. An error is logged, but no alarm is generated. No **NACK** is triggered for a request that was originally associated with the expired TRN.

Authentication

Applications are authenticated on receipt of the **openSession** PDU, after which the connection is associated with the authenticated identity.

Subsequent requests on the connection trigger an identity assertion associating the request with the identity that was authenticated with the receipt of the **open session** PDU. A consequence of this behavior is that an application can continue to send messages after the password has been changed. To force a new authentication, close the connection.

Table 21–3 describes the mapping between the Native UCP authentication parameters and the Services Gatekeeper parameters.

Table 21–3 Authentication Parameters

Native UCP Parameter	Services Gatekeeper ConnectionInfoManager Credential Parameter
originating address (OAdC)	application instance name
password	password

The password is stored in the Connection Information Manager. No password information is stored by the Protocol Server Service or by the plug-in.

Availability and Retry

The availability and retry behavior of the Protocol Server Service is as follows.

Application-Initiated traffic

If there is no acknowledgment from the network, the UCP Protocol Server Service does not start any timers per request, does not perform any retries, and does not report an acknowledgment back to the application.

The only exception to this behavior is when the wait on an openSession request exceeds the configured Native UCP **OpenSessionTimeout** maximum. See "Attribute: OpenSessionTimeout" for more information.

If the Protocol Server Service receives an exception when calling the **submit_short_ message** operation, it sends a **NACK** to the application.

Network-Initiated traffic

If the underlying SMSC does not receive an acknowledgment from Services Gatekeeper, the SMSC should resend the request.

If a message is sent to an application but no acknowledgment is returned from the application, the UCP Protocol Server Service does not start any per request timers, does not perform any retries, and does not send back an acknowledgment to the application.

If the Protocol Server Service receives an exception when calling the **deliver_short_ message** or **deliver_notification** operation, it sends a **NACK** to the SMSC.

Delivery reports do not have to be sent on the same server that sent the original delivery request, even in a geo-redundant setup. In a clustered configuration, if the server that submits an SMS fails, another server can handle the delivery report for that SMS.

Client-Side Retry Handling

Native UCP automatically tries to re-establish a dropped connection when an initial openSession attempt or an established session fails. The number of retries attempted is configured by the **maxReconnectAttempts** attribute and the number of milliseconds between retries by the **timeBetweenReconnectAttempts** attribute.

The retry behavior is as follows:

- **Initial openSession failure:** When an application sends the initial **open session** request, Services Gatekeeper sends one **open session** request to each SMSC that matches the current configuration in terms of plug-ins, routes, SLAs, and so on. If one of the SMSCs responds with an **ACK**, Services Gatekeeper returns an **ACK** to the application.

  For all SMSCs that Services Gatekeeper cannot connect to or receive an acknowledgment from, it tries to re-establish a connection. Specifically, retry is triggered in the following cases:

  – Services Gatekeeper receives a **NACK** in response to the **open session** request.

  – The socket to the SMSC cannot be set up.

  – The socket is closed before the acknowledgment is received.

  – The timeout period, configured by the **OpenSessionTimeout** attribute, expires before an acknowledgment is received.

- **Established session failure:** If the client-side connection is dropped and the Services Gatekeeper application instance associated with the dropped connection still has other working connections, Services Gatekeeper sends an **open session** request to each SMSC that matches the current configuration, following the same procedure as described above for an initial openSession failure.

Use the **dumpOngoingClientConnectionsRetryInfo** and **stopOngoingClientConnectionRetry** operations to manage connections that are in the retry state.

Heartbeat Support

Native UCP provides heartbeat support by sending UCP operation "31" (SMT alert) requests at regular intervals. This prevents firewalls and the SMSC from closing an idle connection.

Server-Side Heartbeat Support

Heartbeat support for Native UCP server-side connections has the following characteristics:

- In response to a UCP operation type "31"(SMT Alert), the corresponding acknowledgment is sent.
- There are no timeouts associated with heartbeats.
- Services Gatekeeper does not close any connections because of missing heartbeat requests.
- Heartbeats received on server-side connections are not forwarded to client-side connections.
- There are no configuration attributes associated with server-side heartbeat functions.

Client-Side Heartbeat Support

Heartbeat support for Native UCP client-side connections has the following characteristics:

- Heartbeats are not enabled by default for a client-side connection.
- Heartbeats are enabled by setting the **heartbeatInterval** parameter in the Connection Information Manager. This parameter defines the interval, in

milliseconds, between UCP operation type 31 requests. The value is configured with the **addXParamToCredentialEntry** operation. For information about this operation, see "Managing and Configuring Connection Information" in *System Administrator's Guide*.

- Services Gatekeeper does not close any client-side connections because of missing acknowledgments on heartbeat requests.
- Received client-side heartbeat acknowledgments are not forwarded to server-side connections.

Storage Provider

The Native UCP Protocol Server Service and the Native UCP plug-in use the default Services Gatekeeper store.

Application Interfaces

For information about the application interface for the Native UCP communication service, see the discussion of Native Interfaces in *Application Developer's Guide*.

Events and Statistics

The Native UCP communication service generates Event Data Records (EDRs), Charging Data Records (CDRs), alarms, and statistics to assist system administrators and developers in monitoring the service

For general information, see Appendix A, "Events, Alarms, and Charging."

Event Data Records

Table 21–4 lists IDs of the EDRs created by the Native UCP communication service.

When there are multiple SMSCs, it is possible that EDR data generated for **open session** requests may contain the wrong data for some of the EDRs because EDR data is stored in the current context.

Table 21–4 Event Types Generated by Native UCP

EDR ID	Description
402001	application-initiated openSession to the SMSC
402002	application-initiated submitSM to the SMSC
402003	application-initiated ACK to the SMSC
402004	application-initiated NACK to the SMSC
402010	network-triggered deliverSM to the application
402011	network-triggered deliveryNotification to the application
402012	network-triggered ACK to the application
402013	network-triggered NACK to the application

Table 21–5 describes Native UCP-specific fields included in the Native UCP EDRs.

Table 21–5 Native UCP-Specific EDR Fields

EDR Parameter	Description
UCP_isOperation	`true` if EDR is for an operation, `false` if for an acknowledgment
UCP_opType	UCP operation type; for example, 51 for a submit_short_message request
UCP_trn	UCP transaction number; see "About UCP_trn/UCP_mappedTrn"
UCP_mappedTrn	mapped transaction number; see "About UCP_trn/UCP_mappedTrn"
UCP_sourceConnID	source connection ID; for the connection that received the PDU
UCP_outgoingConnID	outgoing connection ID for the connection on which the PDU was sent
UCP_adc	AdC parameter in the UCP PDU
UCP_oadc	OAdC parameter in the UCP PDU; see "About UCP_oadc"
UCP_scts	Service Center Timestamp (SCTS) parameter in the UCP PDU

About UCP_trn/UCP_mappedTrn

Transaction numbers must be mapped in the following cases:

- When sending a **submit_short_message** operation (51) to the SMSC and receiving the corresponding acknowledgment
- When sending a **deliver_notification** operation (53) to the application and receiving the corresponding acknowledgment
- When sending a **deliver_short_message** operation (52) to the application and receiving the corresponding acknowledgment

A **UCP_mappedTrn** is required in the following circumstances because pooled connections create the possibility of sending conflicting or overlapping transaction numbers were they not mapped:

- When a **submit_short_message** request is sent:
  - **UCP_trn** holds the original transaction number as received by the application.
  - **UCP_mappedTrn** holds the transaction number that was used in the request to the SMSC.
- When the acknowledgment for the **submit_short_message** request is received:
  - **UCP_trn** holds the original transaction number, which is then used to forward the acknowledgment to the application.
  - **UCP_mappedTrn** holds the transaction number that was included in the acknowledgment from the SMSC.

About UCP_oadc

The **UCP_oadc** parameter used in a **deliver_notification** operation (53) identifies the recipient of a message that was previously sent by the **submit_short_message** operation (51).

The **UCP_oadc** parameter normally contains the large account/originator number.

Charging Data Records

The Native UCP communication service generates charging data records (CDRs) under the following conditions:

- After a mobile-originated **SMS UCP** PDU has been processed by the plug-in.
- After a mobile-terminated **SMS UCP** PDU has been processed by the plug-in.
- When an **ACK** is received.

  The CDR includes the service center timestamp (SCTS). The **ACK** is correlated with **submit_short_message** using the connection identifiers and the transaction number (TRN).
- When a delivery report for a mobile-terminated SMS message is received.

  The CDR includes the SCTS for correlation with submit (and submit **ACK**) using SCTS and **AdC** and **OAdC** parameters.

Statistics

Table 21–6 maps methods invoked from either the application or the network to the transaction types collected by the Services Gatekeeper statistics counter.

Table 21–6 Methods and Transaction Types for Native UCP

Method	Transaction Type
submit_short_message	TRANSACTION_TYPE_MESSAGING_SEND
deliver_short_message	TRANSACTION_TYPE_MESSAGING_RECEIVE

Alarms

For the list of alarms, see *Alarm Handling Guide*.

Managing Native UCP

This section describes the properties and workflow for the Native UCP communication service.

Native UCP relies upon facilities in Services Gatekeeper Connection Information Manager to create and store connection and credential information for a UCP plug-in instance. See "Managing and Configuring Connection Information" in *System Administrator's Guide*.

Plug-in instances are associated with application instances and authentication credentials through the **createOrUpdateCredentialMap** operation in the Connection Information Manager.

There can be only one application instance per large account.

The work manager is registered when the managed plug-in is started. Settings in the Native UCP managed plug-in provide the IP address and port where the plug-in registers its work manager. These settings apply to all the Native UCP plug-in instances. If these settings are changed, a restart is required.

Properties for Native UCP Protocol Server Service

Table 21–7 lists the technical specifications for the UCP protocol server service.

Table 21–7 Native UCP Protocol Server Service Properties

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Container Services > UCPService

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng
	InstanceName=UCPService
	Type=oracle.ocsg.protocol.ucp.management.UCPServerMBean

Exposes this interface to applications	EMI-UCP 5.0
Deployment artifacts	oracle.ocsg.protocol.ucp_5.0.0.0.jar, oracle.ocsg.protocol.ucp_api_5.0.0.0.jar

Properties for Native UCP Managed Plug-in

Table 21–8 lists the technical specifications for the Native UCP managed plug-in.

Table 21–8 Native UCP Managed Plug-in Properties

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > oracle.ocsg.native_ucp_sms

MBean	Domain=com.bea.wlcp.wlng
	AppName=native_ucp_sms#5.0.0
	InstanceName = oracle.ocsg.native_ucp_sms
	Type = oracle.ocsg.plugin.nativefacade.ucp.management.NativeUCPManagedPluginMBean

Supported Network Interface	UCP v5.0
Supported Application Interface	UCP v5.0
Depployment artifact	wlng_nt_native_ucp_sms.ear

Properties for Native UCP Plug-in Instance

Table 21–9 lists the technical specifications for the plug-in instance.

Table 21–9 Native UCP Plug-in Instance Properties

Property	Description
Managed object in Administration Console	*domain_name* > OCSG > *server_name* > Communication Services > *plugin_instance_id*

MBean	Domain=com.bea.wlcp.wlng
	AppName=native_ucp_sms#5.0.0
	Instance Name=same as the network protocol *instance_id* assigned when the plug-in instance is created
	Type=oracle.ocsg.plugin.nativefacade.ucp.management.NativeUCPPluginMBean

Table 21–9 (Cont.) Native UCP Plug-in Instance Properties

Property	Description
Supported network interface	UCP v5.0
Supported application interface	UCP v5.0
Supported character sets	7-bit GSM charset + Unicode (16-bit UCS2)
Supported address scheme	tel
Deployment artifact	wlng_nt_native_ucp_sms.ear

Configuration Workflow for Native UCP Communication Service

Following is an outline for configuring the plug-in using the Administration Console or an MBean browser.

1. Configure the listen address and the listen port in the Native UCP managed plug-in MBean. These values are used by all the Native UCP plug-in instances.

   - Attribute: listenAddress
   - Attribute: listenPort

   See Reconfiguring Native UCP Listen Ports if you need to change these values.

2. [Optional] Configure the Native UCP address routing interceptor if there is a possibility of multiple SMSCs owning the same address. This ensures that messages sent to the same address are sent to the same SMSC. It also ensures that all message segments for a concatenated SMS message are sent to the same SMSC.

   This interceptor is not enabled by default.

   To enable it, edit the interceptor chain to include the NativeUCPAddressRouting class:

   a. Open the **config.xml** file that is bundled in the interceptors.ear file in the Services Gatekeeper installation.

      This is the file in which the interceptor chain is defined.

   b. Locate the RoundRobinPluginList routing interceptor in the file. The line for the RoundRobinPluginList interceptor looks like this:

      ```
      <interceptor class="com.bea.wlcp.wlng.interceptor.RoundRobinPluginList"
      index="1000"/>
      ```

   c. Add the NativeUCPAddressRouting class interceptor immediately before the RoundRobinPluginList routing interceptor. The line for the NativeUCPAddressRouting interceptor looks like this:

      ```
      <interceptor class="com.bea.wlcp.wlng.interceptor.NativeUCPAddressRouting"
      index="950"/>
      ```

      The resulting interceptor chain should look like this:

      ```
       ...
        <interceptor
      class="com.bea.wlcp.wlng.interceptor.FilterPluginListUsingCustomMatch"
      index="800"/>
        <interceptor class="com.bea.wlcp.wlng.interceptor.RemoveOptional"
      index="900"/>
        <interceptor
      class="com.bea.wlcp.wlng.interceptor.NativeUCPAddressRouting" index="950"/>
      ```

```
  <interceptor class="com.bea.wlcp.wlng.interceptor.RoundRobinPluginList"
index="1000"/>
  <interceptor
class="com.bea.wlcp.wlng.interceptor.InvokeServiceCorrelation"
index="1100"/>
  ...
```

The interceptor verifies that the request it is intercepting is a Native UCP request before it modifies the plug-in list. If it is not a Native UCP request, the list is not modified.

Provisioning Workflow for Native UCP Communication Service

Perform steps 1 through 5 in the Connection Information Manager. The operations and attributes used in these steps are described in "Managing and Configuring Connection Information" in *System Administrator's Guide*.

1. Create one or more Native UCP plug-in instances. See "Managing and Configuring the Plug-in Manager" in *System Administrator's Guide*. Use the plug-in service ID described in the "Properties for Native UCP Plug-in Instance" section.
2. Set up the network connection mapping for the plug-in instance using the Connection Information Manager.
   - **createOrUpdateLocalHostAddress**
   - **createOrUpdateRemoteHostAddress**
   - **createOrUpdateListenAddress**
3. Set up the network credential mapping. This associates a user and password with a credential ID. Use the following operation:
   - **createOrUpdateUserPasswordCredentialEntry**
4. Create or update the credential map for the plug-in instance. This entry associates the credential ID with the application instance ID and the plug-in instance ID. Use the following operation:
   - **createOrUpdateCredentialMap**
5. Add any connection-specific parameters needed to support windowing and heartbeats. See "Windowing and Transaction Numbers" and "Heartbeat Support" for more information. Use the following operation:
   - **addXParamToCredentialEntry**
6. Configure the retry behavior for the Native UCP Protocol Server Service.
   - Attribute: MaxReconnectAttempts
   - Attribute: TimeBetweenReconnectAttempts
7. Configure the timeout limit for the plug-in instance.
   - Attribute: OpenSessionTimeout
8. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Accounts and SLAs Guide*.
9. Provision the service provider accounts and application accounts. For information, see *Accounts and SLAs Guide*.

Reconfiguring Native UCP Listen Ports

The listen port and address is used by the Native UCP plug-in upon startup to register a work manager in the UCP Protocol Server Service.

To reconfigure the listen port and listen address:

1. In the Connection Information Manager, create the new listen address and port using the **createOrUpdateListenAddress** operation.
2. Remove the old listen address and port using the Connection Information Manager's **removeListenAddress** operation.
3. View the new listen address configuration using the Connection Information Manager's **getAllListenAddress operation**.
4. In the Native UCP Managed Plug-in, change the **listenAddress** and **listenPort** attributes to match the new values that you just configured in the Connection Information Manager. See "Attribute: listenAddress" and "Attribute: listenPort" for more information.
5. Register the work manager at the new port using the **reRegisterWorkManager** operation. See "Operation:reRegisterWorkManager" for more information.

   Services Gatekeeper cannot accept new server-side connections on the new ports until you restart the ports using the **restartPorts** operation. See "Operation: restartPorts" for more information.
6. Using "Operation: listUCPServersString", in the Native UCP Protocol Server Service, view the currently running listen ports.
7. Using "Operation: restartPorts", close and restart all current listening ports. This closes all server-side and client-side connections.
8. Using "Operation: listUCPServersString", verify that Native UCP is listening on the new ports.
9. Using "Operation: dumpServerSideConnectionsInfo", verify that applications are reconnecting on the new listen ports.
10. Using "Operation: dumpClientSideConnectionsInfo", verify that connections to the SMSCs have been re-established.

Reference: Attributes and Operations for Native UCP Protocol Server Service

This section describes the attributes and operations for configuration and maintenance:

- Attribute: MaxReconnectAttempts
- Attribute: TimeBetweenReconnectAttempts
- Attribute: UCPProtocol (read-only)
- Operation: closeClientSideConnection
- Operation: closeServerSideConnection
- Operation: dumpClientSideConnectionsInfo
- Operation: dumpOngoingClientConnectionsRetryInfo
- Operation: dumpServerSideConnectionsInfo

- Operation: listUCPServersString
- Operation: restartPorts
- Operation: stopOngoingClientConnectionRetry

Attribute: MaxReconnectAttempts

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the maximum number of reconnect retries permitted.

Table 21–10 MaxReconnectAttempts Values

Value	Meaning
-1	retry forever; no maximum
0	no retries
< 0	maximum number of retries
-1	retry forever; no maximum

See "Attribute: TimeBetweenReconnectAttempts" and "Client-Side Retry Handling" for information about how dropped connections are handled.

Attribute: TimeBetweenReconnectAttempts

Scope: Cluster

Unit: Milliseconds

Format: Integer

Specifies the time, in milliseconds, between reconnect attempts.

See "Attribute: TimeBetweenReconnectAttempts" and "Client-Side Retry Handling" for information about how dropped connections are handled.

Attribute: UCPProtocol (read-only)

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the UCP protocol string.

This value must match a protocol string defined for a listen address in the Connection Information Manager.

Operation: closeClientSideConnection

Scope: Server

Closes a client-side connection between Services Gatekeeper (the client in this relationship) and an SMSC.

After this method is used to close a client-side connection, no retries are attempted on the closed connection.

Does not implicitly close any server-side connections.

Use "Operation: dumpClientSideConnectionsInfo" to see information about the open client-side connections

Signature:

```
closeClientSideConnection(pluginInstanceID: String, ocsgUser: String,
connectionID: String)
```

Table 21–11 closeClientSideConnection Parameters

Parameter	**Description**
pluginInstanceID	Instance ID of the connected plug-in instance
ocsgUser	User name used to connect
connectionID	Connection ID the request was sent on
pluginInstanceID	Instance ID of the connected plug-in instance

Operation: closeServerSideConnection

Scope: Server

Closes a server-side connection between an application and Services Gatekeeper (the server in this relationship).

Does not implicitly close any client-side connections

Use "Operation: dumpServerSideConnectionsInfo" to see information about the open server-side connections.

Signature:

```
closeServerSideConnection(pluginInstanceID: String, ocsgUser: String,
connectionID: String)
```

Table 21–12 closeServerSideConnection Parameters

Parameter	**Description**
pluginInstanceID	Instance ID of the connected plug-in instance
ocsgUser	User name used to connect
connectionID	Connection ID the request was sent on
pluginInstanceID	Instance ID of the connected plug-in instance

Operation: dumpClientSideConnectionsInfo

Scope: Server

Lists information for all the current client-side connections. These are the connections between Services Gatekeeper and SMSCs.

Dumped information includes pluginInstanceID, ocsgUser, and connectionID for all current connections.

Signature:

```
dumpClientSideConnectionInfo()
```

Operation: dumpOngoingClientConnectionsRetryInfo

Scope: Server

Lists current client-side connections that are experiencing periodic retry attempts.

The following sample output shows a dump for a connection that has already performed seven retries and is configured to perform an infinite number of retries (Attribute: MaxReconnectAttempts = -1), with 60 seconds between retry attempts (Attribute: TimeBetweenReconnectAttempts= 60000):

```
<pluginInstance id="native_ucp_sms_plugin_2">
<user name="1234567">
<connection max_retries="-1" current_retries="7" retry_interval="60000" id="c_
localhost:9887_tmp_8237645"/>
</user>
</pluginInstance>
```

Signature:

```
dumpOngoingClientConnectionsRetryInfo()
```

Operation: dumpServerSideConnectionsInfo

Scope: Server

Lists information for all the current server-side connections. These are the connections between Services Gatekeeper and applications.

Dumped information includes pluginInstanceID, ocsgUser, and connectionID for all current connections.

Signature:

```
dumpServerSideConnectionInfo()
```

Operation: listUCPServersString

Scope: Server

Lists the currently running UCP servers as a comma-separated list of strings in the format *host*:*port*.

Signature:

```
listUCPServersString()
```

Operation: restartPorts

Scope: Server

Restarts the Native UCP listen ports.

This operation must be performed if users of the Native UCP Protocol Server Service have reregistered their work managers at new ports. See "Operation:reRegisterWorkManager" for more information.

The new ports must have been configured in the Connection Information Manager MBean for the UCP protocol. See the **createOrUpdateListenAddress** and **removeListenAddress** operations in "Managing and Configuring Connection Information" in *System Administrator's Guide*.

This operation abruptly terminates all ongoing traffic and closes all server-side and client-side connections.

Signature:

```
restartPorts()
```

Operation: stopOngoingClientConnectionRetry

Scope: Server

Stops ongoing retry attempts for the specified connection.

Use "Operation: dumpOngoingClientConnectionsRetryInfo" to see information about the current client-side connections that are in the retry state.

See "Client-Side Retry Handling" for more information.

Signature:

```
stopOngoingClientConnectionRetry(pluginInstanceID: String, ocsgUser: String,
connectionID: String)
```

Table 21–13 stopOngoingClientConnectionRetry Parameters

Parameter	Description
pluginInstanceID	Instance ID of the plug-in instance that is trying to reconnect
ocsgUser	User name used to connect
connectionID	Connection ID the request was sent on
pluginInstanceID	Instance ID of the plug-in instance that is trying to reconnect

Reference: Attributes and Operations for Native UCP Managed Plug-in

This section describes the attributes and operations for configuration and maintenance:

- Attribute: listenAddress
- Attribute: listenPort
- Operation:reRegisterWorkManager

Attribute: listenAddress

Scope: Cluster

Unit: Not applicable

Format: String

The listen address, along with the listen port, is used by the plug-in on startup to register a work manager in the UCP Protocol Server Service.

The default address is **localhost**.

The listen port/listen address must match a port address combination previously configured for the UCP protocol in the Connection Information Manager. See "Managing and Configuring Connection Information" in *System Administrator's Guide*.

Attribute: listenPort

Scope: Cluster

Unit: Not applicable

Format: Integer

The listen port, along with the listen address, is used by the plug-in on startup to register a work manager in the UCP Protocol Server Service.

The default port is **5075**.

The listen port/listen address must match a port address combination previously configured for the UCP protocol in the Connection Information Manager. See "Managing and Configuring Connection Information" in *System Administrator's Guide*.

Operation:reRegisterWorkManager

Scope: Server

Reregisters the work manager in the Native UCP Protocol Server Service.

The work manager is automatically registered during activation of the plug-in. If the listen port and address values are changed, the work manager must be reregistered with the UCP Protocol Server Service.

If you reregister the work manager, you must also restart the ports. See "Operation: restartPorts" for more information.

Signature:

```
reRegisterWorkManager()
```

Reference: Attributes and Operations for Native UCP Plug-in Instance

Following is a list of attributes for configuration and maintenance:

- Attribute: OpenSessionTimeout

Attribute: OpenSessionTimeout

Scope: Cluster

Unit: Milliseconds

Format: Integer

Maximum time to wait for an acknowledgment on an **open session** request.

If the configured timeout expires, the plug-in returns a **NACK** response.

The default is **5000**.

A Events, Alarms, and Charging

This appendix describes the features common to the handling of events, alarms, and charging in Oracle Communications Services Gatekeeper.

Events

Events are handled differently in the access tier and the network tier.

Event handling in the Access Tier

The access tier runs in the WebLogic Server's Web Services Container, so events or alarms that are raised there can be monitored through standard JMX mechanisms or by using the WebLogic Diagnostics Framework.

For more information on how this works, see "Designing Manageable Applications" in *Oracle Fusion Middleware Developing Manageable Applications With JMX for Oracle WebLogic Server* at:

`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13729/designapp.htm`

and *Oracle Fusion Middleware Configuring and Using the Diagnostics Framework for Oracle WebLogic Server* at:

`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13714/toc.htm`

Event handling in the Network Tier

In the network tier, much of the functionality comes from the interaction between communication services and the Services Gatekeeper Container Services. To capture this specialized level of information, and other pertinent information about the status of the tier, Services Gatekeeper has developed specific mechanisms to record the data.

In standard communication services, all status information generated by the network tier - events, alarms, charging data, and usage statistics - begins as an event, which is fired whenever designated methods are called or exceptions are thrown. These events are then sent to the EDR Service.

In the EDR Service, events are processed through XML-based filters, which provide the criteria by which the events are classified into types. The filters can also be used to transform the data in the original event, including adding other useful information. When the information has been processed by the filters, it is delivered to type-specific listeners. There are three types of filters that are all found in the **wlng-edr.xml** file. They produce three distinct types of data: Event Data Records (EDRs), Charging Data

Records (CDRs), and Alarms. All three of these filters can be customized as desired, using the Administrative Console. These filters can also deliver desired event-based information to external JMS-based listeners. Such listeners are set up as standard JMS topic subscribers and can be anywhere on the network. See *System Administrator's Guide* for more information on setting up these filters.

Each EDR always includes the data in Table A–1.

Table A–1 EDR Data

Element	Represents
ServiceName	The service type (SMS, Call Handling, etc.) that produced the event
ServerName	The name of the WLS host
Timestamp	The time at which the event was triggered (in milliseconds from midnight 1 January 1970)
ContainerTransactionID	The transaction ID from WebLogic Server, if available. This identifies the thread on which the request is executed
Class	The name of the class that logged the event
Method	The name of the method that logged the event
Source	The kind of event. There are two possible values for this field: ■ Method: the event was fired in relation to a method call ■ Exception: the event was fired in relation to an exception being thrown

In addition, most events includethe data in Table A–2.

Table A–2 Event Data

Element	Represents
Direction	The direction in which the request is traveling. There are two possible values for this field: ■ **South**: traveling toward the network node ■ **North:** traveling toward the application
Position	The position of the EDR relative to the method that logged the EDR. There are two possible values for this field: ■ **Before**: the event occurred before the method ■ **After**: the event occurred after the method
Interface	The interface at which the EDR is logged. There are three possible values for this field: ■ **North**: the event was logged at the north plug-in interface ■ **South**: the event was logged at the south plug-in interface ■ **Other**: the event was logged someplace other than the north or south interfaces
Exception	The name of the exception that triggered the EDR

Table A–2 (Cont.) Event Data

Element	Represents
SessionId	The application's session identifier
ServiceProviderId	The service provider account identifier
ApplicationId	The application account identifier
AppInstanceGroupId	The authentication user name of the Application Account. This is a string that is equivalent to the 2.2 value: Application Instance Group ID
OrigAddress	The originating address with scheme. For example: tel:12123334444
DestAddress	The destination address. If this is a send list, the first address will be listed here. Additional addresses are stored in the **AdditionalInfo** field.
AdditionalInfo	Variable information depending on the communication service. Stored as "key=value\n" pairs.
PluginID	The unique ID of the plug-in instance.

Alarms

Network tier alarms are those events that are of immediate interest to the operator. They are EDRs that are defined via filters created in the internal configuration file. While each alarm begins as an EDR, not all the information available in the EDR is stored when the alarm is written to the database (although that information can be retrieved using an external listener). Each alarm entry in the database includes the information described in Table A–3.

Table A–3 Alarm Data

Element	Represents
alarm_id	A unique sequential identifier
source	The name of the software module that raised the alarm and the IP address of the server in which the module runs. This is *not* the same as the Source field in the event
timestamp	The time at which the event was triggered (in milliseconds from midnight 1 January 1970)
severity	The importance of the alarm. There are four possible values for this field: ▪ 4 for warning ▪ 3 for minor ▪ 2 for major ▪ 1 for critical
identifier	The alarm type
alarm_info	Information provided by the module that raised the alarm

Table A–3 (Cont.) Alarm Data

Element	Represents
additional_info	This field includes: ▪ Service Provider ID ▪ Application ID ▪ Application Instance ID ▪ Plug-in instance ID ▪ Other information depending on context

Management integration

Services Gatekeeper supports integration of its alarm and event mechanisms with external management tools.

OSS

An Operation Support System (OSS) can integrate with Services Gatekeeper alarm and event services through the creation of external JMS listeners. Integration can be managed by OAM scripts through the use of JMX-based tools.

SNMP

Services Gatekeeper supports the sending of alarms as SNMP traps to SNMP managers. The alarms sent to the SNMP managers can be filtered on alarm severity.

Charging Data Records

CDRs originate as filtered EDRs. While each CDR begins as an EDR, not all the information available in the EDR is stored when the CDR is written to the database, although that information can be retrieved using an external listener. Each CDR entry in the database includes the information described in Table A–4.

Table A–4 CDR Data

Element	Represents
transaction_id	The Services Gatekeeper transaction sequence number
service_name	The communication service whose use is being tracked
service_provider	The Service Provider ID
application_id	The Application ID
application_instance_id	The user name of the Application Account. This is a string that is equivalent to the 2.2 value: Application Instance Group ID
container_transaction_id	The transaction ID from WebLogic Server, if available. This identifies the thread on which the request is executed
server_name	The name of the server in which the CDR was generated
timestamp	The time at which the event was triggered (in milliseconds from midnight 1 January 1970)

Table A–4 (Cont.) CDR Data

Element	Represents
service_correlation_ID	An identifier that allows the usage of multiple service types to be correlated into a single charging unit
charging_session_id	An ID correlating related transactions within a service capability module that belong to one charging session. For example, a call containing three call legs will produce three separate transactions within the same session In installations where sessions are not used, this field contains only a placeholder value.
start_of_usage	The date and time the request began to use the services of the underlying network
connect_time	The date and time the destination party responded. Used for Call Control traffic only
end_of_usage	The date and time the request stopped using the services of the underlying network
duration_of_usage	The total time the request used the services of the underlying network
amount_of_usage	The used amount. Used when charging is not time dependent, as in, for example, flat rate services
originating_party	The originating party's address
destination_party	The destination party's address. This is the first address in the case of send lists, with all additional addresses placed in the additional_info field.
charging_info	A service code added by the application or by policy service
additional_info	If the communication service supports send lists, all destination addresses other than the first, under the **destinationParty** key. In addition any other information provided by the communication service