



Advanced Application Interface Specification

System Version: Location Studio 2.0

Openwave Systems Inc.

Abstract

This document describes Openwave's Advanced Application Interface. This interface allows for the application to use Location Studio to perform location requests, messaging, record billing events, and verify the subscriber.

Document ID: Final
Document Status:

Document Version: 1.7
Date: 13 January 2003

**Openwave Systems
1400 Seaport Boulevard
Redwood City, CA 94063
USA**

OPENWAVE PROPRIETARY AND CONFIDENTIAL

Interface Control Document

REVISION HISTORY

Version Level	Date of Issue	Remarks	Revised By:
0.1	06/27/02	First Draft	Richard Wong
0.2	07/17/02	Minor fixes, updated wap push class	Richard Wong
0.3	07/29/02	Openwave formatting	Richard Wong
0.4	08/08/02	Updated WSDL, minor fixes	Richard Wong
0.5	08/19/02	Updated optional parameters, added detail of format of return items	Richard Wong
0.6	09/02/02	Many changes and additions. WSDL examples removed. Java examples included.	Mats Cedervall
0.7	09/04/02	Update after review meeting	Mats Cedervall
0.71	09/12/02	Some changes in 0.7 were missed because of a WORD crash. These changes are redone in 0.71	Mats Cedervall
1.3	10/24/02	Added error code information	Rose Reynolds
1.4	11/15/02	Added getLocation default values	Rose Reynolds
1.5	11/26/02	Added SMS and WAP Proxy info	Rose Reynolds
1.6	01/10/03	10.1 added information that the mapping of error codes are only done when a simple location request is made.	Per Hubinette
1.7	01/13/03	Added missing parameter 'alt' in getLocation response	Mattias Arbin

TABLE OF CONTENTS

REVISION HISTORY	2
1 INTRODUCTION	5
1.1 Purpose of the Document.....	5
1.2 Glossary of Terms	5
2 REFERENCES	7
3 SYSTEM OVERVIEW.....	8
3.1 Overall Architecture	8
4 SUBSCRIBER VERIFICATION AND VALIDATION.....	9
4.1 Typical Subscriber Verification and Validation Request/Response Sequences	9
4.1.1 Subscriber Verification.....	9
4.1.1.1 Typical Call Flow	9
4.1.2 Subscriber Validation	10
4.1.2.1 Typical Call flow.....	10
4.2 Interface Description.....	11
4.2.1 Authenticate client.....	11
4.2.2 Validate subscription	12
4.2.3 Initiate Verify Subscriber	12
4.2.4 Complete Verify Subscriber.....	13
5 MESSAGING	14
5.1 Typical Messaging Request/Response Sequences	14
5.1.1 Downstream SMS or WAP Example	14
5.1.2 Upstream SMS Proxy Example.....	15
5.1.3 Upstream WAP Proxy Example	15
5.1.4 WAP Ask Example.....	16
5.2 Interface Description.....	17
5.2.1 sendSmsMessage	17
5.2.2 sendWapMessage	18
5.2.3 WAP Proxy Format	19
5.2.4 SMS Proxy Format	20
6 EVENT BILLING	21
6.1 Typical Event Billing Request/Response Sequence	21

Interface Control Document

6.2 Interface Description.....	21
6.2.1 sendEvent.....	22
7 LOCATION REQUESTS	23
7.1 LIF	23
7.2 Simple Location Request	23
7.3 Typical Location Request/Response Sequence.....	23
7.4 Interface Description.....	24
7.4.1 LIF based request.....	24
7.4.1.1 getLocation	24
7.4.2 Simple Location request	26
7.4.2.1 getLocation	26
8 APPLICATION NOTES	28
8.1 Valid LSt ID Types.....	28
8.2 Valid Media Types	28
8.3 WAP proxy.....	28
8.4 WAP Ask.....	29
9 EXAMPLE REQUESTS AND RESPONSES	30
9.1 Pseudo example	30
9.2 Java example.....	30
10 ERROR MESSAGES	32
10.1 Location Request Error Codes	33
11 WSDL.....	36
11.1 Subscriber Validation	36
11.2 Messaging.....	38
11.3 Event Billing.....	40
11.4 Location	41

1 INTRODUCTION

Openwave's Location Studio provides the capability for applications to access the functionality of the operator's network without custom integration. This access is granted through Openwave's Advanced Application Interface (AAI). The AAI is implemented over Web Services for the maximum flexibility and ease of implementation for developers. The following functionality is available through the AAI:

Subscriber Validation - Location Studio provides a mechanism for LCS clients to validate subscribers, subscriptions, and LCS client credentials. The subscriber validation also provides clients with an alias to anonymously identify users.

Messaging - Location Studio provides SMS and WAP functionality for LCS clients. Location Studio supports upstream and downstream messaging. Downstream messages are initiated by the LCS client, sent to end user through Location Studio. Upstream messages are sent by end users, proxied by Location Studio to LCS clients. Location Studio also supports a "WAP Ask" functionality as well as privacy proxies.

Event Billing - Location Studio offers LCS clients the functionality to submit event billing records to the operator.

Location - Location Studio offers LCS clients the functionality to request subscriber location over the AAI interface. In Location Studio it is also possible to retrieve location via the XML-based LIF-MLP 3.0 interface.

The WSDL files for the standard deployment of the AAI can be found at Openwave's SDK website or obtained directly from Openwave. The wireless operator may have slightly different functionality available. Please contact your wireless operator for more information.

1.1 Purpose of the Document

This document is intended to assist developers understand Openwave's Advanced Application Interface.

1.2 Glossary of Terms

Table 1: Glossary of Terms

Term	Definition
AAI	Advanced Application Interface
DTD	Document Type Definition
GMLC	Gateway Mobile Location Center
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
LCS	Location Services
LSt	Location Studio

Interface Control Document

Term	Definition
MPC	Mobile Positioning Center
MS	Mobile Station
MSID	Mobile Station Identifier
SMSC	Short Message Service Center
SSL	Secure Socket Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator
XML	Extensible Markup Language

2 REFERENCES

- [1] Openwave, "Implementing Web Services in LocationStudio" Version 1.1, March 2002.
- [2] W3C website on WSDL, <http://www.w3.org/TR/wsd/>
- [3] LIF, "Mobile Location Protocol Specification Version 3.0.0", June 3, 2002.
- [4] Openwave, "LIF MLP 3.0.0 Interface Statement of Compliance." which can be found at **Error! Reference source not found.**

3 SYSTEM OVERVIEW

3.1 Overall Architecture

A sample location service architecture is shown in Figure 1.

LCS Client - a software and/or hardware entity that interacts with a LCS Server for the purpose of obtaining location information for one or more Mobile Stations.

Location Studio – LSt is a gateway into the operator's network to securely and easily obtain subscriber location data. . LSt provides abstraction from the underlying location infrastructure.

GMLC/MPC – The Gateway Mobile Location Center (GSM networks) or the Mobile Positioning Center (CDMA networks) provides location information from the operator's network.

SMSC – The Short Message Service Center is the gateway to sending short messages to mobile subscribers.

WAP GW – The WAP gateway is used by Location Studio to send WAP push messages to subscribers.

Billing System – The billing systems of the operator collects billing information and produces billing statements to subscribers and/or clients.

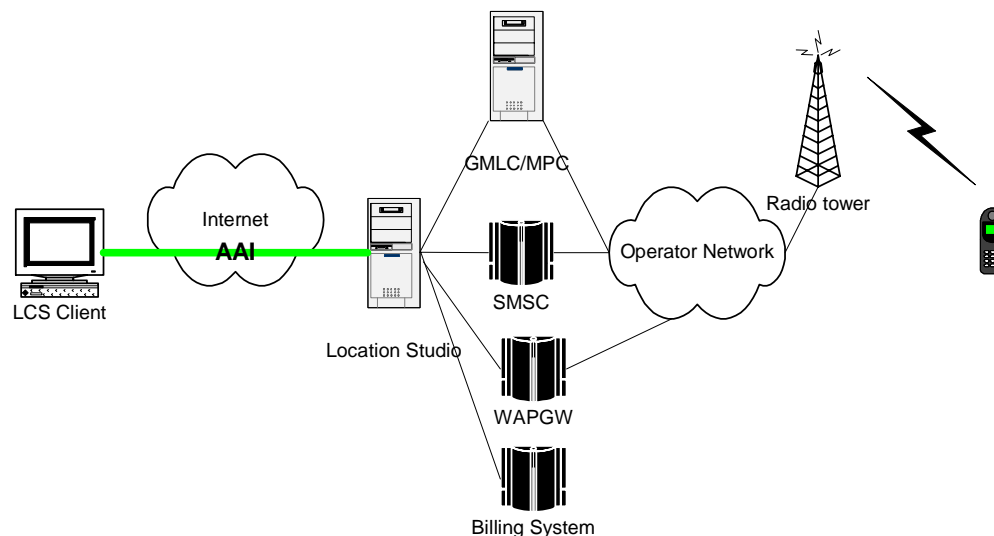


Figure 1 Sample Location Service Architecture

The AAI is implemented using Web Services. Web Services is platform and programming language independent way to expose some business functionality over the Internet using the SOAP protocol. For more information on Web Services see Implementing Web Services in LocationStudio [1] or the World Wide Web Consortium's website on WSDL[2].

4 SUBSCRIBER VERIFICATION AND VALIDATION

The verification and validation feature in Location Studio's AAI provides a mechanism for subscribers to remain anonymous and for LCS clients to validate subscribers, subscriptions, and LCS client credentials. The LCS clients can make use of this interface to:

- Validate that an end-user is still a customer of the operator
- Validate that the end-user is authorized by the operator to access their service
- Validate that the end-user has adequate credit remaining in his/her pre-paid account
- Verify that an end-user registering for a service using an internet browser is in possession of the Mobile Station he/she registers (identity verification loop using SMS pin code)
- Receive a persistent alias that may be used to identify the mobile station in subsequent transactions (location, messaging, billing, toolkit services) with Location Studio while maintaining privacy and anonymity.

The advantages of using these features are that:

- Only requests from valid subscribers with established billing relationship are serviced.
- It provides support for anonymity and privacy for internet-based community and gaming services
- No network resources are consumed (e.g. location attempt) if the subscriber validation is negative.

4.1 Typical Subscriber Verification and Validation Request/Response Sequences

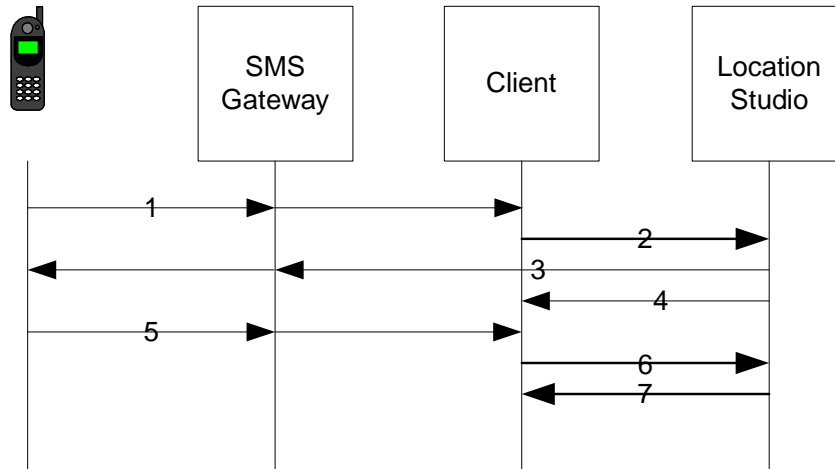
4.1.1 Subscriber Verification

The subscriber verification feature of the AAI was designed to solve 2 problems:

1. If a user signs up for a service, the client can verify that the user is in possession of the handset that they claims to have. This is designed to prevent malicious or inadvertent use of a service.
2. If the user wants to remain anonymous to the Client, a persistent anonymous ID can be generated and assigned to the subscriber by Location Studio. The Client will not know who the user is and must always go through LSt to interact with the subscriber. Personalization on the client site by the subscriber can be maintained as the alias is persistent.

4.1.1.1 Typical Call Flow

Interface Control Document



1. The end user registers with the application via SMS
2. The LCS client sends "initiateVerifySubscriber" to LSt
3. Location Studio sends a code to the end user via SMS
4. Location Studio returns a registration alias to the LCS client as the response to the "initiateVerifySubscriber" call.
5. The end user enters the code to the LCS client
6. The LCS client sends "completeVerifySubscriber" with the code and the verification ID to Location Studio
7. Location Studio returns a persistent alias that the LCS client can use in subsequent location requests to Location Studio for that end user

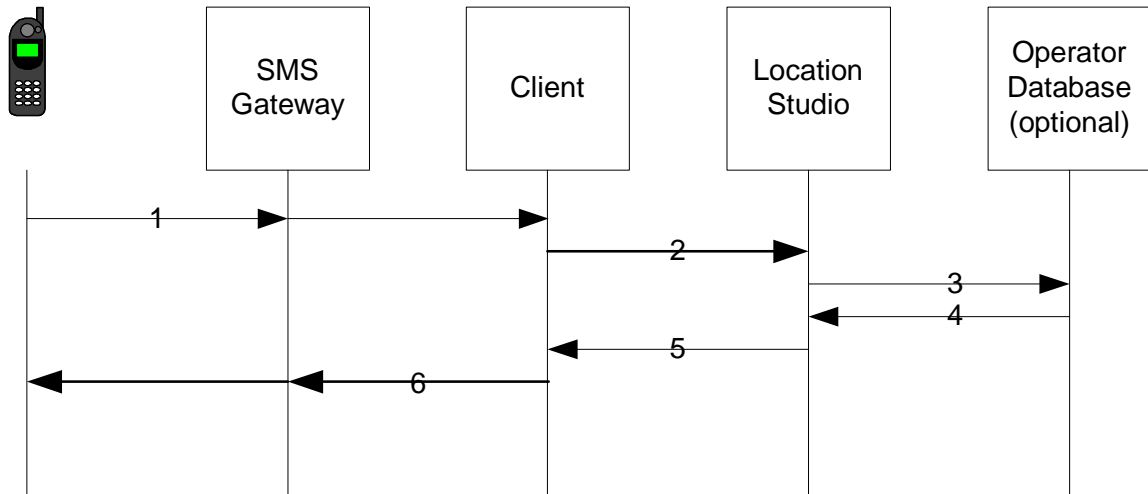
In the preceding example, the end user could have also registered with the client through the wired internet from a computer. In that specific case, everything except for the first step, which is done directly to the client through a web page, remains the same.

4.1.2 Subscriber Validation

The Subscriber Validation allows the Client to validate the subscriber against a number of checks in Location Studio's and/or the operator's databases. This can be useful to validate subscription to a service or if there are sufficient funds to complete a transaction.

4.1.2.1 Typical Call flow

Interface Control Document



1. The end user invokes a function in the LCS client
2. The LCS client sends "validateSubscription" to LSt. Location Studio can check the internal database to verify the subscriber is authorized by the operator to use this client. If desired, the operator may specify an external database for this lookup as illustrated in the optional steps 3 and 4.
3. OPTIONAL. Location Studio queries the operator billing system (or some other data depending on the context of the client) to check if the user is allowed by the operator to use the service.
4. OPTIONAL The billing system returns that the user is not allowed to use the service.
5. Location Studio returns to the LCS client "Not_Subscriber".
6. The LCS client returns an error message to the end user

4.2 Interface Description

The subscriber validation feature has several methods available for use. These are:

Operation	Description
authenticateClient	See Section 4.2.1
validateSubscription	See Section 4.2.2
initiateVerifySubscriber	See Section 4.2.3
completeVerifySubscriber	See Section 4.2.4

4.2.1 Authenticate client

Name	authenticateClient	
Description	This method is used by LCS clients to validate their user names and passwords.	
Mandatory Parameters	clientID	String Identifying the LCS client
	password	String that authenticates the LCS client

Interface Control Document

Optional Parameters	none	
Return Value	Response Code	Integer value that maps to a response. See section 10
	Response Detail	String with optional additional text information about the response

4.2.2 Validate subscription

Name	validateSubscription	
Description	This interface is used by LCS clients to validate the subscriber against some database. The validation that occurs will be based on the context of client submitting the request.	
Parameters	clientID password subscriber	Identifies the LCS client Authenticates the LCS client An object that consists of strinSubscriber ID and string ID Type. For a list of valid ID types see section 8.1.
Optional Parameters	none	
Return Values	Response Code	Integer value that maps to a response. See section 10
	Response Detail	String with optional additional text information about the response
	OperatorName	String. The name of the operator who has the subscription. For single operator deployments this will always be the same.
	transferredDate	Date type. The date the subscription was transferred. This is returned if the subscription has been transferred to another person.

4.2.3 Initiate Verify Subscriber

Name	initiateVerifySubscriber	
Description	This is the first step in verifying a subscriber. The client uses this method to receive a registration ID and to get LSt to send a PIN to the user. The registration ID and the PIN must be returned to LSt when using "complete verify subscriber".	
Mandatory Parameters	clientID password subscriberID message	String. Identifies the LCS client String. Authenticates the LCS client Subscriber ID object that contains two Strings, Type and Identification, defining the subscriber. Valid types are phone number and operator aliases. For a description of the alias types see section 8.1. String. The message that is sent to the end user together with the pin code. The maximum length of this message is configurable by the operator. The

Interface Control Document

	default maximum is 160 characters.	
Optional Parameters	codeSignature	String. Place holder for where the pin code should be inserted in the message. If not specified the default value is %CODE%. This means e.g. that the text %CODE% in message will be replaced by a code (PIN) generated by Location studio.
	messageDelimiter	String. If the message must be split in to many messages, the delimiter specifies where they may be split. If not specified the default value is NULL.
Return Values	Response Code	Integer value that maps to a response. See section 10
	Response Detail	String. Optional additional text information about the response"
	verificationAlias	String. Unique id for this verification

4.2.4 Complete Verify Subscriber

Name	completeVerifySubscriber	
Description	This interface is used by LCS clients to complete the verification of subscribers. Upon submission of a valid PIN and registration ID, the client will be returned a persistent alias	
Mandatory Parameters	clientID	String. Identifies the LCS client
	password	String. Authenticates the LCS client
	verificationAlias	String. Unique id for this verification
	code	String. Code the end user has received from Location Studio
Optional Parameters	None	
Return Values	Response Code	Integer value that maps to a response. See section 10
	Response Detail	String. Optional additional text information about the response"
	subscriberID	Subscriber ID object that contains two Strings, Type and Identification, defining the subscriber. Valid types are phone number and operator aliases. For this method the Type will always be "PSID" (Persistent Subscriber ID).

5 MESSAGING

The Wireless Messaging feature in Location Studio's AAI provides messaging functionality for third-party LCS clients. Wireless messages in this case are either SMS or WAP-push.

The feature supports both upstream and downstream messaging between Clients (applications) and subscribers. Down-stream messages are defined as those initiated by the Client and sent to one or more mobile stations through Location Studio. Upstream messages are originated by mobile subscribers and 'tunneled' through Location Studio to LCS clients.

Advantages of using the Messaging part of the Location Studio AAI include:

- The LCS Client is sheltered from having to support every underlying SMSC protocol (LSt supports SMPP, UCP and CIMD2),
- LSt ID protection services (subscriber alias) can be used for anonymity and privacy
- Message Transaction Detail Records provide auditing and accounting functions

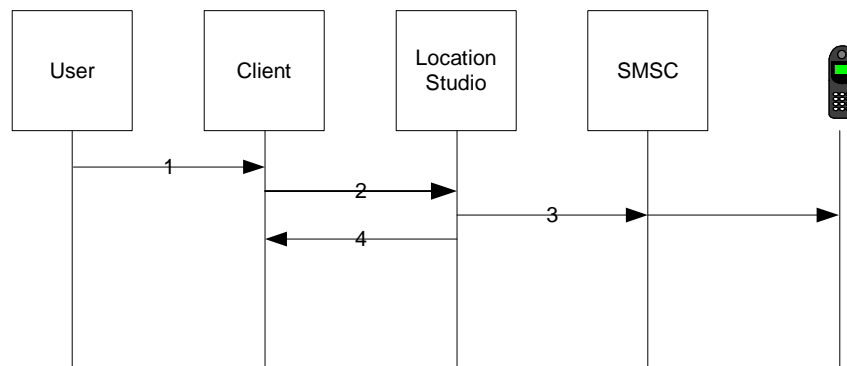
The messaging feature of LSt also encompasses "WAP ask" functionality as well as WAP and SMS privacy proxies. WAP Ask allows the subscriber to have LSt ask for permission to release location to the Client. The privacy proxies provide:

- Replacement of phone number or WAP gateway generated identifier with Location Studio-generated subscriber identifiers (TSID/PSID). The TSID/PSID may be used by the application on all LSt external interfaces.
- TSID/PSID identifiers enable anonymity (application does know subscriber identity or phone number). The PSID is persistent from connection to connection and can provide Personalization and automatic Sign-on capabilities.

5.1 Typical Messaging Request/Response Sequences

5.1.1 Downstream SMS or WAP Example

The following call flow is from an example "Friend Finder" application. A user wants to send an SMS or WAP message to all nearby friends by using that feature in a friend finder application.



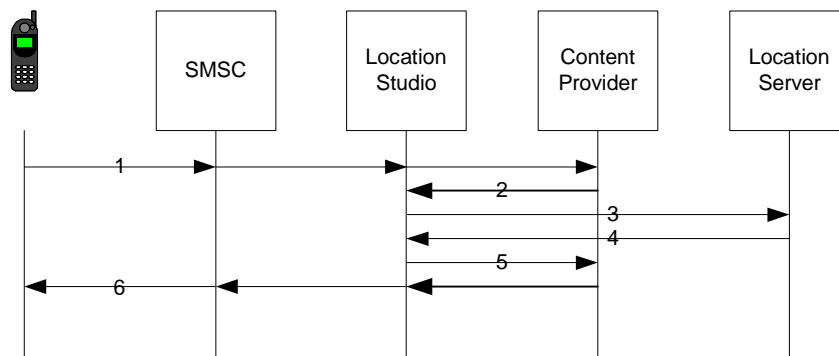
1. The end user requests that the client sends a message to the nearby friends.
2. The client issues a "sendSmsMessage" or "sendWapMessage" to LSt with the list of recipients (any valid LSt identifier is acceptable. See section 8.1) and the message to be sent.

Interface Control Document

3. LSt sends the messages to the SMSC or WAP GW and the SMSC or WAP GW sends the messages to the users.
4. LSt sends a response code back to the client.

5.1.2 Upstream SMS Proxy Example

Upstream communications do not use the AAI directly, but are discussed here for completeness. Upstream SMS communications use Location Studio's SMS Proxy. Use of the SMS proxy requires the use of specially formed LSt URLs to replace the standard client URLs in order to have the session proxied by LSt. A description of the construction of these URLs can be found in section 5.2.4. The following call flow is from an example "Where am I?" application where privacy is required. In this example application, the end user initiates the transaction by sending an SMS asking for their current location.

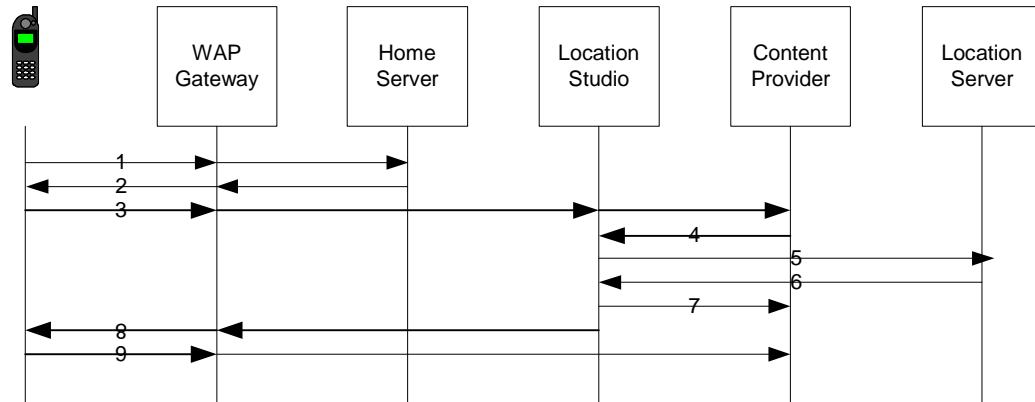


1. The end user sends the word "where" to the carrier's SMS-C and the SMS-C forwards the message to the Location Studio SMS Proxy. The proxy forwards the message to the LCS client after replacing the MSISDN with a TSID or PSID generated by LSt. The format of these messages is described in section 5.2.4.
2. The LCS client requests location for that user.
3. LSt checks the authentication and authorization of the request and if successful, sends a location request to Location Manager.
4. Location Manager responds with the location.
5. LSt returns the location to the client as requested.
6. The LCS client sends the response to the end user through Location Studio.

5.1.3 Upstream WAP Proxy Example

Upstream communications do not use the AAI directly, but are discussed here for completeness. Upstream WAP communications use Location Studio's WAP Proxy. Use of the WAP proxy requires the use of specially formed LSt URLs to replace the standard client URLs in order to have the session proxied by LSt. A description of the construction of these URLs can be found in section 5.2.3. The following call flow is from an example location content service where privacy is required. In this example application, the end user initiates the transaction by sending a WAP request for some location content.

Interface Control Document

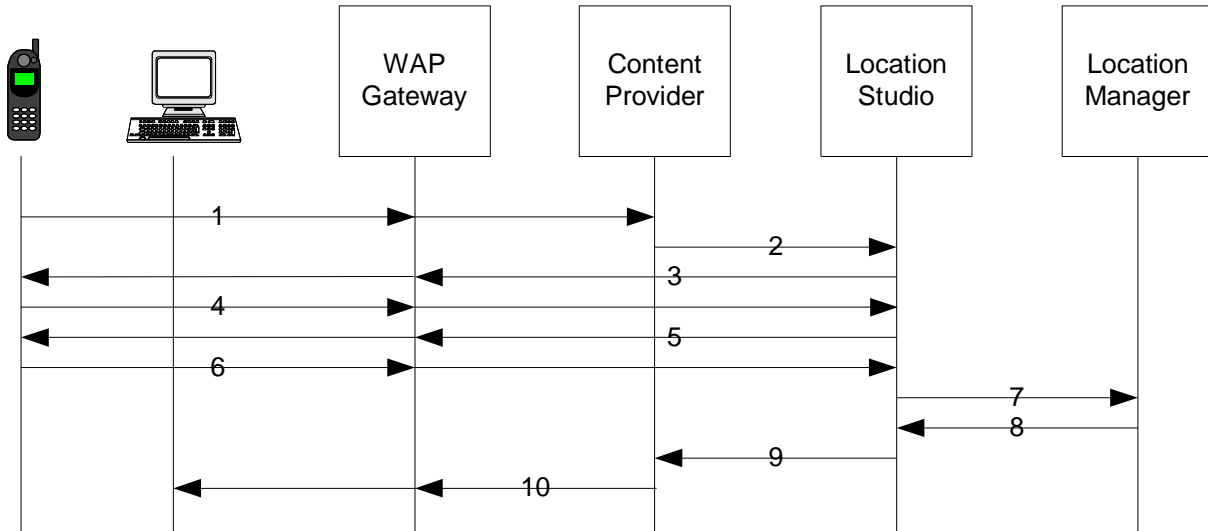


1. User connects to their homepage
2. Some location service link is served to the user.
3. The end user selects a location service that is proxied through LSt. The message is sent to Location Studio. Location Studio forwards the message to the LCS client after replacing the MSISDN or WAP identifier with a TSID or PSID generated by LSt. The format of these messages is described in section 5.2.3.
4. The client requests location for that user.
5. LSt checks the authentication and authorization of the request and if successful, sends a location request to Location Manager.
6. LM responds with the location.
7. LSt returns the location to the client as requested.
8. The LCS client sends the response to the end user through Location Studio.
9. Links on the response page that are not location sensitive are linked directly to the target.

5.1.4 WAP Ask Example

WAP ask is invoked when the subscriber has set their permissions to "Ask". When a location request is received in such a case, a WAP push is sent to the subscriber requesting permission to release the location information. If the user says yes, the information is sent back to the client. For more information on WAP ask see section 8.4.

Interface Control Document



1. A passive location request is made for the location of a subscriber.
2. The Client makes a location request to LSt. LSt checks the subscriber profile and finds that the subscriber has set the permission to "ask".
3. LSt issues a WAP push/UP notification to the gateway. The gateway issues an alert to the subscriber
4. The subscriber chooses to view the alert and is linked to the alert URL.
5. LSt serves up the "Ask" deck.
6. The subscriber responds with an allow location request.
7. LSt requests location from LM
8. LSt receives location from LM
9. LSt returns location to the client
10. Client returns content to the subscriber.

5.2 Interface Description

5.2.1 sendSmsMessage

Name	sendSmsMessage	
Description	Send an SMS message to the subscriber.	
Mandatory Parameters	clientID	String. identifies the LCS client
	password	String. Authenticates the LCS client
	body	String. The actual message being sent. The maximum length of this message is configurable by the operator. If the body length exceeds the maximum allowed by the operator the message will be truncated. The default maximum is 160 characters.
	recipients	Subscriber ID object. One or more recipients as end user identifiers. Valid end user identifiers are defined in section 8.1. The maximum number of recipients is 25.

Interface Control Document

Optional Parameters	<div> <div>delimiter</div> <div>String. If the message must be split in to many messages, the delimiter specifies where they may be split. The length of the delimiter string must be equal to one.</div> </div>
Return Values	<div> <div>If more than one recipient was selected, the return response will have an array of responses with each of the subscribers having a reponse..</div> <div> <div>Response Code</div> <div>Integer value that maps to a response. See section 10</div> </div> <div> <div>Response Detail</div> <div>String. Optional additional text information about the response</div> </div> <div> <div>Subscriber</div> <div>Subscriber ID object. Type and Identification of the subscriber. Valid types are phone number and operator aliases. For a description of the alias types see section 8.1.</div> </div> </div>

5.2.2 sendWapMessage

Name	sendWapMessage	
Description	Send a WAP message to the subscriber.	
Mandatory Parameters	<div> <div>clientID</div> <div>String. Identifies the LCS client</div> </div> <div> <div>password</div> <div>String. Authenticates the LCS client</div> </div> <div> <div>body</div> <div>String. The actual message being sent. The maximum length of this message is configurable by the operator. If the body length exceeds the maximum allowed by the operator the message will be truncated. The default maximum is 160 characters.</div> </div> <div> <div>mediaType</div> <div>The WAP media type. See section 8.2 for details</div> </div> <div> <div>recipients</div> <div>Subscriber ID object. One or more subscribers as end user identifiers. Valid end user identifiers are defined in section 8.1. The maximum number of recipients is 25.</div> </div>	
Optional Parameters	none	
Return Values	<div> <div>If more than one recipient was selected, the return response will have an array of responses with each of the subscribers having a reponse.</div> <div> <div>Response Code</div> <div>Integer value that maps to a response. See section 10</div> </div> <div> <div>Response Detail</div> <div>String. Optional additional text information about the response</div> </div> <div> <div>Subscriber</div> <div>Subscriber ID object. Type and Identification of the subscriber. Valid types are phone number and operator aliases. For a description of the alias types see section 8.1</div> </div> </div>	

5.2.3 WAP Proxy Format

This section describes the format of the WAP proxy URLs. For context and more information, see the “Upstream WAP Proxy” call flow in section 5.1.3.

In step 3 of the example call flow, the end user initiates a transaction by selecting a location service that is proxied through LSt. In order for the user's request to be proxied by LSt, the URL must be in the following format:

`http://<LSt host name>/<Proxy Path>/<Original path>/<SMS Bnumber><Original CGI parameters>[&<MSISDN|WAP ID parameter|other resolvable ID>]`

where

- < LSt host name> is the host name of LSt,
- <Proxy Path> is the path to the LSt Wap Proxy,
- < Original path> is the original path of the target URL,
- < SMS Bnumber> is the B number of the LCS client when it is used with SMS. This number is used also in the WAP case to identify the LCS client original CGI parameters,
- < Original CGI parameters> are any CGI parameters needed by the user's request and
- < MSISDN|WAP ID parameter|other resolvable ID> uniquely identifies the subscriber. This part is added by the WAP Gateway, which means that the URL without the [...] part is the format of the link selected by the end user. The inclusions of the MSISDN will be different for different WAP gateways. Some gateways will include the Wap ID and some will put the id in the header rather than in the URL. Location Studio handles all cases by professional services adaptation of its Wap Proxy.

For example, here is an example of the URL for a message between the WAP gateway and the LSt proxy:

http://lst_wap_proxy.operator.com/locationstudio/waproxy/dialog/start/4477?action=12345&MSISDN=46733201025

The LSt proxy forwards all incoming messages to an LCS client. The LSt proxy looks up the client and its corresponding Post URL field using the SMS BNumber part of the incoming URL and replaces the MSISDN parameter with a TSID, PSID or OSID depending on configuration parameters in the client profile.

The message sent by the LSt proxy to the LCS client will be in the following format:

`http://<LCS>ClientPostURL>/<OriginalPath><OriginalCGIParameters>&TSID|PSID|OSID=<string>[TSID|PSID=<string>]`

where

- <LCSClientPostURL> is the post URL found in the SMS BNumber look up,
- <OriginalPath> is the path found in the incoming message,
- <OriginalCGIParameters> are the CGI parameters found in the incoming message and
- <string> is the TSID, PSID or OSID associated with the incoming MSISDN.

This is the format in which the LCS client should expect to receive messages from the LSt WAP Proxy. For example, if the LCS client application is FriendFinder at the operator M2, the outgoing URL may be:

Interface Control Document

<http://wap.M2ff.com/dialog/start?action=12345&PSID=7656754215342364536253&TSID=1a2e3a3456aac3>

5.2.4 SMS Proxy Format

This section describes the format of the SMS proxy URLs. For context and more information, see the “Upstream SMS Proxy” call flow in section 5.1.2.

In step 1 of the example call flow, the end user initiates a transaction by sending an SMS message to an LCS client that is proxied through LSt. The SMS proxy feature requires that LSt proxy all SMS traffic to and from the specified LCS client. The LSt SMS proxy will forward all incoming requests to the LCS client, translating MIN or MSISDN to a TSID or PSID for each transaction.

The URL posted to the LCS client application has the following format:

<http://<LCSCClientPostURL>?TSID|PSID|OSID=<digits>&message=<SMS sent by user>>

where

- <LCSCClientPostURL> is the post URL of the LCS client application,
- <digits> is the TSID, PSID or OSID associated with the MIN or MSISDN of the incoming SMS message and
- <SMS sent by user> is the SMS message from the incoming message.

This is the format in which the LCS client should expect to receive messages from the LSt SMS Proxy. For example, if the LCD client application is FriendFinder at the operator M2, the outgoing URL may be:

[http://sms.M2ff.com?PSID=7656754215342364536253&message="Where"](http://sms.M2ff.com?PSID=7656754215342364536253&message=)

6 EVENT BILLING

The Wireless Billing feature in Location Studio's AAI provides LCS clients the functionality to submit application/event-level billing records.

Billing events submitted through this interface convey consolidated macro-transactions that result from an action that is billable to the end subscriber using the service. For example, a billing event may be submitted by a yellow-pages application provider for a "premium content search" resulting in a single billing entry on the subscriber's account. However, in order to complete that single billable event there may have been multiple underlying micro-transactions required to complete the request – one or more location requests, one or more geo-coding events, a map rendered, turn-by-turn directions, etc. These micro-events are consolidated into one macro billing event.

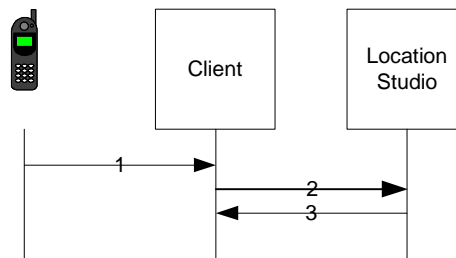
Advantages of using the WBP include:

- Service-level billing events allow integration with the operator billing system so that transactions are charged directly on the subscriber's monthly bill, or debited from pre-pay accounts.
- Facilitates revenue sharing and Bill-On-Behalf-Of relationships between operators and application providers.
- Supports correlation of micro-transactions with the service billable macro-transaction for better understanding of cost-of-delivery.
- No need to custom integration to the operator's billing system.

6.1 Typical Event Billing Request/Response Sequence

Typically the LCS client and the carrier that hosts a Location Studio have defined the possible billing events that particular LCS client may generate. Each billing event is assigned an id, unique for that LCS client, and provisioned in Location Studio.

The actual end user actions behind each event are agreed to by the LCS client provider and the carrier. The LCS client is responsible for keeping track of when these actions occur (consolidating micro events if necessary) and submitting billing events to Location Studio.



1. User requests some billable action from the client
2. Client issues "sendEvent" to LSt.
3. LSt returns "OK"

6.2 Interface Description

Interface Control Document

6.2.1 sendEvent

Name	sendEvent	
Description	Send a macro billing event to LSt for the operator to bill the subscriber.	
Parameters	<div>clientID</div> <div>password</div> <div>eventID</div> <div>subscriber</div> <div>startTime</div> <div>endTime</div> <div>trackingID</div>	<div>String. Identifies the LCS client</div> <div>String. Authenticates the LCS client</div> <div>String. Identifies what event that has occurred. This value must be known by both the operator and the client to identify and successfully bill the event. Default maximum is 50 characters.</div> <div>Subscriber ID object. Identifies the end user that will be billed for this event. Valid end user identifiers are defined in section 8.1.</div> <div>Date object. Time stamp when this event started. The format of the time is YYYY/MM/DD HH:mm:ss:zzz</div> <div>Date object. Time stamp when this event ended. The format of the time is YYYY/MM/DD HH:mm:ss:zzz</div> <div>String. Identifier for other information that the LCS client wishes to pass to the operator. Default maximum is 50 characters.</div>
Optional Parameters	none	
Return Values	<div>Response Code</div> <div>Response Detail</div>	<div>Integer value that maps to a response. See section 10</div> <div>String. Optional additional text information about the response</div>

7 LOCATION REQUESTS

Location Studio supports 3 types of location requests.

- One is XML over http and is based based on the Location Interoperability Forum (LIF) recommendation, Mobile Location Protocol (MLP), as the standards based location interface for third-party LCS clients. This protocol is standalone and not considered to be part of the AAI. Please see the LIF MLP 3.0 statement of Compliance document for further details.
- The second one is also based on the LIF MLP 3.0 standard, but it is adapted to web services. This protocol is part of AAI.

The third one is a more streamlined request model with a subset of features available. This protocol is based on LIF MLP, but all parameters that are foreseen to not being used so often are removed. For most applications it will be sufficient to use this part of AAI.

7.1 LIF

The MLP specification is very extensive, and includes many parameters in anticipation of future needs. As such, the complete specification is not implemented within Location Studio – only the basic set of features required for today's LBS applications (Standard Location Immediate Service). Location Studio will maintain compliance with the relevant portions of the LIF MLP specification as it is updated and ratified (currently MLP 3.0).

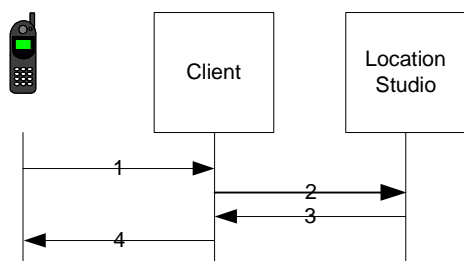
The LIF location requests may be made the standards compliant XML messages to LSt or with the WSDL interface described in this section.

7.2 Simple Location Request

For those applications that only requires basic location information the simple location request offers a streamlined, more efficient and simpler method of obtaining that information. This is enough for most LCS clients.

7.3 Typical Location Request/Response Sequence

A typical request/response sequence for both the LIF and the Simple Location Request would consist of a subscriber requesting some location based content from a client as in the following example:



1. User requests some location content from the client
2. Client issues "getLocation" to LSt.
3. LSt returns the location information to the client
4. Client returns location content to the user.

7.4 Interface Description

Although both request types are made with the same call, for clarity the descriptions are separated into 2 sections.

7.4.1 LIF based request

All of the parameters and return values are compliant with the LIF specification. For detailed description on each of these values please refer back to the LIF specification version 3.0.0[3]. For detailed information on what LIF functionality are supported by LSt, please refer to the Openwave LSt Interface Statement of Compliance for LIF MLP[4].

7.4.1.1 getLocation

Name	getLocation	
Description	Get the location of up to 25 subscribers. For request with multiple subscribers, Location Studio will attempt to retrieve all the locations and return them in a single response. If all locations are not received by location studio within the timeout period, only those locations that have been received will be returned.	
Mandatory Parameters	clientID clientPwd subscriber	String. Identifies the LCS client String. Authenticates the LCS client Subscriber ID object. Type and Identification of the subscriber. Valid types are phone number and operator aliases. For a description of the alias types see section 8.1. One may submit multiple subscribers in a single request for positioning.
Optional Parameters	respReq respTimer llAcc horAcc altAcc maxLocAge locType	String. This attribute represents response time requirement. Values: NO_DELAY LOW_DELAY DELAY_TOL Default: DELAY_TOL. Integer. Defines a timer for the response time within which the current location should be obtained and returned to the LCS Client. Units: seconds Default: 0 Integer. Longitude and latitude accuracy Units: seconds Default: 0 Integer. Requested horizontal accuracy Units: meters Default: 0 Integer. Accuracy of altitude Units: meters Default: 0 Integer. This states the maximum allowable age in seconds of a location sent as a response to a location request Units: seconds Default: 0 String. the type of location requested

Interface Control Document

	prioType	<p>Values: LAST CURRENT CURRENT_OR_LAST</p> <p>Default: CURRENT</p> <p>String. Defines the priority of a location request</p> <p>Values: NORMAL HIGH</p> <p>Default: NORMAL</p>
Return Values	addInfo	String. A text string containing additional information about a certain result
	result	String. A text string indicating the result. Location Studio is compliant with the error messages outlined in the LIF specification[3]
	resid	Integer. a numeric representation of a result message. Location Studio is compliant with the error IDs outlined in the LIF specification[3]
	mLP300Positions[]	
	time	String. the time when the positioning was performed Format: yyyyMMddhhmmss
	x	String. The first ordinate in a coordinate system Format: DD MM SS.SSSH
	y	String. Second ordinate in a coordinate.system. This is optional if it is a linear coordinate system Format: DD MM SS.SSSH
	z	String. third ordinate in a coordinate.system. This is optional if it is a 2D coordinate system Format: meters
	radius	String. The uncertainty radius is the radius of the uncertainty; this is the geodesic distance between the arc and the position point.. Units: meters
	altAcc	String. Accuracy of altitude in meters Units: meters
	alt	String. The altitude of the mobile station. Units: meters
	speed	String. The speed of the MS Units: meters/second
	direction	String. Specifies the direction that a positioned MS is moving in Units: degrees
	result	String. A text string indicating the result of an individual positioning
	resId	String. a numeric representation of a result message
	utcOff	String. Specifies the UTC offset in hours and minutes. Positive values indicate time zones east of Greenwich
	levConf	String. the probability in percent that the MS is located in the position area that is returned
	addInfo	String. A text string containing additional information

Interface Control Document

	subscriber	about a certain result MLP300Msid object. MLP ID Type and Identification of the subscriber. Valid types are phone number and operator aliases. For a description of the alias types see section 8.1.
--	------------	---

7.4.2 Simple Location request

7.4.2.1 getLocation

Name	getLocation	
Description	Get the location of up to 25 subscribers. For request with multiple subscribers, Location Studio will attempt to retrieve all the locations and return them in a single response. If all locations are not received by location studio within the timeout period, only those locations that have been received will be returned.	
Mandatory Parameters	clientID password Timeout maxLocAge subscriber	String. Identifies the LCS client String. Authenticates the LCS client Integer. Maximum time before a response must be returned. Units: seconds Integer. This states the maximum allowable age of a location sent as a response to a location request Units: seconds Subscriber ID object. Type and Identification of the subscriber. Valid types are phone number and operator aliases. For a description of the alias types see section 8.1. One may submit multiple subscribers in a single request for positioning.
Optional Parameters	none	
Return Values	response Code response Detail WLP10Position[] response Code response Detail subscriber latitude	Integer value that maps to a response. See section 10 String. Optional additional text information about the response Integer value that maps to a response. See section 10 String. Optional additional text information about the response Subscriber ID object. Type and Identification of the subscriber. Valid types are phone number and operator aliases. For a description of the alias types see section 8.1. String. The latitude returned Format: DD MM SS.SSSH

Interface Control Document

	longitude	String. The longitude returned Format: DD MM SS.SSSH
	time	String. the time when the positioning was performed Format: yyyyMMddhhmmss
	radius	String. radius of a circle of uncertainty Units: meters

8 APPLICATION NOTES

8.1 Valid LSt ID Types

Identifier	Description	LIF Type(s)	AAI LSt type	Example
Phone number	Either Mobile Identification Number (the mobile phone number in a TIA based network) or Mobile Station ISDN (An MSISDN consists of a country code, a national destination code and a subscriber number. MSISDNs are used in ETSI based networks).	MIN, MSISDN	MSID	13033811000
Operator Alias	This is an alias defined by the operator or other external agency. These could be an login ID, NAI, ESN, WAP ID, email address, etc. Specific valid operator aliases are defined by the serving operator.	IMSI, IMEI, OPE_ID	OSID	192383@gateway.abc.com
Temporary Alias	This is an alias assigned by LSt for a finite period of time.	SESSID	TSID	21654485
Persistent Alias	This is an alias assigned by LSt for an indeterminate period of time.	ASID	PSID	21654485

8.2 Valid Media Types

Location Studio passes through the media type to the WAP gateway. As a result, the LCS client must ensure that both the target handset and the gateway support the proposed media type. The recommended, and default, media type for messaging requests is Service indication (text/vnd.wap.si). For definitions of valid media types please see the wap 2.0 specification.

8.3 WAP proxy

Since the WAP proxy passes CGI parameters in the URL, the LCS client must support the use of HTTP GET to receive the proxied parameters.

The WAP proxy acts a means of verifying implicit permission of active location requests. An active location request is one made by a subscriber to locate themselves. By using the WAP proxy in such a request, LSt can confirm that it is an active request and grant location without having to use WAP ASK. WAP Ask is used to request permission in a passive scenario.

8.4 WAP Ask

The WAP ask feature allows the subscriber to have control of the location information without necessarily having a formal permission relationship with the client. This feature is only relevant for passive location requests. A passive request is made by a third party, requesting the location of a particular subscriber. Active location requests, those made by the subscriber to locate themselves, have implicit authorization and are handled by the WAP proxy feature to ensure that it is an active request.

9 EXAMPLE REQUESTS AND RESPONSES

9.1 Pseudo example

Pseudo code for fetching a persistent alias that may be used in subsequent requests to LSt:

```
initRequest = new WVP10InitRequest(new SubscriberId("MSID", "46730511400"), "Please reply to this
message with %CODE% to become a member of FriendFinder.", "%CODE%");
initResponse = wviValidation.initiateVerifySubscriber("theasp", "thepwd", initRequest);
if (initResponse.getStatus().equals("ok"))
{
    validationSubscriberId = initResponse.getVsid();
}
completionRequest = new WVP10CompletionRequest(validationSubscriberId, "1234");
completionResponse = wviValidation.completeVerifySubscriber("theasp", "thepwd", completionRequest);
if (completionResponse.getStatus().equals("ok"))
{
    persistentSubscriberId = completionResponse.getPsid();
}
```

9.2 Java example

The following Simple AAI demo in JAVA authenticates a client, validates a subscriber and then sends an SMS message to the subscriber.

```
import com.openwavecorp.lst.client.*;
import electric.registry.Registry;
import electric.util.Context;

/**
 * Simple AAI demo. Authenticates a client, validates a subscriber and
 * then sends an SMS message to the subscriber.
 */
public class LStClientDemo {

    public static final void main(String [] args) {
        try {
            // Web service URL
            String url =
"http://localhost:8080/locationstudio/webservices/aai.wsdl";

            String clientId = "theasp";
            String clientPwd = "thepwd";

            // Create a subscriber
            SubscriberId subscriber = new SubscriberId();
            subscriber.setType("MSID");
            subscriber.setId("3033813000");
```

Interface Control Document

```
// Bind to validation service
WVIVValidation validation = (WVIVValidation)Registry.bind(url,
WVIVValidation.class);

// Authenticate client
WVP10AuthenticationResponse authResp =
validation.authenticateClient(clientId, clientPwd);
if(authResp.getResponseCode()!=Constants.OK) {
    throw new Exception("Client authentication failed:
"+authResp.getResponseCode()+" "+authResp.getResponseDetail());
}

// Validate subscriber
WVP10ValidationResponse valResp =
validation.validateSubscription(clientId,clientPwd,subscriber);
if(authResp.getResponseCode()!=Constants.OK) {
    throw new Exception("Subscriber validation failed:
"+valResp.getResponseCode()+" "+valResp.getResponseDetail());
}

// Bind to messaging service
WMIMessaging messaging = (WMIMessaging)Registry.bind(url,
WMIMessaging.class);

// Send an SMS message to the subscriber
WMP10SmsRequest smsRequest = new WMP10SmsRequest("Hello
subscriber",subscriber);
WMP10MessagingResponse msgResp =
messaging.sendSmsMessage(clientId,clientPwd,smsRequest);
if(msgResp.getResponseCode()!=Constants.OK) {
    throw new Exception("Sending SMS failed:
"+msgResp.getResponseCode()+" "+msgResp.getResponseDetail());
}
System.out.println("Message sent");
}
catch(electric.util.WrappedException e) {
    e.getException().printStackTrace();
    e.printStackTrace();
}
catch(Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
}
}
```

10 ERROR MESSAGES

Result ID	Message	Description
-1	UNDEFINED	Undefined state
0	OK	Indicates that an action was successful
10	SYSTEM_FAILURE	A server side system failure occurred
100	CLIENT_AUTHENTICATION_FAILED	Indicates that the provided client does is not authorized to perform the requested action: Value = 100
101	INCORRECT_PASSWORD	The password provided is incorrect
102	INVALID_ARGUMENT	an argument passed to one of the methods of an interface was either out of range, null when null is not allowed, etc
300	CLIENT_NOT_FOUND	The provided client was not found
301	CLIENT_NOT_ENABLED	The client is not enabled.
310	SUBSCRIBER_NOT_FOUND	The provided subscriber was not found
311	INVALID_SUBSCRIBER_ID	An illegal subscriber type or subscriber id was provided
312	UNSUPPORTED_SUBSCRIBER_TYPE	Some operations only support some subscriber types (MSID, PSID, TSID etc), this code indicates that requested action does not support the provided subscriber type
313	MASTER_PRIVACY_DENY_ALL	The request failed because master privacy was enabled for the subscriber.
314	OPERATOR_SERVICE_NOT_ENABLED	The operator has disabled this client.
315	SUBSCRIBER_SERVICE_NOT_ENABLED	The subscriber has disabled access for this client.
316	SUBSCRIBER_TYPE_NOT_ALLOWED	The provided subscriber id type is not allowed.
3001	BILLING_NOT_AUTHORIZED	Response code indicating that billing was not authorized
3002	BILLING_DISABLED	Response code indicating that billing is disabled
4001	FAILED_TO_SEND_SMS	Indicates that the server failed to send the SMS message
4002	SUBSCRIBER_ERRORS	Indicates that errors occurred for one or more of the provided subscribers
4003	MESSAGING_NOT_ALLOWED	Response code indicating that messaging was not allowed
4004	MESSAGING_DISABLED	Response code indicating that messaging is disabled
5000	WRONG_PIN	Wrong PIN supplied
5001	INCORRECT_VERIFICATION_ALIAS	The verification alias supplied is not correct
5003	UNSUPPORTED_SUBSCRIBER_TYPE	The subscriber type supplied is not allowed for the method called
5004	PERMISSION_DOES_NOT_EXIST	A permission for this subscriber does not exists

Interface Control Document

	ST	
5005	PSID_ALREADY_EXISTS	A PSID for this subscriber/client relation does already
5006	SUBSCRIBERSTATUS_INVALID_CHECK	The subscriberValidation failed
5007	SUBSCRIBERSTATUS_TRANSFERRED	The subscription has been transferred
5008	SUBSCRIBERSTATUS_NOFUNDS	The subscription has no funds available
5009	SUBSCRIBERSTATUS_NOTSUBSCRIBER	The subscription does not exists (or does not belong the operator)
5010	SUBSCRIBERSTATUS_AUTHORIZATIONFAILED	The client is not authorized to perform this method
5011	VALIDATION_DISABLED	Response code indicating that validation is disabled
6000	POSITION_METHOD_FAILURE	Location server can not meet the request requirements
6001	CONGESTION_IN_LOCATION_SERVER	Request can not be handled due to congestion in the location server
6002	CONGESTION_IN_MOBILE_NETWORK	Request can not be handled due to congestion in the mobile network
6003	ABSENT_SUBSCRIBER	The subscriber is currently not reachable
6004	TOO_MANY_POSITION_ITEMS	Too many position items have been specified in the request
6005	FORMAT_ERROR	Request contained a format error
6006	SYNTAX_ERROR	Request contained a syntax error
6010	INVALID_PROTOCOL_ELEMENT_VALUE	Request contained an invalid protocol element value
6011	INVALID_PROTOCOL_ELEMENT_ATTRIBUTE	Request contained an invalid protocol element attribute
6012	PROTOCOL_ELEMENT_VALUE_NOT_SUPPORTED	Request contained an unsupported protocol element value
6013	PROTOCOL_ELEMENT_ATTRIBUTE_VALUE_NOT_SUPPORTED	Request contained an unsupported protocol element value
6014	QOP_NOT_ATTAINABLE	The requested Quality of Position could not be provided
6015	POSITIONING_NOT_ALLOWED	Positioning not allowed
6016	DISALLOWED_BY_LOCAL REGULATIONS	Positioning disallowed
6017	MISCONFIGURATION_OF_LOCATION_SERVER	Location server is not properly configured
6018	LOCATION_NOT_ALLOWED	Location not allowed

10.1 Location Request Error Codes

Location Studio's compliance with the LIF MLP 3.0 specification includes compliance with the error codes defined by that specification. As a result, both forms of the getLocation wsd operation described in the "Location Requests" section of this document return error codes that originated from LIF MLP error codes. Specifically, if an error occurs while attempting to satisfy a getLocation request, Location Studio

Interface Control Document

returns a LIF MLP error code, which is then mapped to one of the above AAI error codes. AAI error codes in the range 6000-6999 directly correspond to error codes defined in the LIF MLP specification. However, some of the AAI error codes in the range 0-1000 are not compliant with the LIF MLP specification and therefore cannot be returned by the getLocation requests. Note that the mapping is only done when using the simple location request. If a MLP3.0 request is sent the LIF MLP error code will be returned.

For example, the AAI includes an "INCORRECT_PASSWORD" error code but the LIF MLP specification does not. All authorization failures in the LIF MLP specification are lumped into the LIF MLP "UNAUTHORIZED_APPLICATION" error code. Therefore, the getLocation request can never return an "INCORRECT_PASSWORD" error code. However, the other AAI requests (billing, messaging and validation requests) are not limited to the LIF MLP error codes and can therefore return more detailed error codes defined by the AAI.

The following table shows the mapping from LIF MLP error codes to AAI error codes.

LIF MLP error code	AAI error code
OK(0)	OK(0)
SYSTEM_FAILURE(1)	SYSTEM_FAILURE(10)
UNSPECIFIED_ERROR(2)	UNDEFINED(-1)
UNAUTHORIZED_APPLICATION(3)	CLIENT_AUTHENTICATION_FAILED(100)
UNKNOWN_SUBSCRIBER(4)	SUBSCRIBER_NOT_FOUND(310)
ABSENT_SUBSCRIBER(5)	ABSENT_SUBSCRIBER(6003)
POSITION_METHOD_FAILURE(6)	POSITION_METHOD_FAILURE(6000)
CONGESTION_IN_LOCATION_SERVER(101)	CONGESTION_IN_LOCATION_SERVER(6001)
CONGESTION_IN_MOBILE_NETWORK(102)	CONGESTION_IN_MOBILE_NETWORK(6002)
TOO_MANY_POSITION_ITEMS(104)	TOO_MANY_POSITION_ITEMS(6004)
FORMAT_ERROR(105)	FORMAT_ERROR(6005)
SYNTAX_ERROR(106)	SYNTAX_ERROR(6006)
INVALID_PROTOCOL_ELEMENT_VALUE(110)	INVALID_PROTOCOL_ELEMENT_VALUE(6010)
INVALID_PROTOCOL_ELEMENT_ATTRIBUTE(111)	INVALID_PROTOCOL_ELEMENT_ATTRIBUTE(6011)
PROTOCOL_ELEMENT_VALUE_NOT_SUPPORTED(112)	PROTOCOL_ELEMENT_VALUE_NOT_SUPPORTED(6012)
PROTOCOL_ELEMENT_ATTRIBUTE_VALUE_NOT_SUPPORTED(113)	PROTOCOL_ELEMENT_ATTRIBUTE_VALUE_NOT_SUPPORTED(6013)
QOP_NOT_ATTAINABLE(201)	QOP_NOT_ATTAINABLE(6014)
POSITIONING_NOT_ALLOWED(202)	POSITIONING_NOT_ALLOWED(6015)
DISALLOWED_BY_LOCAL_REGULATIONS(204)	DISALLOWED_BY_LOCAL_REGULATIONS(6016)
MISCONFIGURATION_OF_LOCATION_SERVER(207)	MISCONFIGURATION_OF_LOCATION_SERVER(6017)

Note also that even though the LIF MLP request includes a list of subscribers, it does not include an error code that indicates partial success. In other words, if the getLocation request failed for 1 or more subscribers but succeeded for 1 or more other subscribers, there is no LIF MLP error code to indicate such a condition. Therefore, the developer is responsible for looking at the result code for each and every subscriber in the response list. Do not rely on the overall error code. It may say the request was successful but that does not guarantee that the location of all subscribers is successfully returned.

If you are using the getLocation operation with LIF MLP arguments, then the response contains an array of mLPPositions, one for each subscriber passed in the request. Each mLPPositions element contains resId, result and addInfo elements that tell you whether the location request for the associated subscriber was successful or not. Look at these elements for each subscriber individually. Do not rely on the overall resId, result and addInfo elements.

If you are using the getLocation operation with simplified arguments, then the response contains an array of WLP10Position, one for each subscriber passed in the request. Each WLP10Position element contains reponseCode and responseDetail elements that tell you whether the location request for the

Interface Control Document

associated subscriber was successful or not. Look at these elements for each subscriber individually. Do not rely on the overall responseCode and responseDetail elements.

11 WSDL

11.1 Subscriber Validation

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by GLUE on Thu Aug 08 14:33:42 CEST 2002-->
<definitions name="WVValidationService" targetNamespace="http://www.openwave.com/wsd/WVValidationService/"
  xmlns:tns="http://www.openwave.com/wsd/WVValidationService/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tme="http://www.openwave.com/" xmlns:ns11="http://www.openwave.com/package/com.openwavecorp.lst.client/">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.openwave.com/package/com.openwavecorp.lst.client/">
      <complexType name="WVP10AuthenticationResponse">
        <complexContent>
          <extension base="ns11:Response"/>
        </complexContent>
      </complexType>
      <complexType name="Response">
        <sequence>
          <element name="responseCode" type="int"/>
          <element name="responseDetail" nillable="true" type="string"/>
        </sequence>
      </complexType>
      <complexType name="WVP10ValidationResponse">
        <complexContent>
          <extension base="ns11:Response">
            <sequence>
              <element name="transferredDate" nillable="true" type="dateTime"/>
              <element name="operatorName" nillable="true" type="string"/>
            </sequence>
          </extension>
        </complexContent>
      </complexType>
      <complexType name="WVP10SubscriberId">
        <complexContent>
          <extension base="ns11:SubscriberId"/>
        </complexContent>
      </complexType>
      <complexType name="SubscriberId">
        <sequence>
          <element name="id" nillable="true" type="string"/>
          <element name="type" nillable="true" type="string"/>
        </sequence>
      </complexType>
      <complexType name="WVP10InitResponse">
        <complexContent>
          <extension base="ns11:Response">
            <sequence>
              <element name="verificationAlias" nillable="true" type="string"/>
            </sequence>
          </extension>
        </complexContent>
      </complexType>
      <complexType name="WVP10InitRequest">
        <sequence>
          <element name="subscriber" nillable="true" type="ns11:WVP10SubscriberId"/>
          <element name="message" nillable="true" type="string"/>
          <element name="codeSignature" nillable="true" type="string"/>
          <element name="messageDelimiter" nillable="true" type="string"/>
        </sequence>
      </complexType>
      <complexType name="WVP10CompletionResponse">
        <complexContent>
          <extension base="ns11:Response">
            <sequence>
              <element name="subscriber" nillable="true" type="ns11:WVP10SubscriberId"/>
            </sequence>
          </extension>
        </complexContent>
      </complexType>
      <complexType name="WVP10CompletionRequest">
        <sequence>
          <element name="verificationAlias" nillable="true" type="string"/>
          <element name="code" nillable="true" type="string"/>
        </sequence>
      </complexType>
    </schema>
  </types>
```

Interface Control Document

```
</types>
<message name="authenticateClient0In">
  <part name="clientId" type="xsd:string"/>
  <part name="password" type="xsd:string"/>
</message>
<message name="authenticateClient0Out">
  <part name="Result" type="ns11:WVP10AuthenticationResponse"/>
</message>
<message name="validateSubscription1In">
  <part name="clientId" type="xsd:string"/>
  <part name="password" type="xsd:string"/>
  <part name="subscriber" type="ns11:WVP10SubscriberId"/>
</message>
<message name="validateSubscription1Out">
  <part name="Result" type="ns11:WVP10ValidationResponse"/>
</message>
<message name="initiateVerifySubscriber2In">
  <part name="clientId" type="xsd:string"/>
  <part name="password" type="xsd:string"/>
  <part name="request" type="ns11:WVP10InitRequest"/>
</message>
<message name="initiateVerifySubscriber2Out">
  <part name="Result" type="ns11:WVP10InitResponse"/>
</message>
<message name="completeVerifySubscriber3In">
  <part name="clientId" type="xsd:string"/>
  <part name="password" type="xsd:string"/>
  <part name="request" type="ns11:WVP10CompletionRequest"/>
</message>
<message name="completeVerifySubscriber3Out">
  <part name="Result" type="ns11:WVP10CompletionResponse"/>
</message>
<portType name="WVValidation">
  <operation name="authenticateClient" parameterOrder="clientId password">
    <input name="authenticateClient0In" message="tns:authenticateClient0In"/>
    <output name="authenticateClient0Out" message="tns:authenticateClient0Out"/>
  </operation>
  <operation name="validateSubscription" parameterOrder="clientId password subscriber">
    <input name="validateSubscription1In" message="tns:validateSubscription1In"/>
    <output name="validateSubscription1Out" message="tns:validateSubscription1Out"/>
  </operation>
  <operation name="initiateVerifySubscriber" parameterOrder="clientId password request">
    <input name="initiateVerifySubscriber2In" message="tns:initiateVerifySubscriber2In"/>
    <output name="initiateVerifySubscriber2Out" message="tns:initiateVerifySubscriber2Out"/>
  </operation>
  <operation name="completeVerifySubscriber" parameterOrder="clientId password request">
    <input name="completeVerifySubscriber3In" message="tns:completeVerifySubscriber3In"/>
    <output name="completeVerifySubscriber3Out" message="tns:completeVerifySubscriber3Out"/>
  </operation>
</portType>
<binding name="WVValidation" type="tns:WVValidation">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="authenticateClient">
    <soap:operation soapAction="authenticateClient" style="rpc"/>
    <input name="authenticateClient0In">
      <soap:body use="encoded"/>
    </input>
    <output name="authenticateClient0Out">
      <soap:body use="encoded"/>
    </output>
  </operation>
  <operation name="validateSubscription">
    <soap:operation soapAction="validateSubscription" style="rpc"/>
    <input name="validateSubscription1In">
      <soap:body use="encoded"/>
    </input>
    <output name="validateSubscription1Out">
      <soap:body use="encoded"/>
    </output>
  </operation>
  <operation name="initiateVerifySubscriber">
    <soap:operation soapAction="initiateVerifySubscriber" style="rpc"/>
    <input name="initiateVerifySubscriber2In">
      <soap:body use="encoded"/>
    </input>
    <output name="initiateVerifySubscriber2Out">
      <soap:body use="encoded"/>
    </output>
  </operation>
  <operation name="completeVerifySubscriber">
    <soap:operation soapAction="completeVerifySubscriber" style="rpc"/>
    <input name="completeVerifySubscriber3In">
      <soap:body use="encoded"/>
    </input>
    <output name="completeVerifySubscriber3Out">
      <soap:body use="encoded"/>
    </output>
  </operation>
</binding>
</service>
</definitions>
</schema>
```

Interface Control Document

```

        <soap:body use="encoded"
namespace="http://tempuri.org/com.openwavecorp.products.locationstudio.presentation.validation.WVValidationService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
</operation>
<operation name="completeVerifySubscriber">
    <soap:operation soapAction="completeVerifySubscriber" style="rpc"/>
    <input name="completeVerifySubscriber3In">
        <soap:body use="encoded"
namespace="http://tempuri.org/com.openwavecorp.products.locationstudio.presentation.validation.WVValidationService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output name="completeVerifySubscriber3Out">
        <soap:body use="encoded"
namespace="http://tempuri.org/com.openwavecorp.products.locationstudio.presentation.validation.WVValidationService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
    </operation>
</binding>
<service name="WVValidationService">
    <documentation>Wireless Validation Interface WebService</documentation>
    <port name="WVValidation" binding="tns:WVValidation">
        <soap:address location="http://trinity.swed.openwavecorp.com:8080/locationstudio/webservices/validation"/>
    </port>
</service>
</definitions>
```

11.2 Messaging

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by GLUE on Thu Aug 08 14:33:04 CEST 2002-->
<definitions name="WMIMessagingService" targetNamespace="http://www.openwave.com/wsdl/WMIMessagingService/"
xmlns:tns="http://www.openwave.com/wsdl/WMIMessagingService/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tme="http://www.openwave.com/" xmlns:ns11="http://www.openwave.com/package/com.openwavecorp.lst.client/">
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.openwave.com/package/com.openwavecorp.lst.client/">
            <complexType name="WMP10MessagingResponse">
                <complexContent>
                    <extension base="ns11:Response">
                        <sequence>
                            <element name="subscriberStatus" nillable="true" type="ns11:ArrayOfWMP10SubscriberStatus"/>
                        </sequence>
                    </extension>
                </complexContent>
            </complexType>
            <complexType name="Response">
                <sequence>
                    <element name="responseCode" type="int"/>
                    <element name="responseDetail" nillable="true" type="string"/>
                </sequence>
            </complexType>
            <complexType name="WMP10SubscriberStatus">
                <complexContent>
                    <extension base="ns11:Response">
                        <sequence>
                            <element name="subscriberId" nillable="true" type="ns11:WMP10SubscriberId"/>
                        </sequence>
                    </extension>
                </complexContent>
            </complexType>
            <complexType name="ArrayOfWMP10SubscriberStatus">
                <complexContent>
                    <restriction base="soapenc:Array">
                        <attribute ref="soapenc:arrayType" wsdl:arrayType="ns11:WMP10SubscriberStatus[]"/>
                    </restriction>
                </complexContent>
            </complexType>
            <complexType name="WMP10SubscriberId">
                <complexContent>
                    <extension base="ns11:SubscriberId"/>
                </complexContent>
            </complexType>
            <complexType name="SubscriberId">
                <sequence>
                    <element name="id" nillable="true" type="string"/>
                    <element name="type" nillable="true" type="string"/>
                </sequence>
            </complexType>
            <complexType name="WMP10SmsRequest">
```

Interface Control Document

```
<complexContent>
  <extension base="ns11:WMP10MessagingRequest">
    <sequence>
      <element name="delimiter" nillable="true" type="string"/>
    </sequence>
  </extension>
</complexContent>
</complexType>
<complexType name="WMP10MessagingRequest" abstract="true">
  <sequence>
    <element name="body" nillable="true" type="string"/>
    <element name="recipients" nillable="true" type="ns11:ArrayOfWMP10SubscriberId"/>
  </sequence>
</complexType>
<complexType name="ArrayOfWMP10SubscriberId">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="ns11:WMP10SubscriberId[]" />
    </restriction>
  </complexContent>
</complexType>
<complexType name="WMP10WapRequest">
  <complexContent>
    <extension base="ns11:WMP10MessagingRequest">
      <sequence>
        <element name="mediaType" nillable="true" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
</schema>
</types>
<message name="sendSmsMessage0In">
  <part name="clientId" type="xsd:string"/>
  <part name="password" type="xsd:string"/>
  <part name="request" type="ns11:WMP10SmsRequest"/>
</message>
<message name="sendSmsMessage0Out">
  <part name="Result" type="ns11:WMP10MessagingResponse"/>
</message>
<message name="sendWapMessage1In">
  <part name="clientId" type="xsd:string"/>
  <part name="password" type="xsd:string"/>
  <part name="request" type="ns11:WMP10WapRequest"/>
</message>
<message name="sendWapMessage1Out">
  <part name="Result" type="ns11:WMP10MessagingResponse"/>
</message>
<portType name="WMIMessaging">
  <operation name="sendSmsMessage" parameterOrder="clientId password request">
    <input name="sendSmsMessage0In" message="tns:sendSmsMessage0In"/>
    <output name="sendSmsMessage0Out" message="tns:sendSmsMessage0Out"/>
  </operation>
  <operation name="sendWapMessage" parameterOrder="clientId password request">
    <input name="sendWapMessage1In" message="tns:sendWapMessage1In"/>
    <output name="sendWapMessage1Out" message="tns:sendWapMessage1Out"/>
  </operation>
</portType>
<binding name="WMIMessaging" type="tns:WMIMessaging">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="sendSmsMessage">
    <soap:operation soapAction="sendSmsMessage" style="rpc"/>
    <input name="sendSmsMessage0In">
      <soap:body use="encoded"/>
    </input>
    <output name="sendSmsMessage0Out">
      <soap:body use="encoded"/>
    </output>
  </operation>
  <operation name="sendWapMessage">
    <soap:operation soapAction="sendWapMessage" style="rpc"/>
    <input name="sendWapMessage1In">
      <soap:body use="encoded"/>
    </input>
    <output name="sendWapMessage1Out">
      <soap:body use="encoded"/>
    </output>
  </operation>
</binding>
</service>
</definitions>
</wsdl>
```

Interface Control Document

```
</binding>
<service name="WMIMessagingService">
  <documentation>Wireless Messaging Interface WebService</documentation>
  <port name="WMIMessaging" binding="tns:WMIMessaging">
    <soap:address location="http://trinity.swed.openwavecorp.com:8080/locationstudio/webservices/messaging"/>
  </port>
</service>
</definitions>
```

11.3 Event Billing

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by GLUE on Thu Aug 08 14:34:27 CEST 2002-->
<definitions name="WBIBillingService" targetNamespace="http://www.openwave.com/wsd/WBIBillingService/"
  xmlns:tns="http://www.openwave.com/wsd/WBIBillingService/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tme="http://www.openwave.com/" xmlns:ns11="http://www.openwave.com/package/com.openwavecorp.lst.client/">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.openwave.com/package/com.openwavecorp.lst.client/">
      <complexType name="WBP10BillingResponse">
        <complexContent>
          <extension base="ns11:Response"/>
        </complexContent>
      </complexType>
      <complexType name="Response">
        <sequence>
          <element name="responseCode" type="int"/>
          <element name="responseDetail" nillable="true" type="string"/>
        </sequence>
      </complexType>
      <complexType name="WBP10BillingRequest">
        <sequence>
          <element name="eventId" nillable="true" type="string"/>
          <element name="subscriber" nillable="true" type="ns11:WBP10SubscriberId"/>
          <element name="startTime" nillable="true" type="dateTime"/>
          <element name="endTime" nillable="true" type="dateTime"/>
          <element name="trackingId" nillable="true" type="string"/>
        </sequence>
      </complexType>
      <complexType name="WBP10SubscriberId">
        <complexContent>
          <extension base="ns11:SubscriberId"/>
        </complexContent>
      </complexType>
      <complexType name="SubscriberId">
        <sequence>
          <element name="id" nillable="true" type="string"/>
          <element name="type" nillable="true" type="string"/>
        </sequence>
      </complexType>
    </schema>
  </types>
  <message name="sendEvent0In">
    <part name="clientId" type="xsd:string"/>
    <part name="password" type="xsd:string"/>
    <part name="wbpRequest" type="ns11:WBP10BillingRequest"/>
  </message>
  <message name="sendEvent0Out">
    <part name="Result" type="ns11:WBP10BillingResponse"/>
  </message>
  <portType name="WBIBilling">
    <operation name="sendEvent" parameterOrder="clientId password wbpRequest">
      <input name="sendEvent0In" message="tns:sendEvent0In"/>
      <output name="sendEvent0Out" message="tns:sendEvent0Out"/>
    </operation>
  </portType>
  <binding name="WBIBilling" type="tns:WBIBilling">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sendEvent">
      <soap:operation soapAction="sendEvent" style="rpc"/>
      <input name="sendEvent0In">
        <soap:body use="encoded" namespace="http://tempuri.org/com.openwavecorp.products.locationstudio.presentation.billing.WBIBillingService"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </input>
      <output name="sendEvent0Out">
        <soap:body use="encoded" namespace="http://tempuri.org/com.openwavecorp.products.locationstudio.presentation.billing.WBIBillingService"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </output>
    </operation>
  </binding>
</service>
<documentation>Wireless Billing Interface WebService</documentation>
```


Interface Control Document

```
<port name="WBIBilling" binding="tns:WBIBilling">
  <soap:address location="http://trinity.swed.openwavecorp.com:8080/locationstudio/webservices/billing"/>
</port>
</service>
</definitions>
```

11.4 Location

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by GLUE on Thu Jun 27 14:52:18 CEST 2002-->
<definitions name="WLLocationService" targetNamespace="http://www.themindelectric.com/wsdl/WLLocationService/"
  xmlns:tns="http://www.themindelectric.com/wsdl/WLLocationService/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tme="http://www.themindelectric.com/" xmlns:ns11="http://www.themindelectric.com/package/com.openwavecorp.lst.client/">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.themindelectric.com/package/com.openwavecorp.lst.client/">
      <complexType name="MLP300LocationResponse">
        <sequence>
          <element name="addInfo" nillable="true" type="string"/>
          <element name="result" nillable="true" type="string"/>
          <element name="resId" nillable="true" type="string"/>
          <element name="mLP300Positions" nillable="true" type="ns11:ArrayOfMLP300Position"/>
        </sequence>
      </complexType>
      <complexType name="MLP300Position">
        <sequence>
          <element name="time" nillable="true" type="string"/>
          <element name="x" nillable="true" type="string"/>
          <element name="y" nillable="true" type="string"/>
          <element name="z" nillable="true" type="string"/>
          <element name="radius" nillable="true" type="string"/>
          <element name="altAcc" nillable="true" type="string"/>
          <element name="alt" nillable="true" type="string"/>
          <element name="speed" nillable="true" type="string"/>
          <element name="direction" nillable="true" type="string"/>
          <element name="result" nillable="true" type="string"/>
          <element name="resId" nillable="true" type="string"/>
          <element name="utcOff" nillable="true" type="string"/>
          <element name="levConf" nillable="true" type="string"/>
          <element name="addInfo" nillable="true" type="string"/>
          <element name="mLP300Msid" nillable="true" type="ns11:MLP300Msid"/>
        </sequence>
      </complexType>
      <complexType name="ArrayOfMLP300Position">
        <complexContent>
          <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="ns11:MLP300Position[]" />
          </restriction>
        </complexContent>
      </complexType>
      <complexType name="MLP300Msid">
        <sequence>
          <element name="msid" nillable="true" type="string"/>
          <element name="msidType" nillable="true" type="string"/>
        </sequence>
      </complexType>
      <complexType name="MLP300LocationRequest">
        <sequence>
          <element name="clientId" nillable="true" type="string"/>
          <element name="clientPwd" nillable="true" type="string"/>
          <element name="respReq" nillable="true" type="string"/>
          <element name="respTimer" nillable="true" type="string"/>
          <element name="ilAcc" type="int"/>
          <element name="horAcc" type="int"/>
          <element name="altAcc" type="int"/>
          <element name="maxLocAge" type="int"/>
          <element name="locType" nillable="true" type="string"/>
          <element name="prioType" nillable="true" type="string"/>
          <element name="mLP300Msid" nillable="true" type="ns11:ArrayOfMLP300Msid"/>
        </sequence>
      </complexType>
      <complexType name="ArrayOfMLP300Msid">
        <complexContent>
          <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="ns11:MLP300Msid[]" />
          </restriction>
        </complexContent>
      </complexType>
    </schema>
  </types>
```

Interface Control Document

```
<message name="getLocationIn">
  <part name="mLP300LocationRequest" type="ns11:MLP300LocationRequest"/>
</message>
<message name="getLocationOut">
  <part name="Result" type="ns11:MLP300LocationResponse"/>
</message>
<portType name="WLILocationService">
  <operation name="getLocation" parameterOrder="mLP300LocationRequest">
    <input name="getLocationIn" message="tns:getLocationIn"/>
    <output name="getLocationOut" message="tns:getLocationOut"/>
  </operation>
</portType>
<binding name="WLILocationService" type="tns:WLILocationService">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getLocation">
    <soap:operation soapAction="getLocation" style="rpc"/>
    <input name="getLocationIn">
      <soap:body use="encoded"
namespace="http://tempuri.org/com.openwavecorp.products.locationstudio.presentation.location.WLILocationService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output name="getLocationOut">
      <soap:body use="encoded"
namespace="http://tempuri.org/com.openwavecorp.products.locationstudio.presentation.location.WLILocationService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>
</binding>
<service name="WLILocationService">
  <documentation>com.openwavecorp.products.locationstudio.presentation.location.WLILocationService web service</documentation>
  <port name="WLILocationService" binding="tns:WLILocationService">
    <soap:address location="http://192.168.12.136:8080/locationstudio/webservices/location"/>
  </port>
</service>
</definitions>
```