BEA WebLogic Network Gatekeeper™

Application Development Guide

Version 3.0 ™

Document Revised: 14 September 2007

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

Document Roadmap

Creating Applications for WebLogic Network Gatekeeper

Interacting with Network Gatekeeper

Using WorkShop Controls with Network Gatekeeper

Session management Web Service

Extended Web Services WAP Push

Parlay X 2.1 Interfaces

Access Web Service (deprecated)

CHAPTER 1

Document Roadmap

This chapter describes the audience for and the organization of this document: It includes:

Document Scope and Audience

This document provides information for those developers who wish to integrate functionality provided by telecom networks into their programs by using the Web Services offered by WebLogic Network Gatekeeper. It includes a high-level overview of the process, including the login and security procedures, and a description of the interfaces and operations that are available for use.

Guide to This Document

The document contains the following chapters:

Chapter 1, “Document Roadmap”: This chapter

Chapter 2, “Creating Applications for WebLogic Network Gatekeeper”: A general introduction to the concepts involved in using Network Gatekeeper

Chapter 3, “Interacting with Network Gatekeeper”: SOAP message requirements in Network Gatekeeper

Chapter 4, “Using WorkShop Controls with Network Gatekeeper”: Using WebLogic WorkShop with Network Gatekeeper

Chapter 5, “Session management Web Service”: A detailed description of the Session Manager Web Service

Chapter 6, “Extended Web Services WAP Push”: A detailed description of the available operations used to send WAP Push messages

Chapter 7, “Parlay X 2.1 Interfaces”: A description of the Parlay X 2.1 interfaces available with details on how they are implemented in Network Gatekeeper.

Chapter 8, “Access Web Service (deprecated)”: A description of the Access Web Service

Terminology

The following terms and acronyms are used in this document:

- Account—A registered application or service provider, associated with an SLA
- Account group—Multiple registered service providers or services which share a common SLA
- Administrative User—Someone who has privileges on the Network Gatekeeper management tool. This person has an administrative user name and password
- Alarm—The result of an unexpected event in the system, often requiring corrective action
- API—Application Programming Interface
- Application—A TCP/IP based, telecom-enabled program accessed from either a telephony terminal or a computer
- Application-facing Interface—The Application Services Provider facing interface
- Application Service Provider—An organization offering application services to end users through a telephony network
- AS—Application Server
- Application User—An Application Service Provider from the perspective of internal Network Gatekeeper administration. An Application User has a user name and password

- CBC—Content Based Charging
- End User—The ultimate consumer of the services that an application provides. An end user can be the same as the network subscriber, as in the case of a prepaid service or they can be a non-subscriber, as in the case of an automated mail-ordering application where the subscriber is the mail-order company and the end user is a customer to this company
- Enterprise Operator —See Service Provider
- Event—A trackable, expected occurrence in the system, of interest to the operator
- HA —High Availability
- HTML—Hypertext Markup Language
- IP—Internet Protocol
- JDBC—Java Database Connectivity, the Java API for database access
- Location Uncertainty Shape—A geometric shape surrounding a base point specified in terms of latitude and longitude. It is used in terminal location
- MAP—Mobile Application Part
- Mated Pair—Two physically distributed installations of WebLogic Network Gatekeeper nodes sharing a subset of data allowing for high availability between the nodes
- MM7—A multimedia messaging protocol specified by 3GPP
- MPP—Mobile Positioning Protocol
- Network Plug-in—The WebLogic Network Gatekeeper module that implements the interface to a network node or OSA/Parlay SCS through a specific protocol
- NS—Network Simulator
- OAM —Operation, Administration, and Maintenance
- Operator—The party that manages the Network Gatekeeper. Usually the network operator
- OSA—Open Service Access
- PAP—Push Access Protocol
- Plug-in—See Network Plug-in

- Plug-in Manager—The Network Gatekeeper module charged with routing an application-initiated request to the appropriate network plug-in
- Policy Engine—The Network Gatekeeper module charged with evaluating whether a particular request is acceptable under the rules
- Quotas—Access rule based on an aggregated number of invocations. See also Rates
- Rates—Access rule based on allowable invocations per time period. See also Quotas
- Rules—The customizable set of criteria - based on SLAs and operator-desired additions - according to which requests are evaluated
- SCF—Service Capability Function or Service Control Function, in the OSA/Parlay sense.
- SCS—Service Capability Server, in the OSA/Parlay sense. WebLogic Network Gatekeeper can interact with these on its network-facing interface
- Service Capability—Support for a specific kind of traffic within WebLogic Network Gatekeeper. Defined in terms of traffic paths
- Service Provider—See Application Service Provider
- SIP—Session Initiation Protocol
- SLA—Service Level Agreement
- SMPP—Short Message Peer-to-Peer Protocol
- SMS—Short Message Service
- SMSC—Short Message Service Centre
- SNMP—Simple Network Management Protocol
- SOAP—Simple Object Access Protocol
- SPA—Service Provider APIs
- SS7—Signalling System 7
- Subscriber—A person or organization that signs up for access to an application. The subscriber is charged for the application service usage. See End User
- SQL—Structured Query Language
- TCP—Transmission Control Protocol

- Traffic Path—The data flow of a particular request through WebLogic Network Gatekeeper. Different Service Capabilities use different traffic paths
- USSD—Unstructured Supplementary Service Data
- VAS—Value Added Service
- VLAN—Virtual Local Area Network
- VPN—Virtual Private Network
- WebLogic Network Gatekeeper Core—The container that holds the Core Utilities
- WebLogic Network Gatekeeper Core Utilities—A set of utilities common to all traffic paths
- WSDL —Web Services Definition Language
- XML—Extended Markup Language

Related Documentation

This application development guide is a part of the WebLogic Network Gatekeeper documentation set. The other documents include:

- *Architectural Overview*
- *System Administrator's Guide*
- *Installation Guide*
- *Integration Guidelines for Partner Relationship Management*
- *Managing Service Providers and Applications*
- *Statement of Compliance*
- *Handling Alarms*
- *SDK User Guide*
- *Extension Toolkit - Developer's Guide*
- *System Backup and Restoration Guide*
- *Licensing*

- *Traffic Path Reference*

CHAPTER 2

Creating Applications for WebLogic Network Gatekeeper

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

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

- Basic Concepts
- Functional Overview
- Application Testing Workflow

Basic Concepts

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

- Traffic Paths
- Traffic Types

Traffic Paths

The basic functional unit in WebLogic Network Gatekeeper is the traffic path. A traffic path consists of a service type (Short Messaging, User Location, etc.), an application-facing interface (also called a “north” interface), and a network-facing interface (also called a “south” interface). A request for service enters through one interface, is subjected to internal processing, including evaluation for policy and protocol translation, and is then sent on using the other interface.

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

Traffic Types

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

Application-initiated Traffic

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

Network-triggered Traffic

In many cases, application-initiated traffic provides all the functionality necessary to accomplish the desired tasks. But there are certain situations in which useful information may not be

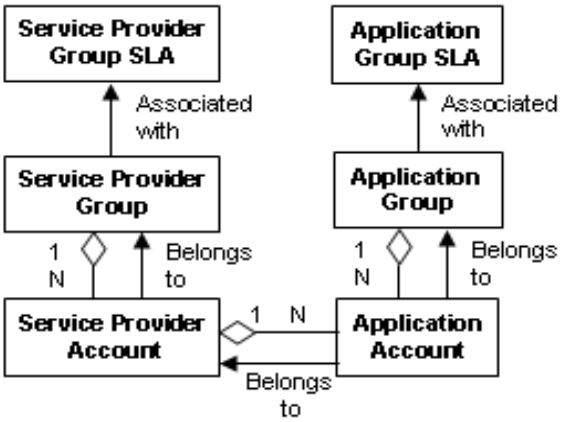
immediately available for return to the application. For example, the application might send an SMS to a mobile phone that the user has turned off. The network won't deliver the message until the user turns the phone back on, which might be hours or even days later. The application can poll to find out whether or not the message has been delivered, using `GetSmsDeliveryStatus`, which functions much like `GetCallInformation` described above. But given the possibly extended period of time involved, it would be convenient simply to have the network *notify* the application once delivery to the mobile phone has been accomplished. To do this, two things must happen:

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

Management Structures

In order to help telecom operators organize their relationships with application providers, Network Gatekeeper uses a hierarchical system of accounts. Each application is assigned a unique `username` (same as application instance group ID) and that username is tied to an Application Account. All the Application Accounts that belong to a single entity are assigned to a Service Provider Account. Application Accounts with similar requirements are put into Application Groups and Service Providers with similar requirements are put into Service Provider Groups. Each Application Group is associated with an Application Group Service Level Agreement (SLA) and each Service Provider Group are associated with Service Provider Group SLAs. See Figure 2-1 for more information. These Service Level Agreements define and regulate the contractual agreements between the telecom operator and the application provider, and cover such things as which services the application may access, the maximum bandwidth available for use, and the number of concurrent sessions that are supported.

Figure 2-1 Accounts, Groups, and SLAs



Functional Overview

Network Gatekeeper provides eleven different types of traffic paths. The application-facing interfaces of these traffic paths are largely based on the Parlay X 2.1 specifications. The functionality supported by these traffic paths includes:

- Third Party Call
  Using this traffic path, an application can set up a call between two parties (the caller and the callee), poll for the status of the call, and end the call.
- Audio Call
  Using this traffic path, an application can set up a call to a telephone subscriber and then, when the subscriber answers, play an audio message, such as a meeting reminder.
- Call Notification
  Using this traffic path, an application can set up and end notifications on call events, such as the callee in a third party call attempt is busy. In addition, in some cases the application can then reroute the call to another party.
- Call Handling
  Using this traffic path, an application can establish rules that will automatically handle calls that meet certain criteria. These rules might establish, for example, that calls from a particular number are always blocked, or are always forwarded if the initial callee is busy. In addition, the application can retrieve rules that are currently in place.

- Short Messaging

  Using this traffic path, an application can send SMS text messages, ringtones, or logos to one or multiple addresses, set up and receive notifications for final delivery receipts of those sent items, and arrange to receive SMSes meeting particular criteria from the network.

- Multimedia Messaging

  Using this traffic path, an application can send Multimedia Messages to one or multiple addresses, set up and receive notifications for final delivery receipts of those sent items, and arrange to receive MMSes meeting particular criteria from the network.

- Terminal Status

  Using this traffic path, an application can request the status (reachable, unreachable, or busy) of one or more terminals and set up and receive notifications for a change in status for particular terminals.

- Terminal Location

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

- Presence

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

- Payment

  Using this traffic path, an application can communicate charging information to an operator in situations where the cost of the service is based on the nature of the content delivered and not on connect time. For example, an end user could request the download of a music video, which costs a specific amount. The application can notify the operator that the user should be charged a particular amount or be refunded a particular amount. In the case of pre-paid accounts, it can also reserve a certain amount of the user's available funds and then charge or release the reservation depending, say, on whether or not the download was successful.

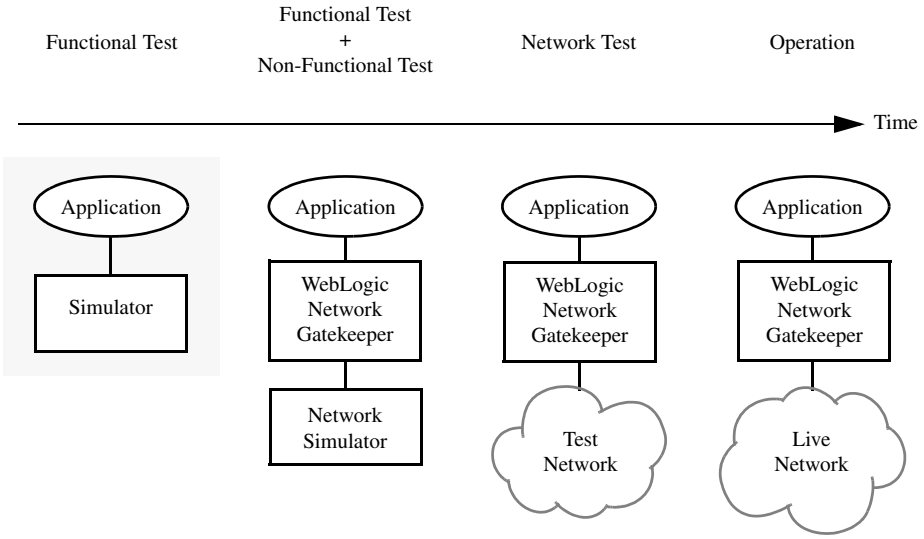
- WAP Push

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

Application Testing Workflow

Application testing in a telecom environment is usually conducted in a stepwise manner. For the first step, applications are run against simulators like the optional WebLogic Network Gatekeeper Simulator. The Network Gatekeeper Simulator emulates both the Network Gatekeeper and the underlying network, and allows developers to sort out basic functional issues without having to be connected to a network or network simulator. Once basic functional issues are sorted through, the application is connected to an instance of the Network Gatekeeper attached to a network simulator for non-functional testing. Next the application is tested against a test network, to eliminate any network related issues. Finally, the application can be placed into production on a live network. Figure 2-2 shows the complete application test flow, from the developer's functional tests to deployment in a live network. While Simulator-based tests may be performed in-house by an Application Service Provider, the other tests require the cooperation of the target network operator.

Figure 2-2 Application Testing Cycle



CHAPTER 3

Interacting with Network Gatekeeper

In order to interact with Network Gatekeeper, applications must manipulate the SOAP messages that they use to make requests in certain specialized ways. They must add specific information to the SOAP header, and, if they are using Multimedia Messaging or WAP Push, they must send their message payload as a SOAP attachment. The following chapter presents a high-level description of these mechanisms, and how they function to manage the interaction between Network Gatekeeper and the application. It covers:

- The SOAP Header
  - Authentication
  - Session Management
  - Service Correlation
  - Parameter Tunneling
- SOAP attachments

The mechanisms for dealing with these requirements programmatically depend on the environment in which the application is being developed.

Note: Clients created using Axis 1.2 or older will not work with some traffic paths. Developers should use Axis 1.4 or newer if they wish to use Axis.

For examples using the WebLogic Server environment to accomplish these sorts of tasks, see the final section of this chapter:

- Managing SOAP headers and SOAP attachments programmatically

The SOAP Header

There are three types of elements you may need to add to your application's SOAP messages to Network Gatekeeper.

Authentication

In order to secure Network Gatekeeper and the telecom networks to which it provides access, applications are usually required to provide authentication information in every SOAP request which the application submits. Network Gatekeeper leverages the WLS Web Services Security framework to process this information.

Note: WS Security provides three separate modes of providing security between a Web Service client application and the Web Service itself for message level security - Authentication, Digital Signatures, and Encryption. For an overview of WLS WS Security, see *Programming Web Services of WebLogic Server*, the "Configuring Security" chapter.

Network Gatekeeper supports three authentication types:

- Username Token
- X.509 Certificate Token
- SAML Token

The type of token that the particular Network Gatekeeper operator requires is indicated in the Policy section of the WSDL files that the operator makes available for each application-facing interface it supports. In the following WSDL fragment, for example, the required form of authentication, indicated by the `<wssp:Identity>` element, is Username Token.

Listing 3-1 WSDL fragment showing Policy

```
<s0:Policy s1:Id="Auth.xml">
       <wssp:Identity>
       <wssp:SupportedTokens>
              <wssp:SecurityToken
TokenType="http://docs.oasisopen.org/wss/2004/01/oasis200401wssusernametok
enprofile1.0#UsernameToken">
              <wssp:UsePassword
Type="http://docs.oasisopen.org/wss/2004/01/oasis200401wssusernametokenpro
file1.0#PasswordText"/>
```

```
            </wssp:SecurityToken>
            <wssp:SecurityToken
TokenType="http://docs.oasisopen.org/wss/2004/01/oasis200401wssx509tokenpr
ofile1.0#X509v3"/>
       </wssp:SupportedTokens>
       </wssp:Identity>
</s0:Policy>
<wsp:UsingPolicy n1:Required="true"/>
```

Note: If the WSDL also has a `<wssp: Integrity>` element, digital signing is required (WebLogic Server provides WS-Policy: sign.xml). If it has a `<wssp:Confidentiality>` element, encryption is required (WebLogic Server provides WS-Policy: encrypt.xml).

SOAP Header Element for Authentication

Below are examples of the three types of authentication that can be used with Network Gatekeeper.

Username Token

In the Username Token mechanism, which is specified by the use of the `<wsse:UsernameToken>` element in the header, authentication is based on a username, specified in the `<wsse:Username>` element and a password, specified in the `<wsse:Password>` element.

Two types of passwords are possible, indicated by the Type attribute in the Password element:

- `PasswordText` indicates the password is in clear text format.
- `PasswordDigest` indicates that the sent value is a Base64 encoded, SHA-1 hash of the UTF8 encoded password.

There are two more optional elements in Username Token, introduced to provide a countermeasure for replay attacks:

- `<wsse:Nonce>`, a random value that the application creates.
- `<wsu:Created>`, a timestamp.

If either or both `the Nonce` and `Created` elements are present, the Password Digest is computed as: Password_Digest = Base64(SHA-1(nonce+created+password))

When the application sends a SOAP message using Username Token, the WSEE implementation in Network Gatekeeper evaluates the username using the associated authentication provider. The authentication provider connects to the Network Gatekeeper database and authenticates the username and the password. In the database, passwords are stored as MD5 hashed representations of the actual password.

Listing 3-2 Example of a WSSE: Username Token SOAP header element

```
<wsse:UsernameToken wsu:Id="Example-1">
  <wsse:Username> myUsername </wsse:Username>
  <wsse:Password Type="PasswordText">myPassword</wsse:Password>
  <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
  <wsu:Created> ... </wsu:Created>
</wsse:UsernameToken>
```

The UserName is equivalent to the string that was called the Application Instance Group ID in Network Gatekeeper 2.2. The Password part is the password associated with this UserName when the application was provisioned in Network Gatekeeper.

For more information on Username Token, see http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

X.509 Certificate Token

In the X.509 Token mechanism, the application's identity is authenticated by the use of an X.509 digital certificate. See http://dev2dev.bea.com/pub/advisory/30.

Typically a certificate binds the certificate holder's public key with a set of attributes linked to the holder's real world identity – for example the individual's name, organization and so on. The certificate also contains a validity period in the form of two date and time fields, specifying the beginning and end of the interval during which the certificate is recognized.

The entire certificate is (digitally) signed with the key of the issuing authority. Verifying this signature guarantees

- that the certificate was indeed issued by the authority in question
- that the contents of the certificate have not been forged, or tampered with in any way since it was issued

For more information on X.509 Token, see
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

The default identity assertion provider in Network Gatekeeper verifies the authenticity of X.509 tokens and maps them to valid Network Gatekeeper users.

Note: While it is possible to use the out-of-the-box keystore configuration in Network Gatekeeper for testing purposes, these should not be used for production systems. The digital certificates in these out-of-the-box keystores are only signed by a demonstration certificate authority For information on configuring keystores for production systems, refer to *Securing WebLogic Server*, the Configuring Identity and Trust section.

The x.509 certificate common name (CN) for an application must be the same as the account UserName, which is the string that was referred to as the `applicationInstanceGroupId` in previous versions of Network Gatekeeper. This is provided by the operator when the account is provisioned.

Listing 3-3 Example of a WSSE: X.509 Certificate SOAP header element

```
<wsse:Security xmlns:wsse="..." xmlns:wsu="...">
  <wsse:BinarySecurityToken wsu:Id="binarytoken"
    ValueType="wsse:X509v3"
    EncodingType="wsse:Base64Binary">
      MIIEZzCCA9CgAwIBAgIQEmtJZc0…
  </wsse:BinarySecurityToken>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:Reference URI="#body">…</ds:Reference>
    <ds:Reference URI="#binarytoken">…</ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>HFLP…</ds:SignatureValue>
```

```
    <ds:KeyInfo>
      <wsse:SecurityTokenReference>
        <wsse:Reference URI="#binarytoken" />
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
  </ds:Signature>
</wsse:Security>
```

SAML Token

Network Gatekeeper, using WebLogic Server's WSSE implementation, supports SAML versions 1.0 and 1.1. The versions are similar. See http://www.oasis-open.org/committees/download.php/3412/sstc-saml-diff-1.1-draft-01.pdf for an overview of the differences between the versions.

In SAML, a third party, the Asserting Party, provides the identity information for a Subject that wishes to access the services of a Relying Party. This information is carried in an Assertion. In the SAML Token type of Authentication, the Assertion (or a reference to an Assertion) is provided inside the `<WSSE:Security>` header in the SOAP message. The Relying Party (which in this case is Network Gatekeeper, using the WebLogic Security framework) then evaluates the trustworthiness of the assertion, using one of two confirmation methods.

- Holder-of-Key
- Sender-Voucher

For more information on these confirmation methods, see "SAML Token Profile Support in WebLogic Web Services" in *Understanding WebLogic Security*.

Listing 3-4 Example of a WSSE: SAML Token SOAP header element

```
<wsse:Security>
<saml:Assertion MajorVersion="1" MinorVersion="0"
    AssertionID="186CB370-5C81-4716-8F65-F0B4FC4B4A0B"
    Issuer="www.test.com" IssueInstant="2001-05-31T13:20:00-05:00">
```

```
  <saml:Conditions NotBefore="2001-05-31T13:20:00-05:00"
    NotAfter="2001-05-31T13:25:00-05:00"/>
  <saml:AuthenticationStatement AuthenticationMethod="password"
    AuthenticationInstant="2001-05-31T13:21:00-05:00">
    <saml:Subject>
      <saml:NameIdentifier>
        <SecurityDomain>"www.bea.com"</SecurityDomain>
        <Name>"cn=localhost,co=bea,ou=sales"</Name>
      </saml:NameIdentifier>
    </saml:Subject>
  </saml:AuthenticationStatement>
</saml:Assertion>
...
</wsse:Security>
```

Session Management

Before an application can begin sending requests through Network Gatekeeper, it must establish a session, using the Session Manager Web Service. The session allows Network Gatekeeper to keep track of all of the traffic sent by a particular application for the duration of the session, which lasts until the session times out, based on an operator-set interval, or until the application logs out. The session is good for an entire WebLogic Network Gatekeeper domain, across clusters, and covers all traffic paths to which the application has contractual access

An application establishes a session in Network Gatekeeper by invoking the `getSession()` operation on the Session Manager Web Service. This is the only request that does not require a SessionID. In the response to this operation, a string representing the Session ID is returned to the client, and a Network Gatekeeper session, identified by the ID, is established. The session is valid until either the session is terminated by the application or an operator-established time period has elapsed. The SessionID must appear in the `wlng:Session` element in the header of every subsequent SOAP request.

Listing 3-5 Example of a SessionID SOAP header element

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

Service Correlation

In some cases the service that an application provides to its end-users may involve accessing multiple Network Gatekeeper traffic paths. For example, a mobile user might send an SMS to an application asking for the pizza place nearest to his current location. The application then makes a Terminal Location request to find the user's current location, looks up the address of the closest pizza place, and then sends the user an MMS with all the appropriate information. Three Network Gatekeeper traffic paths are involved in executing what for the application is a single service. In order to be able to correlate the three traffic path requests, Network Gatekeeper uses a Service Correlation ID, or SCID. This is a string that is captured in all the CDRs and EDRs generated by Network Gatekeeper. The CDRs and EDRs can then be orchestrated in order to provide special treatment for a given chain of service invocations, by, for example, applying charging to the chain as a whole rather than to the individual invocations.

The SCID is not provided by Network Gatekeeper. When the chain of services is initiated by an application-initiated request, the application must provide, and ensure the uniqueness of, the SCID within the chain of service invocations.

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

When the chain of services is initiated by a network-triggered request, Network Gatekeeper calls an external interface to get the SCID. This interface must be implemented by an external system. No utility or integration is provided out-of the box; this must be a part of a system integration project. It is the responsibility of the external system to provide, and ensure the uniqueness of, the SCID.

The SCID is passed between Network Gatekeeper and the application through an additional SOAP header element, the SCID element. Because not every application requires the service

correlation facility, this is an optional element. In version 3.0, this option is available only with enhanced traffic paths.

Listing 3-6 Example of a SCID SOAP header element

```
<scid id="myid"/>
```

Parameter Tunneling

Parameter tunneling is a feature that allows an application to send additional parameters to Network Gatekeeper and lets a plug-in use these parameters. This feature makes it possible for an application to tunnel parameters that are not defined in the interface that the application is using and can be seen as an extension to the application-facing interface.

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

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

Listing 3-7 SOAP header with a tunneled parameter.

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

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

SOAP attachments

The payloads for Multimedia Messages and WAP Push messages in Network Gatekeeper are sent as SOAP attachments. Listing 3-8 below shows a Multimedia Messaging `sendMessage` operation that contains an attachment carrying a jpeg image.

Listing 3-8 Example of a SOAP message with attachment (full content is not shown)

```
POST /parlayx21/multimedia_messaging/SendMessage HTTP/1.1

Content-Type: multipart/related; type="text/xml";
start="<1A07DC767BC3E4791AF25A04F17179EE>";
boundary="----=_Part_0_2633821.1170785251635"

Accept: application/soap+xml, application/dime, multipart/related, text/*

User-Agent: Axis/1.4

Host: localhost:8000

Cache-Control: no-cache

Pragma: no-cache

SOAPAction: ""

Content-Length: 4652

Connection: close

------=_Part_0_2633821.1170785251635

Content-Type: text/xml; charset=UTF-8

Content-Transfer-Encoding: binary

Content-Id: <1A07DC767BC3E4791AF25A04F17179EE>

  <?xml version="1.0" encoding="UTF-8"?>

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

      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Header>
        <ns1:Security ns1:Username="app:-4206293882665579772"
          ns1:Password="app:-4206293882665579772"
          soapenv:actor="wsse:PasswordToken"
          soapenv:mustUnderstand="1"
          xmlns:ns1="/parlayx21/multimedia_messaging/SendMessage">
        </ns1:Security>
      </soapenv:Header>
      <soapenv:Body>
        <sendMessage xmlns=
          "http://www.csapi.org/schema/parlayx/multimedia_messaging/send/v2_4/
local">
          <addresses>tel:234</addresses>
          <senderAddress>tel:567</senderAddress>
          <subject>Default Subject Text</subject>
          <priority>Normal</priority>
          <charging>
            <description xmlns="">Default</description>
            <currency xmlns="">USD</currency>
            <amount xmlns="">1.99</amount>
            <code xmlns="">Example_Contract_Code_1234</code>
          </charging>
        </sendMessage>
      </soapenv:Body>
    </soapenv:Envelope>
------=_Part_0_2633821.1170785251635
Content-Type: image/jpeg
```

```
Content-Transfer-Encoding: binary

Content-Id:

  <9FFD47E472683C870ADE632711438CC3>???? JFIF      ??
C#%$""!&+7/&)4)!"0A149;>>>%.DIC<H7=>;??
C;("(;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;??  ? w" ??           ??
7      !1AQ"aq2???#?BRr?3Cb?????             ?? '         !1"AQ2Raq???? ?
??{?????>?"7B?7!1???????Z e{????ax??5??CC??-Du?
??X?)Y!??=R@??g?????T??c????f?Wc??eCi?l?????5s??\E???6I??(?x?^???=??d?#?itoi?{
;? ??G.......

------=_Part_0_2633821.1170785251635--
```

Managing SOAP headers and SOAP attachments programmatically

This section illustrates how to manage the Network Gatekeeper required SOAP headers and SOAP attachments when you are using WebLogic Server and WebLogic Server tools to generate stubs for your Web Services clients. If you are using a different environment, the steps you need to take to accomplish these tasks will be different.

For an overview of using WebLogic Server to create Web Service clients, see the Invoking Web Services chapter of *Programming Web Services for WebLogic Server*. The following examples show particularly the use of a SOAP message handler as covered in that chapter.

Note: These examples show the use of a single message handler to add both SOAP Headers and SOAP attachments. Chains of custom message handlers are not supported in Network Gatekeeper.

The WebLogic Server environment relies heavily on using supplied Ant tasks. In Listing 3-9 a supplied Ant task, `clientgen`, is added to the standard build.xml file. A handler configuration file, `SOAPHandlerConfig.xml` is added as the value for the `handlerChainFile` attribute. `SOAPHandlerConfig.xml` is shown in Listing 3-10.

Listing 3-9 Snippet from build.xml

```
<clientgen
  wsdl="${wsdl-file}"
  destDir="${class-dir}"
```

```
  handlerChainFile="SOAPHandlerConfig.xml"
  packageName="com.bea.wlcp.wlng.test"
  autoDetectWrapped="false"
  generatePolicyMethods="true"
/>
```

The configuration file for the message handler contains the handler-name and the associated handler-class. The handler class, `TestClientHandler`, is described in Listing 3-11.

Listing 3-10 SOAPHandlerConfig.xml

```
<weblogic-wsee-clientHandlerChain
  xmlns="http://www.bea.com/ns/weblogic/90"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:j2ee="http://java.sun.com/xml/ns/j2ee">
  <handler>
    <j2ee:handler-name>clienthandler1</j2ee:handler-name>
    <j2ee:handler-class>
      com.bea.wlcp.wlng.client.TestClientHandler
    </j2ee:handler-class>
  </handler>
</weblogic-wsee-clientHandlerChain>
```

`TestClientHandler` provides the following functionality:

- Adds a Session ID to the SOAP header, see Session Management. The session ID is hardcoded into the member variable `sessionId`.

- Adds a service correlation ID to the SOAP header. See Service Correlation for more information.
- Adds a SOAP attachment in the form of a MIME message with content-type text/plain. See SOAP attachments for more information.

Listing 3-11 TestClientHandler

```
package com.bea.wlcp.wlng.client;
import javax.xml.rpc.handler.Handler;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.*;
import javax.xml.namespace.QName;
public class TestClientHandler implements Handler{
  public String sessionId = "myID";
  public String SCID = "mySCId";
  public String contenttype = "text/plain";
  public String content = "The content";

  public boolean handleRequest(MessageContext ctx) {
    if (ctx instanceof SOAPMessageContext) {
      try {
        SOAPMessageContext soapCtx = (SOAPMessageContext) ctx;
        SOAPMessage soapmsg = soapCtx.getMessage();
        SOAPHeader header = soapCtx.getMessage().getSOAPHeader();
        SOAPEnvelope envelope =
          soapCtx.getMessage().getSOAPPart().getEnvelope();
```

```
// Begin: Add session ID
Name headerElementName = envelope.createName("session","",
  "http://schemas.xmlsoap.org/soap/envelope/");
SOAPHeaderElement headerElement =
  header.addHeaderElement(headerElementName);
headerElement.setMustUnderstand(false);
headerElement.addNamespaceDeclaration("soap",
  "http://schemas.xmlsoap.org/soap/envelope/");
SOAPElement sessionId = headerElement.addChildElement("SessionId");
sessionId.addTextNode(sessionId);
// End: Add session ID
// Begin: Add Combined Services ID
Name headerElementName = envelope.createName("SCID","",
  "http://schemas.xmlsoap.org/soap/envelope/");
SOAPHeaderElement headerElement =
  header.addHeaderElement(headerElementName);
headerElement.setMustUnderstand(false);
headerElement.addNamespaceDeclaration("soap",
  "http://schemas.xmlsoap.org/soap/envelope/");
SOAPElement sessionId = headerElement.addChildElement("SCID");
sessionId.addTextNode(SCID);
// End: Add Combined Services ID
// Begin: Add SOAP attachment
AttachmentPart part = soapmsg.createAttachmentPart();
part.setContent(content, contenttype);
soapmsg.addAttachmentPart(part);
// End: Add SOAP attachment
```

```
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    return true;
  }
  public boolean handleResponse(MessageContext ctx) {
    return true;
  }
  public boolean handleFault(MessageContext ctx) {
    return true;
  }
  public void init(HandlerInfo config) {
  }
  public void destroy() {
  }
  public QName[] getHeaders() {
    return null;
  }
}
```

CHAPTER 4

Using WorkShop Controls with Network Gatekeeper

BEA Workshop for WebLogic Platform 9.2 Controls can be used to develop applications for Network Gatekeeper.

WebLogic Workshop needs to be patched with patch ID AYKE (CR309605) and patch ID 442J (CR309605).

Using WebLogic WorkShop, generate Service Controls from the WSDLs provided by Network Gatekeeper. For information on how to use WebLogic Workshop, see the documentation for BEA Workshop for WebLogic Platform.

Network Gatekeeper uses information in the SOAP header for various purposes, such as maintaining a session ID, service correlation, attachments for payload in MMS messages, and more. The Service Controls do not have methods to set these elements directly. A `Document` must be created using DOM, and then use it as a factory to create the SOAP Header Elements which are passed to the `setOutputHeaders` method as defined in the Interface ServiceControl in `com.bea.control`. Below is an outline of the workflow:

1. Create client Web Service.
2. Add the Service Control reference.
3. Add the code described below to the client code which calls the control method, as, for example `sendSMS`.

The following imports are necessary for creating SOAP headers:

Listing 4-1 Imports needed for creating SOAP headers

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

The header is created using DocumentBuilderFactory as described below.

Listing 4-2 Create header element

```
Document doc = null;
try {
  //create the document factory
  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
  factory.setNamespaceAware(true);
  doc = factory.newDocumentBuilder().newDocument();
} catch (ParserConfigurationException pce) {
  //add exception
}
Element header =
doc.createElementNS("http://schemas.xmlsoap.org/soap/envelope/",
"SOAP-ENV:Header");
```

Below is an example of how to create WSSE UsernameToken header elements and add them to the SOAP header.

Listing 4-3 Create WSSE Element

```
String nameSpace
="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.
0.xsd";

//create Security Element

Element headerContent = doc.createElementNS(nameSpace, "wsse:Security");

//Create UsernameToken Element

Element userTokenElement = doc.createElementNS(nameSpace,"wsse:UsernameToken");

//Create Username Element

Element userElement = doc.createElementNS(nameSpace, "wsse:Username");

//Append usernametext

userElement.appendChild(doc.createTextNode("usrName"));

userTokenElement.appendChild(userElement);

//Create Password Element

Element pwdElement = doc.createElementNS(nameSpace, "wsse:Password");

Attr nsAttr = doc.createAttributeNS(nameSpace, "Type");

nsAttr.setValue("PasswordText");

pwdElement.setAttributeNodeNS(nsAttr);

pwdElement.appendChild(doc.createTextNode("passwd"));

userTokenElement.appendChild(pwdElement);

headerContent.appendChild(userTokenElement);

//append Security element

header.appendChild(headerContent);
```

Below is an example of how to create a session element and add it to the SOAP header.

Listing 4-4 Create a session element

```
//Create Session Element
String ns1= "";
Element session = doc.createElementNS(ns1,"Session");
Element sessionId = doc.createElementNS(ns1, "SessionId");
sessionId.appendChild(doc.createTextNode("sessionValue"));
session.appendChild(sessionId);
//append Session Element
header.appendChild(session);
```

When the SOAP header is created it must be added to the Service Control as described below.

Listing 4-5 Append header to the Control

```
smsServiceControl.setOutputHeaders(new Element[] { header });
```

CHAPTER 5

Session management Web Service

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

Before an application can perform any operations on the Parlay X or Extended Web Services interfaces, a session must be established with Network Gatekeeper. When a session is established, a session ID is returned which must be used in each subsequent operation towards Network Gatekeeper.

Endpoint

The WSDL for the Session Manager can be found at
`http://<host>:<port>/session_manager/SessionManager`

where host and port depend on the Network Gatekeeper deployment.

Interface: SessionManager

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

Operation: getSession

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

Input message: getSession

Part name	Part type	Optional	Description
-	-	-	-

Output message: getSessionResponse

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

Referenced faults

GeneralException

Operation: changeApplicationPassword

Changes the password for an application.

Input message: changeApplicationPassword

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

Output message: changeApplicationPasswordResponse

Part name	Part type	Optional	Description
-	-	-	-

Referenced faults

-

Operation: getSessionRemainingLifeTime

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

Input message: getSessionRemainingLifeTime

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

Output message: getSessionRemainingLifeTimeResponse

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

Referenced faults

-

Operation: refreshSession

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

Input message: refreshSession

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

Output message: refreshSessionResponse

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

Referenced faults

-

Operation: destroySession

Destroys an established session.

Input message: destroySession

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

Output message: destroySessionResponse

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

Referenced faults

-

Examples

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

Listing 5-1 Get hold of the Session Manager

```
protected ClientSessionManImpl(String sessionManagerURL, PolicyBase pbase)
throws Exception {

  SessionManagerService accessservice =

    new SessionManagerService_Impl(sessionManagerURL+"?WSDL");

  port = accessservice.getSessionManager();

  pbase.prepareStub((Stub)port);

  }
```

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

Listing 5-2 Prepare the Session Manager with Username Token information

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

  private String username;
  private String password;

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

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

```
}
```

CHAPTER 6

Extended Web Services WAP Push

The Extended Web Services WAP Push Web Service allows for the sending of messages, which are rendered as WAP Push messages by the adressee's terminal. The content of the message is coded as a PAP message. It also provides an asynchronous notification mechanism for delivery status.

The payload of a WAP Push message must adhere to the following:

- WAP Service Indication Specification, as specified in Service Indication Version 31-July-2001, Wireless Application Protocol WAP-167-ServiceInd-20010731-a.
- WAP Service Loading Specification, as specified in Service Loading Version 31-Jul-2001, Wireless Application Protocol WAP-168-ServiceLoad-20010731-a.
- WAP Cache Operation Specification, as specified in Cache Operation Version 31-Jul-2001, Wireless Application Protocol WAP-175-CacheOp-20010731-a.

See http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html for links to the specifications.

The payload is sent as a SOAP attachment.

Namespaces

The PushMessage interface and service use the namespaces:

- http://www.bea.com/wlcp/wlng/wsdl/ews/push_message/interface
- http://www.bea.com/wlcp/wlng/wsdl/ews/push_message/service

The PushMessageNotification interface and service use the namespaces:

- http://www.bea.com/wlcp/wlng/wsdl/ews/push_message/notification/interface
- http://www.bea.com/wlcp/wlng/wsdl/ews/push_message/notification/service

The data types are defined in the namespace:

- http://www.bea.com/wlcp/wlng/schema/ews/push_message

In addition, Extended Web Services WAP Push uses definitions common for all Extended Web Services interfaces:

- The datatypes are defined in the namespace:
  - http://www.bea.com/wlcp/wlng/schema/ews/common
- The faults are defined in the namespace:
  - targetNamespace="http://www.bea.com/wlcp/wlng/wsdl/ews/common/faults"

Endpoint

The endpoint for the PushMessage interface is:
`http://<host:port>/ews/push_message/PushMessage`

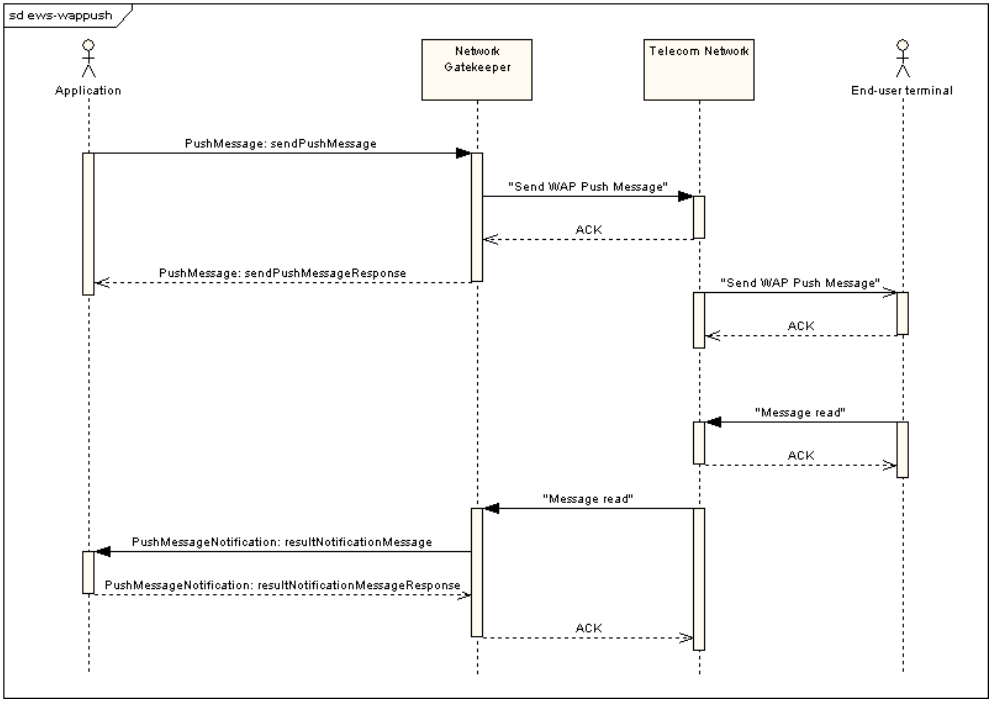
Where host and port depend on the Network Gatekeeper deployment.

Sequence Diagram

The following diagram shows the general message sequence for sending a WAP Push message from an Extended Web Services WAP Push application to the network. In this message sequence the application also receives a notification from the network indicating the delivery status of the WAP Push message, that is, that the message has been read The interaction between the network and Network Gatekeeper is illustrated in a protocol-agnostic manner. The exact operations and sequences depend on which network protocol is being used.

Note: Zero or more `resultNotificationmesages` are sent to the application, depending on parameters provided in the initial `SendPushMessage` request.

Figure 6-1 Sequence diagram Extended Web Services WAP Push



XML Schema data type definition

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

PushResponse structure

Defines the response that the Network Gatekeeper returns from a `sendPushMessage` operation.

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

ResponseResult structure

Defines the result element in the `PushResponse` structure, which is used in the response returned from a `sendPushMessage` operation.

Element Name	Element type	Optional	Description
code	xsd:string	N	A code representing the outcome when sending the push message. Generated by the network node. Possible status codes are listed in Table 6-1.
description	xsd:string	N	Textual description.

Table 6-1 Outcome status codes

Status code	Description
1000	OK.
1001	Accepted for processing.
2000	Bad request.
2001	Forbidden.
2002	Address error.
2003	Address not found.
2004	Push ID not found.
2005	Capabilities mismatch.
2006	Required capabilities not supported.
2007	Duplicate push ID.
2008	Cancellation not possible.
3000	Internal server error.
3001	Not implemented.
3002	Version not supported.
3003	Not possible.
3004	Capability matching not possible.
3005	Multiple addresses not supported.
3006	Transformation failure.
3007	Specified delivery method not possible.
3008	Capabilities not available.
3009	Required network not available.
3010	Required bearer not available.

Status code	Description
3011	Replacement not supported.
4000	Service failure.
4001	Service unavailable.

ReplaceMethod enumeration

Defines the values for the `replacePushId` parameter in the `sendPushMessage` operation. This parameter is used to replace an existing message based on a given push ID. This parameter is ignored if it is set to NULL.

Enumeration value	Description
all	Indicates that this push message MUST be treated as a new push submission for all recipients, no matter if a previously submitted push message with pushId equal to the replacePushId in this push message can be found or not.
pending-only	Indicates that this push message should be treated as a new push submission only for those recipients who have a pending push message that is possible to cancel. In this case, if no push message with pushId equal to the replacePushId in this push message can be found, the server responds with status code PUSH_ID_NOT_FOUND in the responseResult. Status code CANCELLATION_NOT_POSSIBLE may be returned in the responseResult if no message can be cancelled. Status code CANCELLATION_NOT_POSSIBLE may also be returned in a subsequent resultNotification to indicate a non-cancellable message for an individual recipient.

MessageState enumeration

Defines the values for the `messageState` parameter in a `resultMessageNotification`.

Enumeration value	Description
rejected	Message was not accepted by the network.
pending	Message is being processed.
delivered	Message successfully delivered to the network.
undeliverable	The message could not be delivered.
expired	The message reached the maximum allowed age or could not be delivered by the time specified when the message was sent. **Note:** Some network elements allows for defining policies on maximum age of messages.
aborted	The end-user terminal aborted the message.
timeout	The delivery process timed out.
cancelled	The message was cancelled.
unknown	The state of the message is unknown.

Web Service interface description

The following describes the interfaces and operations that are available in the Extended Web Services WAP Push Web Service.

Interface: PushMessage

Operations to send, or to manipulate previously sent, WAP Push messages.

Operation: sendPushMessage

Sends a WAP Push message. The message Content Entity (the payload) is provided as a SOAP attachment in MIME format. The Content Entity is a MIME body part containing the content to be sent to the wireless device. The content type is not defined, and can be any type as long as it can be described by MIME. The Content Entity is included only in the push submission and is not included in any other operation request or response.

Input message: sendPushMessage

Part name	Part type	Optional	Description
pushId	xsd:string	N	Provided by the application. Serves as a message ID. The application is responsible for its uniqueness, for example, by using an address within its control (for example a URL) combined with an identifier for the push message as the value for `pushId`. Supported types are PLMN and USER. For example: "www.wapforum.org/123" or "123@wapforum.org"
destinationAddresses	xsd:string [1..unbounded]	N	An array of end-user terminal addresses. The addresses should be formatted according to the Push Proxy Gateway Service Specification (WAP-249-PPGService-20010713-a). Example addresses: • WAPPUSH=+155519990730 TYPE=PLMN@ppg.carrier.com • WAPPUSH=john.doe%40wapforum.org TYPE=USER@ppg.carrier.com
resultNotificationEndpoint	xsd:anyURI	Y	Specifies the URL the application uses to return result notifications. The presence of this parameter indicates that a notification is requested. If the application does not want a notification, this parameter must be set to NULL.

Part name	Part type	Optional	Description
replacePushId	xsd:string	Y	The `pushId` of the still pending message to replace. The presence of this parameter indicates that the client is requesting that this message replace one previously submitted, but still pending push message. The following rules apply: • Setting the `replacePushId` parameter to NULL indicates that it is a new message. It does not replace any previously submitted message. • The initial pending (pending delivery to the end-user terminal) message is cancelled, if possible, for *all* recipients of the message. This means that it is possible to replace a message for only a subset of the recipients of the original message. • Message replacement will occur only for the recipients for whom the pending message can be cancelled.
replaceMethod	push_message_xsd:ReplaceMethod	N	Defines how to replace a previously sent message. Used in conjunction with the `replacePushId` parameter described above. Ignored if `replacePushId` is NULL.
deliverBeforeTimestamp	xsd:dateTime	Y	Defines the date and time by which the content must be delivered to the end-user terminal. The message is not delivered to the end-user terminal after this time and date. If the network node does not support this parameter, the message is rejected.
deliverAfterTimestamp	xsd:dateTime	Y	Specifies the date and time after which the content should be delivered to the wireless device. The message is delivered to the end-user terminal after this time and date. If the network node does not support this parameter, the message is be rejected.

Part name	Part type	Optional	Description
sourceReference	xsd:string	Y	A textual name of the content provider.
progressNotesRequested	xsd:boolean	Y	This parameter informs the network node if the client wants to receive progress notes. TRUE means that progress notes are requested. Progress notes are delivered via the `PushMessageNotification` interface. If not set, progress notes are not sent.
serviceCode	xsd:string	N	Used for charging purposes.
requesterID	xsd:string	N	The application ID as given by the operator.
additionalProperties	ews_common_xsd:AdditionalProperty [0...unbounded]	Y	Additional properties, defined as name/value pairs, can be sent using this parameter. The supported properties are: pap.priority, pap.delivery-method, pap.network, pap.network-required, pap.bearer, pap.bearer-required.

Output message: sendPushMessageResponse

Part name	Part type	Optional	Description
result	push_message_xsd:PushResponse	N	The response that Network Gatekeeper returns for `sendPushMessage` operation

Referenced faults

ServiceException:

ESVC0001 -A service error occurred. Error code is %1

ESVC0002 -Invalid input value for message part %1

ESVC000331 -No valid addresses provided in message part %1";

ESVC0004 -System several overloaded

ESVC0005 -The service request timed out.

PolicyException:

EPOL0001 -The policy service denied the service request. Error code is %1 and error message is %2

Interface: PushMessageNotification

Operation: resultNotificationMessage

Input message: resultNotificationMessage

Part name	Part type	Optional	Description
pushId	xsd:string	N	Defined by the application in the corresponding `sendPushMessage` operation. Used to match the notification to the message.
address	xsd:string	N	The address of the end-user terminal.
messageState	push_message_xsd:MessageState	N	State of the message.
code	xsd:string	N	Final status of the message.
description	xsd:string	Y	Textual description of the notification. Supplied by the network. May or may not be present, depending on the network node used.
senderAddress	xsd:string	Y	Address of the network node. May or may not be present, depending on the network node used.
senderName	xsd:string	Y	Name of the network node. May or may not be present, depending on the network node used.
receivedTime	xsd:dateTime	Y	Time and date when the message was received at the network node.

Part name	Part type	Optional	Description
eventTime	xsd:dateTime	Y	Time and date when the message reached the end-user terminal.
additionalProperties	ews_common_xsd:AdditionalProperty	Y	Additional properties can be sent using this parameter in the form of name/value pairs. The supported properties are: • pap.priority • pap.delivery-method • pap.network • pap.network-required • pap.bearer • pap.bearer-required Which properties are sent, if any, is dependent on the network node.

Output message: resultNotificationMessageResponse

Part name	Part type	Optional	Description
none			

Referenced faults

None.

WSDLs

The document/literal WSDL representation of the PushMessage interface can be retrieved from the Web Services endpoint.

The document/literal WSDL representation of the PushMessageNotification interface can be downloaded from

```
http://<host>:<port>/ews/push_message/wsdls/ews_common_types.xsd
http://<host>:<port>/ews/push_message/wsdls/ews_push_message_notification_
interface.wsdl
http://<host>:<port>/ews/push_message/wsdls/ews_push_message_notification_
```

service.wsdl
http://<host>:<port>/ews/push_message/wsdls/ews_push_message_types.xsd

Where host and port are depending on the Network Gatekeeper deployment.

Error Codes

The following error codes are defined for ESVC0001: Service error:

- See General error codes.
- 6001 Push response processing error.
- 6002 Bad message response.
- 6003 Push message parameter validation error.
- 11000 Invalid address.
- 11001 Invalid notification method.
- 6100 Push message send error.

The following error codes are defined for EPOL0001: Policy error:

- See General policy error codes.
- 900 Content type is not allowed for service provider.
- 901 Content type is not allowed for service provider.
- 902 Message size limit exceeded for service provider.
- 903 Message size limit exceeded for service provider.

Sample Send WAP Push Message

Listing 6-1 Example Send WAP Push Message

```
// Add handlers for MIME types needed for WAP MIME-types

MailcapCommandMap mc = (MailcapCommandMap) CommandMap.getDefaultCommandMap();

mc.addMailcap("text/vnd.wap.si;;x-java-content-handler=com.sun.mail.handlers.t
ext_xml");
```

```
CommandMap.setDefaultCommandMap(mc);
// Create a MIME-message where with the actual content of the WAP Push message.
InternetHeaders headers = new InternetHeaders();
headers.addHeader("Content-type", "text/plain; charset=UTF-8");
headers.addHeader("Content-Id", "mytext");
byte[] bytes = "Test message".getBytes();
MimeBodyPart mimeMessage = new MimeBodyPart(headers, bytes);

// Create PushMessage with only the manadatory parameters

// SendPushMessage is provided in the stubs generated from the WSDL.
SendPushMessage sendPushMessage = new SendPushMessage();
String [] destinationAddresses = {"wappush=461/type=user@ppg.o.se"};
sendPushMessage.setDestinationAddresses(destinationAddresses);
// Create “unique” pushId, using a combination of timestamp and domain.
sendPushMessage.setPushId(System.currentTimeMillis() + "@wlng.bea.com");
// ReplaceMethod is provided by the stubs generated from the WSDL.
sendPushMessage.setReplaceMethod(ReplaceMethod.pendingOnly);
// Defined by the operator/service provider contractual agreement
sendPushMessage.setServiceCode(”Service Code xxx”);
// Defined by the operator/service provider contractual agreement
sendPushMessage.setRequesterID(”Requester ID xxx”);
// Endpoint to send notifications to. Implemented on the application side.
String notificationEndpoint =
"http://localhost:80/services/PushMessageNotification";
sendPushMessage.setResultNotificationEndpoint(new URI(notificationEndpoint));

// Send the WAP Push message
```

```
PushMessageService pushMessageService = null;
// Define the endpoint of the WAP Push Web Service
String endpoint = "http://localhost:8001/ews/push_message/PushMessage?WSDL";
try {
  // Instantiate an representation of the Web Service from the generated stubs.
  pushMessageService = new PushMessageService_Impl(endpoint);
} catch (ServiceException e) {
  e.printStackTrace();
  throw e;
}
PushMessage pushMessage = null;
try {
  // Get the Web Service interface to operate on.
  pushMessage = pushMessageService.getPushMessage();
} catch (ServiceException e) {
    e.printStackTrace();
    throw e;
  }
SendPushMessageResponse sendPushMessageResponse = null;
try {
  // Send the WAP Push message.
  sendPushMessageResponse = pushMessage.sendPushMessage(sendPushMessage);
} catch (RemoteException e) {
  e.printStackTrace();
  throw e;
} catch (PolicyException e) {
  e.printStackTrace();
```

```
  throw e;
} catch (com.bea.wlcp.wlng.schema.ews.common.ServiceException e) {
  e.printStackTrace();
  throw e;
}
// Assign the pushId provided in the in the response to a local variable.
String pushId = sendPushMessageResponse.getPushId();
```

Parlay X 2.1 Interfaces

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

See http://parlay.org/en/specifications/pxws.asp for links to the specifications.

Parlay X 2.1 Third Party Call

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

Interface: ThirdPartyCall

The endpoint for this interface is:
http://<host>:<port>/parlayx21/third_party_call/ThirdPartyCall

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

MakeCall

Sets up a call between two parties.

GetCallInformation

Displays information about a call.

EndCall

Ends a call.

CancelCall

Cancels a call setup procedure.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.
- 4001 Unknown Error.
- 4002 Invalid criteria.
- 4003 Invalid interface type.
- 4004 Invalid event type.
- 4005 Invalid session ID.
- 4006 Invalid network state.
- 4007 Invalid address.
- 4008 Unsupported address plan.
- 4009 Invalid assignment ID.
- SVC0002 Invalid input
- SVC0260 Trying to cancel a call already connected
- SVC0261 Trying to terminate a call already terminated

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.
- 500 Max number of call legs exceeded for service provider.
- 501 Max number of call legs exceeded for application.
- 502 Call event criteria not allowed for service provider.

- 503 Call event criteria not allowed for application.

Parlay X 2.1 Part 3: Call Notification

This set of interfaces is compliant to ETSI ES 202 391-3 V1.2.1 (2006-12) Open Service Access (OSA); Parlay X Web Services; Part 3: Call Notification (Parlay X 2).

Interface: CallDirection

This interface is implemented by an application, and the consumer of this interface is Network Gatekeeper. The WSDL that defines the interface can be downloaded from:

`http://<host>:<port>/parlayx21/call_notification/wsdls/parlayx_call_direction_interface_2_2.wsdl`

`http://<host>:<port>/parlayx21/call_notification/wsdls/parlayx_call_direction_service_2_2.wsdl`

`http://<host>:<port>/parlayx21/call_notification/wsdls/parlayx_call_notification_types_2_2.xsd`

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

HandleBusy

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

HandleNotReachable

Network Gatekeeper calls this method, which is implemented by an application, when the called party is not reachable.

HandleNoAnswer

Network Gatekeeper calls this method, which is implemented by an application, when the called party does not answer.

HandleCalledNumber

Network Gatekeeper calls this method, which is implemented by an application, prior to call setup.

Interface: CallNotification

This interface is implemented by an application, and the consumer of this interface is Network Gatekeeper. The WSDL that defines the interface can be downloaded from:

```
http://<host>:<port>/parlayx21/call_notification/wsdls/parlayx_call_notification_interface_2_2.wsdl
```

```
http://<host>:<port>/parlayx21/call_notification/wsdls/parlayx_call_notification_service_2_2.wsdl
```

```
http://<host>:<port>/parlayx21/call_notification/wsdls/parlayx_call_notification_types_2_2.xsd
```

NotifyBusy

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

NotifyNotReachable

Network Gatekeeper calls this method, which is implemented by an application, when the called party is not reachable.

NotifyNoAnswer

Network Gatekeeper calls this method, which is implemented by an application, when the called party does not answer.

NotifyCalledNumber

Network Gatekeeper calls this method, which is implemented by an application, prior to call setup.

Interface: CallNotificationManager

The endpoint for this interface is:
http://<host>:<port>/parlayx21/call_notification/CallNotificationManager

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

StartCallNotification

Starts a subscription for call notifications.

StopCallNotification

Stops a subscription for call notifications.

Interface: CallDirectionManager

The endpoint for this interface is:
http://<host>:<port>/parlayx21/call_notification/CallDirectionManager

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

StartCallDirectionNotification

Starts a subscription for call direction notifications.

StopCallDirectionNotification

Stops a subscription for call direction notifications.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.
- CN-000001Two requests for call direction overlap with each other.
- CN-000002 Internal error to access the subscription storage.
- CN-000003 Could not find the callback plug-in.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.

Parlay X 2.1 Part 4: Short messaging

This set of interfaces is compliant to ETSI ES 202 391-4 V1.2.1 (2006-12) Open Service Access (OSA); Parlay X Web Services; Part 4: Short Messaging (Parlay X 2).

Interface: SendSms

The endpoint for this interface is: http://<host>:<port>/parlayx21/sms/SendSms

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

If a backwards-compatible traffic path is used:

- The parameter senderAddress is either of the format tel:<mailbox ID>\<mailbox password>\<Sender name> or just <sender name> depending on how the application was provisioned in Network Gatekeeper.
- The priority parameter is not supported.

SendSms

Sends an SMS to one or more destinations.

SendSmsLogo

Sends an SMS Logo to one or more destinations.

Logos in SmartMessaging and EMS are supported. The image is not scaled.

Logos in the following raw image formats are supported:

- bmp
- gif
- jpg
- png

The logos are in pure black and white (gray scale not supported). Animated images are not supported. Scaling is not supported.

If the logo shall be converted to SmartMessaging format, the image cannot be larger than 72x14 pixels.

If the logo shall be is sent in EMS format, the following rules apply:

- If the image is 16x16 pixels, the logo is sent as an EMS small picture.
- If the image is 32x32 pixels, the logo is sent as an EMS large picture.
- If the image is of any other size, the logo is sent as an EMS variable picture.
- Images up to 1024 pixels are supported.

SendSmsRingtone

Sends an SMS Ringtone to one or more destinations.

Ringtones can be in any of these formats:

- RTX
- SmartMessaging
- EMS (iMelody)

GetSmsDeliveryStatus

Gets the delivery status of a previously sent SMS.

It is possible to query delivery status of an SMS only if a callback reference was not defined when the SMS was sent. If a callback reference was defined, NotifySmsDeliveryReceipt is invoked by Network Gatekeeper and the delivery status is not stored. If the delivery status is stored in Network Gatekeeper, it is stored for a configurable period of time.

Interface: SmsNotification

This interface is implemented by an application, and the consumer of this interface is Network Gatekeeper. The WSDL that defines the interface can be downloaded from:

`http://<host>:<port>/parlayx21/sms/wsdls/parlayx_sms_notification_interface_2_2.wsdl`

`http://<host>:<port>/parlayx21/sms/wsdls/parlayx_sms_notification_service_2_2.wsdl`

`http://<host>:<port>/parlayx21/sms/wsdls/parlayx_sms_types_2_2.xsd`

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

NotifySmsReception

Sends an SMS that is received by Network Gatekeeper to an application given that the SMS fulfills a set of criteria. The criteria is either defined by the application itself, using startSmsNotification or defined using provisioning step in Network Gatekeeper.

Shortcode translation is applied.

NotifySmsDeliveryReceipt

Sends a delivery receipt that a previously sent SMS has been received by its destination. The delivery receipt is propagated to the application given that the application provided a callback reference when sending the SMS.

Interface: ReceiveSms

The endpoint for this interface is: http://<host>:<port>/parlayx21/sms/ReceiveSms

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

GetReceivedSms

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

If a backwards-compatible traffic path is used:

- The format of the parameter registrationIdentifier is tel:<mailbox ID>\<mailbox password>
- Mailbox ID and password are defined as a part of the provisioning steps by the Network Gatekeeper administrator.

  Example:

  "tel:50000\apassword"

  Received message are stored in Network Gatekeeper only for a configurable period of time

Interface: SmsNotificationManager

The endpoint for this interface is: http://<host>:<port>/parlayx21/sms/SmsNotificationManager

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

StartSmsNotification

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

Note: Service activation number may be provisioned to cater for a range of numbers via short code translations.

Note: The equivalent to this operation may have been performed as an off-line provisioning step by the Network Gatekeeper administrator.

If a backwards-compatible traffic path is used:

- The format of the parameter smsServiceActivationNumber is tel:<mailbox ID>;mboxPwd=<mailbox password>.
- Mailbox ID and password are provisioned by the Network Gatekeeper administrator.

Example:

"tel:50000;mboxPwd=apassword"

StopSmsNotification

Ends a previously started notification.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.
- 3001 Unknown error.
- 3002 Address is not in URI format.
- 3003 Invalid authentication information.
- 3004 Invalid mailbox
- 3005 Invalid criteria
- 3006 Invalid assignment ID.
- 3007 Invalid message folder.
- 3008 Invalid property.
- 3009 property not set.
- 3010 Mailbox is locked.
- 3011 No matching properties found.
- 3012 No multimedia message.
- 3013 Insufficient privileges.
- 3014 Message not removed.
- 3015 Invalid message ID.
- 3016 Message property size too large.
- 3017 Mailbox folder too large.

- 3018 Failed to find transaction number.
- 3019 Properties array empty.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.
- 200 Message event criteria not allowed for service provider.
- 201 Message event criteria not allowed for application.

Parlay X 2.1 Part 5: Multimedia messaging

This set of interfaces is compliant to ETSI ES 202 391-5 V1.2.1 (2006-12) Open Service Access (OSA); Parlay X Web Services; Part 5: Multimedia Messaging (Parlay X 2).

Interface: SendMessage

The endpoint for this interface is:
http://<host>:<port>/parlayx21/multimedia_messaging/SendMessage

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

SendMessage

Sends a multimedia message. The content of the message is sent as a SOAP attachment. E-mail is not supported.

The parameter senderAddress is either of the format tel:<mailbox ID>\<mailbox password>\<Sender name> or just <sender name> depending on how the application was provisioned in Network Gatekeeper.

The priority parameter is not supported.

GetMessageDeliveryStatus

Gets the delivery status of a previously sent MMS.

It is possible to query delivery status of an MMS only if a callback reference was not defined when the message was sent. I a callback reference was defined, NotifyMessageDeliveryReceipt is invoked by Network Gatekeeper and the delivery status is not stored. If the delivery status is stored in Network Gatekeeper, it is stored for a configurable period of time.

Note: Network Gatekeeper may be configured not to store delivery status for MMS.

Interface: ReceiveMessage

The endpoint for this interface is:
http://<host>:<port>/parlayx21/multimedia_messaging/ReceiveMessage

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

GetReceivedMessages

Polls Network Gatekeeper for received messages.

The registrationIdentifier is required. The priority parameter is not supported.

The format of the parameter registrationIdentifier is tel:<mailbox ID>\<mailbox password>

Mailbox ID and password are defined as a part of the provisioning steps by the Network Gatekeeper administrator.

Example:

```
"tel:50000\apassword"
```

Received message are stored in Network Gatekeeper only for a configurable period of time.

GetMessageURIs

Not supported.

GetMessage

Gets a specific message that was received by WLNG and belongs to the application.

Interface: MessageNotification

This interface is implemented by an application, and the consumer of this interface is Network Gatekeeper. The WSDL that defines the interface can be downloaded from:

```
http://<host>:<port>/parlayx21/multimedia_messaging/wsdls/parlayx_mm_notification_interface_2_4.wsdl
```

```
http://<host>:<port>/parlayx21/multimedia_messaging/wsdls/parlayx_mm_notification_service_2_4.wsdl
```

```
http://<host>:<port>/parlayx21/multimedia_messaging/wsdls/parlayx_mm_types_2_4.xsd
```

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

NotifyMessageReception

Sends a notification to an application that an MMS destined for the application is received by Network Gatekeeper.

NotifyMessageDeliveryReceipt

Sends a notification to an application that a previously sent MMS has been delivered to its destination.

Note: Network Gatekeeper can be configured to support delivery notifications or not.

Interface: MessageNotificationManager

The endpoint for this interface is:
http://<host>:<port>/parlayx21/multimedia_messaging/MessageNotificationManager

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

StartMessageNotification

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

The format of the parameter MessageServiceActivationNumber is

tel:<mailbox ID>;mboxPwd=<mailbox password>

Mailbox ID and password are provisioned by the Network Gatekeeper administrator.

Example:

```
"tel:50000;mboxPwd=apassword"
```

StopMessageNotification

Ends a previously started notification.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.
- 3001 Unknown error.
- 3002 Address is not in URI format.

- 3003 Invalid authentication information.
- 3004 Invalid mailbox
- 3005 Invalid criteria
- 3006 Invalid assignment ID.
- 3007 Invalid message folder.
- 3008 Invalid property.
- 3009 property not set.
- 3010 Mailbox is locked.
- 3011 No matching properties found.
- 3012 No multimedia message.
- 3013 Insufficient privileges.
- 3014 Message not removed.
- 3015 Invalid message ID.
- 3016 Message property size too large.
- 3017 Mailbox folder too large.
- 3018 Failed to find transaction number.
- 3019 Properties array empty.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.
- 200 Message event criteria not allowed for service provider.
- 201 Message event criteria not allowed for application.
- 202 Multi media content type not allowed for service provider.
- 203 Multi media content type not allowed for application.
- 204 Message encoding type not allowed for service provider.
- 205 Message encoding type not allowed for application.

- 206 Multimedia message size limit exceeded for service provider.
- 207 Multimedia message size limit exceeded for application.
- 208 Message size limit exceeded for service provider.
- 209 Message size limit exceeded for application.

Parlay X 2.1 Part 6: Payment

This set of interfaces is compliant to ETSI ES 202 391-6 V1.2.1 (2006-12), Open Service Access (OSA); Parlay X Web Services; Part 6: Payment (Parlay X 2).

Network Gatekeeper does not manage Payment operations, it passes on the requests to a network node that performs these operations.

Interface: AmountCharging

The endpoint for this interface is: http://<host>:<port>/parlayx21/payment/AmountCharging

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

ChargeAmount

Charges an amount from an account.

RefundAmount

Refunds an amount to an account.

Interface: VolumeCharging

The endpoint for this interface is: http://<host>:<port>/parlayx21/payment/VolumeCharging

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

ChargeVolume

Charges a volume from an account.

GetAmount

Converts a volume to an amount.

RefundVolume

Refunds a volume to an account.

Interface: ReserveAmountCharging

The endpoint for this interface is:
http://<host>:<port>/parlayx21/payment/ReserveAmountCharging

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

ReserveAmount

Reserves an amount.

ReserveAdditionalAmount

Reserves an additional amount.

ChargeReservation

Charge the reserved amount.

ReleaseReservation

Releases the reserved amount.

Interface: ReserveVolumeCharging

The endpoint for this interface is:
http://<host>:<port>/parlayx21/payment/ReserveVolumeCharging

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

GetAmount

Converts a given volume to an amount.

ReserveVolume

Reserves a volume.

ReserveAdditionalVolume

Reserves an additional volume.

ChargeReservation

Charges a reserved volume.

ReleaseReservation

Releases a reserved volume.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.
- 5001 Unknown error.
- 5002 Invalid amount.
- 5003 Invalid unit.
- 5004 Invalid session ID.
- 5005 Invalid user.
- 5006 Invalid account.
- 5007 Invalid request number.
- 5008 Invalid currency.
- 5009 Invalid volume.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.
- 100 Amount not allowed for service provider.
- 101 Amount not allowed for application.
- 102 Currency not allowed for service provider.
- 103 Currency not allowed for application.

Parlay X 2.1 Part 8: Terminal Status

This set of interfaces is compliant to ETSI ES 202 391-8 V1.2.1 (2006-12), Open Service Access (OSA); Parlay X Web Services; Part 8: Terminal Status (Parlay X 2).

Network Gatekeeper does not hold status information, it passes on the requests to a network node that performs these operations.

Interface: TerminalStatus

The endpoint for this interface is:
http://<host>:<port>//parlayx21/terminal_status/TerminalStatus

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

GetStatus

Gets the status of a terminal.

GetStatusForGroup

Gets the status of a group of terminals.

Group URIs are not supported

Interface: TerminalStatusNotificationManager

The endpoint for this interface is:
http://<host>:<port>/parlayx21/terminal_status/TerminalStatusNotificationManager

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

StartNotification

Initiates status notifications to the application for a given address and criteria. Only status changes matching the criteria are notified to the application.

Group URIs are not supported in the parameter Addresses.

The parameters Frequency and Duration are not supported.

When the number of notifications is equal to the number given in parameter Count, the subscription for notifications is removed.

EndNotification

Ends a previously started notification.

Interface: TerminalNotification

This interface is implemented by an application, and the consumer of this interface is Network Gatekeeper. The WSDL that defines the interface can be downloaded from:

```
http://<host>:<port>/parlayx21/terminal_status/wsdls/parlayx_terminal_status_notification_interface_2_2.wsdl
```

```
http://<host>:<port>/parlayx21/terminal_status/wsdls/parlayx_terminal_status_notification_service_2_2.wsdl
```

```
http://<host>:<port>/parlayx21/terminal_status/wsdls/parlayx_terminal_status_types_2_2.xsd
```

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

StatusNotification

Notifies an application about a change of status for a terminal.

StatusError

Notifies an application that the subscription for status notifications was cancelled by network Gatekeeper.

StatusEnd

Notifies an application that no more status notifications are being sent to the application.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.

Parlay X 2.1 Part 9: Terminal location

This set of interfaces is compliant to ETSI ES 202 391-9 V1.2.1 (2006-12), Open Service Access (OSA); Parlay X Web Services; Part 9: Terminal Location (Parlay X 2).

Interface: TerminalLocation

The endpoint for this interface is:
http://<host>:<port>/parlayx21/terminal_location/TerminalLocation

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

GetLocation

Gets the location for a terminal.

GetTerminalDistance

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

GetLocationForGroup

Gets the location for one or more terminals.

Interface: TerminalLocationNotificationManager

The endpoint for this interface is:
http://<host>:<port>/parlayx21/terminal_location/TerminalLocationNotificationManager

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

StartGeographicalNotification

Initiates location notifications to the application when one or more terminal changes their location according to a criteria.

StartPeriodicNotification

Initiates location notifications to the application on a periodic basis.

EndNotification

Ends a previously started notification.

Interface: TerminalLocationNotification

This interface is implemented by an application, and the consumer of this interface is Network Gatekeeper. The WSDL that defines the interface can be downloaded from:

`http://<host>:<port>/parlayx21/terminal_location/wsdls/parlayx_terminal_location_notification_interface_2_2.wsdl`

`http://<host>:<port>/parlayx21/terminal_location/wsdls/parlayx_terminal_location_notification_service_2_2.wsdl`

`http://<host>:<port>/parlayx21/terminal_location/wsdls/parlayx_terminal_location_types_2_2.xsdl`

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

LocationNotification

Notifies an application about a change of location for a terminal.

LocationError

Notifies an application that the subscription for location notifications was cancelled by network Gatekeeper.

LocationEnd

Notifies an application that no more location notifications are being sent to the application.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.
- 6001 Unknown error.
- 6002 Could not find a plug-in.
- 6003 Time-out in plug-in.
- 6004 Requested location information is not available.
- 6005 General problem in the underlying network.
- 6006 The network is unauthorized to obtain the location.

- 6007 The application is not authorized to obtain the location.
- 6008 Unknown subscriber.
- 6009 The subscriber is currently not reachable.
- 6010 Failed to obtain the subscriber's location.
- 6012 Timeout value not in accepted interval.
- 6013 NULL parameter not allowed.
- 6014 Periodic interval too low.
- 6015 Application is not activated.
- 6016 Invalid interface type.
- 6017 Requested accuracy can not be delivered.
- 6018 Requested response time can not be delivered.
- 6019 Invalid ID.
- 6020 Invalid reporting interval.
- 6022 Invalid trigger.
- 6023 Invalid trigger request.
- 6024 Invalid parameter.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.

Parlay X 2.1 Part 10: Call handling

This set of interfaces is compliant to ETSI ES 202 391-10 V1.2.1 (2006-12), Open Service Access (OSA); Parlay X Web Services; Part 10: Call Handling (Parlay X 2).

Interface: CallHandling

The endpoint for this interface is: http://<host>:<port>/parlayx21/call_handling/CallHandling

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

SetRules

Sets call handling rules for the destination of a call.

SetRulesForGroup

Sets call handling rules for multiple destinations of a call.

GetRules

Gets call handling rules for a given destination.

ClearRules

Clears all call handling rules for a the given destinations.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.

Parlay X 2.1 Part 11: Audio call

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

Interface: PlayAudio

The endpoint for this interface is: http://<host>:<port>/parlayx21/audio_call/AudioCall

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

PlayTextMessage

Plays a message to the given destination address. The message is given as a text.

PlayAudioMessage

Plays a message to the given destination address. The message is given as an audio file.

PlayVoiceXmlMessage

Plays a message to the given destination address. The message is given as an VoiceXML file.

GetMessageStatus

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

EndMessage

Cancel or stops the playing of the message.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.

Parlay X 2.1 Part 14: Presence

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

Interface: PresenceConsumer

The endpoint for this interface is: http://<host>:<port>//parlayx21/presence/PresenceConsumer

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

subscribePresence

Subscription to get presence information about a presentity.

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

getUserPresence

Get presence information about a presentity.

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

startPresenceNotification

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

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

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

endPresenceNotification

Ends a previously started notification.

Interface: PresenceNotification

This interface is implemented by an application, and the consumer of this interface is Network Gatekeeper. The WSDL that defines the interface can be downloaded from:

`http://<host>:<port>/parlayx21/presence/wsdls/parlayx_presence_notification_interface_2_3.wsdl`

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

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

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

statusChanged

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

statusEnd

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

notifySubscription

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

subscriptionEnded

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

Interface: PresenceSupplier

This interface is not supported.

publish

Not supported.

getOpenSubscriptions

Not supported.

updateSubscriptionAuthorization

Not supported.

getMyWatchers

Not supported.

getSubscribedAttributes

Not supported.

blockSubscription

Not supported.

Error Codes

The following error codes are defined for SVC0001: Service error:

- See General error codes.
- PRESENCE-000001Failed to use the default duration for a notification.
- PRESENCE-000002 Failed to use the default count for a notification.
- PRESENCE-000003 The application have no SIP-URI mapping configured.
- PRESENCE-000004 Internal error.

The following error codes are defined for POL0001: Policy error:

- See General policy error codes.

About notifications

When an application has started a notification, the notification is persisted. This means that if an application has started a notification and destroys the session or logs out, the notification is still registered and matching notifications are sent to the application when it connects to Network Gatekeeper.

General error codes

The following are general error codes for SVC0001: Service error:

- Null sessionID (loginTicket) expired.
- WNG-000000 No error.
- WNG-000001 Unknown error.
- WNG-000002 Storage service error.
- PLG-000001 Could not find remote ejb home in access tier.
- PLG-000002 Could not create the ejb.
- PLG-000003 Could not access callback ejb.
- SIP-000001 Could not find remote ejb home.
- SIP-000002 Could not create the ejb.
- SIP-000003 Could not access remote ejb.
- SIP-000004 Could not create the SIP session.
- SIP-000005 Failed to send sip message.
- SIP-000006 Internal sip stack error.
- OSA-000001 Invalid network state.
- OSA-000002 Invalid interface type.
- OSA-000003 Invalid event type.
- OSA-000004 Unsupported address plan.
- OSA-000005 Communication failure.

- 10000 Unknown error.
- 10001 Database errors.
- 10002 CORBA error, for example related to time-outs.
- 10003 System error, for example related to Network Gatekeeper Core.
- 10004 System severely overloaded.
- 10005 Policy deny.
- 10006 Invalid EndUserIdentifier.
- 10007 Invalid requester.
- 10008 Invalid service code.
- 10009 Invalid application session.
- 10010 No available plug-in.
- 10011 Invalid account.

General policy error codes

The following are general error codes for POL0001: Policy error:

- 0 Unspecified.
- 1 No service contract for service provider.
- 2 No service contract for application.
- 3 Service contract out of date application.
- 4 Service contract out of date for application.
- 5 Blacklisted method for service provider.
- 6 Blacklisted method for application.
- 7 Request limit reached for service provider.
- 8 Request limit reached for application.
- 11 Service provider is deactivated.

- 12 Application account is deactivated.
- 13 Quota limit reached for service provider.
- 14 Quota limit reached for application.

Code examples

Below are some code examples that illustrate how to use the Parlay X interfaces.

Example: sendSMS

Below is an example of sending an SMS.

Listing 7-1 SendSMS example

```
org.csapi.schema.parlayx.sms.send.v2_2.local.SendSms request =

new org.csapi.schema.parlayx.sms.send.v2_2.local.SendSms();

SimpleReference sr = new SimpleReference();

sr.setEndpoint(new
URI("http://localhost:8111/SmsNotificationService/services/SmsNotification?WSD
L"));

sr.setCorrelator("cor188");

sr.setInterfaceName("InterfaceName");

ChargingInformation charging = new ChargingInformation();

charging.setAmount(new BigDecimal("1.1"));

charging.setCode("qwerty");

charging.setCurrency("USD");

charging.setDescription("some charging info");

sendInf.setCharging(charging);

URI[] uri = new URI[1];

uri[0] = new URI("1234");

request.setAddresses(uri);
```

```
request.setCharging(charging);

request.setReceiptRequest(sr);

request.setMessage("we are testing sms!");

request.setSenderName(“6001”);

org.csapi.schema.parlayx.sms.send.v2_2.local.SendSmsResponse response =

smport.sendSms(request);

String sendresult = response.getResult();

System.out.println("result: " + sendresult);
```

Example: startSmsNotification

Below is an example on using startSmsNotification.

Listing 7-2 startSmsNotification example

```
org.csapi.schema.parlayx.sms.notification_manager.v2_3.local.StartSmsNotificat
ion parameters =

new
org.csapi.schema.parlayx.sms.notification_manager.v2_3.local.StartSmsNotificat
ion();

parameters.setCriteria("hello");

SimpleReference sr = new SimpleReference();

sr.setEndpoint(new
URI("http://localhost:8111/SmsNotificationService/services/SmsNotification?WSD
L"));

sr.setCorrelator("cor189");

sr.setInterfaceName("interfaceName");

parameters.setReference(sr);

URI uri = new URI("tel:6001;mboxPwd=6001");

parameters.setSmsServiceActivationNumber(uri);
```

```
port.startSmsNotification(parameters);
```

Example: getReceivedSms

Below is an example on polling for SMSes using getReceivedSms.

Listing 7-3 getReceivedSms example

```
org.csapi.schema.parlayx.sms.receive.v2_2.local.GetReceivedSms parameters =
new org.csapi.schema.parlayx.sms.receive.v2_2.local.GetReceivedSms();
parameters.setRegistrationIdentifier("1");
org.csapi.schema.parlayx.sms.receive.v2_2.local.GetReceivedSmsResponse
response =
port.getReceivedSms(parameters);
org.csapi.schema.parlayx.sms.v2_2.SmsMessage[] msgs =
response.getResult();
if(msgs != null) {
  for(org.csapi.schema.parlayx.sms.v2_2.SmsMessage msg : msgs) {
    System.out.println(msg.getMessage());
  }
}
```

Example: sendMessage

Below is an example on sending an MMS.

Listing 7-4 sendMessage example

```
org.csapi.schema.parlayx.multimedia_messaging.send.v2_4.local.SendMessage
request =
```

```
new
org.csapi.schema.parlayx.multimedia_messaging.send.v2_4.local.SendMessage();

ChargingInformation charging = new ChargingInformation();

charging.setAmount(new BigDecimal("1.1"));

charging.setCode("qwerty");

charging.setCurrency("USD");

charging.setDescription("some charging info");

sendInf.setCharging(charging);

SimpleReference sr = new SimpleReference();

if(getProperty("notification_mt").equalsIgnoreCase("true")) {

  sr.setEndpoint(new
URI(getProperty(ClientConstants.NOTIFICATION_LISTENER_URL)));

  sr.setCorrelator(getProperty("correlator"));

  sr.setInterfaceName(getProperty("interfacename"));

}

URI[] uri = new URI[1];

uri[0] = new URI("1234");

request.setAddresses(uri);

request.setCharging(charging);

request.setPriority(MessagePriority.fromString("Default"));

request.setReceiptRequest(sr);

request.setSenderAddress("6001");

request.setSubject("subject");

org.csapi.schema.parlayx.multimedia_messaging.send.v2_4.local.SendMessageRespo
nse response =

smport.sendMessage(request);

String sendresult = response.getResult();

System.out.println("sendresult: " + sendresult);
```

Example: getReceivedMessages and getMessage

Below is an example on polling for a received MMS.

Listing 7-5 getReceivedMessages and getMessage example

```
org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetReceivedMe
ssages parameters =

new
org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetReceivedMe
ssages();

parameters.setPriority(org.csapi.schema.parlayx.multimedia_messaging.v2_4.Mess
agePriority.fromString("Default"));

parameters.setRegistrationIdentifier("2");

org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetReceivedMe
ssagesResponse result =

port.getReceivedMessages(parameters);

org.csapi.schema.parlayx.multimedia_messaging.v2_4.MessageReference[] refs =

result.getResult();

if(refs != null) {

  for(org.csapi.schema.parlayx.multimedia_messaging.v2_4.MessageReference ref :
refs) {

    String id = ref.getMessageIdentifier();

    org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetMessag
e p2 =

    new
org.csapi.schema.parlayx.multimedia_messaging.receive.v2_4.local.GetMessage();

    p2.setMessageRefIdentifier(id);

    port.getMessage(p2);

  }

}
```

Example: getLocation

Below is an example of getting the location of a terminal.

Listing 7-6 getLocation example

```
org.csapi.schema.parlayx.terminal_location.v2_2.local.GetLocation parameters =
new org.csapi.schema.parlayx.terminal_location.v2_2.local.GetLocation();
parameters.setAcceptableAccuracy(5);
parameters.setAddress(new URI("1234"));
parameters.setRequestedAccuracy(5);
TimeMetric maximumAge = new TimeMetric();
maximumAge.setMetric(TimeMetrics.fromString("Hour"));
maximumAge.setUnits(10);
parameters.setMaximumAge(maximumAge);
TimeMetric responseTime = new TimeMetric();
responseTime.setMetric(TimeMetrics.fromString("Hour"));
responseTime.setUnits(1);
parameters.setResponseTime(responseTime);
DelayTolerance tolerance = DelayTolerance.fromString("NoDelay");
parameters.setTolerance(tolerance);
org.csapi.schema.parlayx.terminal_location.v2_2.local.GetLocationResponse
response =
port.getLocation(parameters);
org.csapi.schema.parlayx.terminal_location.v2_2.LocationInfo result =
response.getResult();
System.out.println("accuracy : " + result.getAccuracy());
System.out.println("altitude : " + result.getAltitude().floatValue());
System.out.println("latitude : " + result.getLatitude());
```

```
System.out.println("longitude : " + result.getLongitude());
System.out.println("timestamp : " + result.getTimestamp());
```

CHAPTER 8

Access Web Service (deprecated)

Note: The Access Web Service is deprecated and should only be used by older, existing applications, in order to provide a migration path for these applications. *WebLogic Server Web Services security cannot be used when using the Access Web Service* and must be turned off in WebLogic Network Gatekeeper to be able to use the Access Web Service.

The Access Web Service contains operations for establishing a session with Network Gatekeeper, changing the application's password, querying the amount of time remaining in the session, refreshing the session, and terminating the session.

Before an application can perform any operations on the Parlay X or Extended Web Services interfaces, a session must be established with Network Gatekeeper. When a session is established, a session ID (loginTicket) is returned which must be used in each subsequent operation towards Network Gatekeeper.

The loginTicket shall be present in the SOAP Header element Security, see below. Once the login ticket is acquired, it must be sent in the SOAP header together with a username/password combination each time a Web Service method is invoked. See Examples.

Endpoint

The WSDL for the Access Web Service can be found at
`http://<host:port>/parlayx21/access/Access`

where host and port depend on the Network Gatekeeper deployment.

Interface: Access

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

Operation: applicationLogin

Logs the application into the WebLogic Network Gatekeeper and retrieves a login ticket. This login ticket represents the session and must be added to the SOAP header of every subsequent request that the application makes to the Network Gatekeeper.

In most cases, the login ticket is only valid for a certain time interval, set by the operator. Once the time period has expired, the application has a second operator-set time period to refresh the login ticket. Until the ticket is refreshed, the application can not make any other requests. The operation used to refresh the ticket is refreshLoginTicket, see Operation: refreshLoginTicket. If the ticket is not refreshed during this second period, the session is destroyed, and the application must log back in.

Input message: applicationLoginRequest

Part name	Part type	Optional	Description
serviceProvider	s1:String	N	ID of the service provider as given by the operator or the service provider.
application	s1:String	N	ID of the application as given by the operator or the service provider.
applicationInstanceGroup	s1:String	N	ID of the application instance group as given by the operator or the service provider.
password	s1:String	N	Password for the application as given by the operator or the service provider. Note that this may also have been changed by the by the application provider.

Output message: applicationLoginResponse

Part name	Part type	Optional	Description
applicationLoginReturn	s1:string	N	ID of the login-session. This ID is used for each request towards WebLogic Network Gatekeeper. It must be included in the SOAP header of every subsequent request. If an application logs in several times, the same ID is returned.

Referenced faults

AccessException

GeneralException

Operation: applicationLogout

Logs an application out of the Network Gatekeeper. Destroys the login session and the corresponding login ticket.

Input message: applicationLogoutRequest

Part name	Part type	Optional	Description
loginTicket	s1:string	N	ID of the login-session. The login ticket is retrieved when the application logs in or when it refreshes the login ticket.

Output message: applicationLogoutResponse

Part name	Part type	Optional	Description
-	-	-	-

Referenced faults

AccessException

GeneralException

Operation: changeApplicationPassword

Changes the password for an application.

Input message: changeApplicationPasswordRequest

Part name	Part type	Optional	Description
loginTicket	s1:string	N	ID of the login-session. The login ticket is retrieved when the application logs in or when it refreshes the login ticket.
oldPassword	s1:string	N	The current password.
newPassword	s1:string	N	The new password.

Output message: changeApplicationPasswordResponse

Part name	Part type	Optional	Description
-	-	-	-

Referenced faults

AccessException

GeneralException

Operation: getLoginTicketRemainingLifeTime

Reports the remaining amount of time the login ticket is valid.

Input message: getLoginTicketRemainingLifeTimeRequest

Part name	Part type	Optional	Description
sessionId	s1:string	N	ID of the login-session. The login ticket is retrieved when the application logs in or when it refreshes the login ticket.

Output message: getLoginTicketRemainingLifeTimeReturn

Part name	Part type	Optional	Description
getLoginTicketRemainingLifeTimeReturn	s1:int	N	The time until the login ticket expires. The time is given in minutes.

Referenced faults

AccessException

GeneralException

Operation: refreshLoginTicket

Refreshes the login ticket. This refreshed login ticket must be provided in the SOAP header in all subsequent method calls. The login ticket can be refreshed for a limited, operator-set time interval after the previous login ticket has expired. If this time interval expires, the application must login again. Network Gatekeeper expiration timers are reset, but the same login ticket is returned.

Input message: refreshLoginTicketRequest

Part name	Part type	Optional	Description
loginTicket	s1:string	N	The ID of an established session.
serviceProviderID	s1:string	N	ID of the service provider as given by the operator or the service provider.
applicationID	s1:string	N	ID of the application as given by the operator or the service provider.
applicationInstanceGroupID	s1:string	N	ID of the application instance group as given by the operator or the service provider.
password	s1:string	N	Password for the application as given by the operator or the service provider. Note that this may also have been changed by the by the application provider.

Output message: refreshLoginTicketResponse

Part name	Part type	Optional	Description
refreshLoginTicketReturn	s1:string	N	The refreshed ID of the login-session. This ID is used in each request towards WebLogic Network Gatekeeper. It must be included in the SOAP header of every subsequent request.

Referenced faults

AccessException

GeneralException

Exceptions

AccessException

Exceptions of this type are raised when there are error conditions related to the Access Web Service. Other error conditions are reported using the exception GeneralException.

Part name	Part type	Optional	Description
exceptionMessage	xsd:string	Y	Description of exception.
errorCode	xsd:int	N	Code defining the exception.

GeneralException

Exceptions of this type are raised when the applications session has expired or there are communication problems with the underlying platform.

Part name	Part type	Optional	Description
exceptionMessage	xsd:string	Y	Description of exception.
errorCode	xsd:int	N	Code defining the exception.

Examples

Defining the security header

The loginTicket shall be present in the SOAP Header element Security, see below. Once the login ticket is acquired, it must be sent in the SOAP header together with a username/password combination each time a Web Service method is invoked.

Network Gatekeeper 2.2 used a non-standard security header, as described below.

The loginTicket is supplied in the Password attribute.

Listing 8-1 Network Gatekeeper 2.2 security header (example)

```
<soapenv:Header>
  <ns1:Security ns1:Username="app:-2810834922008400383"
    ns1:Password="app:-2810834922008400383" soapenv:actor="wsse:PasswordToken"
    soapenv:mustUnderstand="1"
    xmlns:ns1="http://localhost:6001/parlayx21/terminal_location/TerminalLocat
ion">
  </ns1:Security>
</soapenv:Header>
```

Below is an example of how to add a Network Gatekeeper 2.2 security header using Axis. The Username attribute must be present but is not used. The header must be added to the Web Service request.

Listing 8-2 Add a Network Gatekeeper 2.2 security header (Axis)

```
org.apache.axis.message.SOAPHeaderElement header =
 new org.apache.axis.message.SOAPHeaderElement(wsdlUrl, "Security", "");
header.setActor("wsse:PasswordToken");
 header.addAttribute(wsdlUrl, "Username", ""+userName);
 header.addAttribute(wsdlUrl, "Password", ""+loginTicket);
 header.setMustUnderstand(true)
```