



LSt 2.0 AAI Developer's Guide

Location Studio

Legal Notice

Copyright © 1999–2003 Openwave Systems Inc. All rights reserved.

The contents of this document constitute valuable proprietary and confidential property of Openwave Systems Inc. and are provided subject to specific obligations of confidentiality set forth in one or more binding legal agreements. Any use of this material is limited strictly to the uses specifically authorized in the applicable license agreement(s) pursuant to which such material has been furnished. Any use or disclosure of all or any part of this material not specifically authorized in writing by Openwave Systems Inc. is strictly prohibited.

Openwave, the Openwave logo, and Services OS are registered trademarks and/or trademarks of Openwave Systems Inc. in various jurisdictions. All other trademarks are the property of their respective owners.

For technical information on Openwave products, go to

<http://support.openwave.com>

Part Number LSDG-20a-001

Contents

About This Book	5
Audience	5
How to Use This Book	5
Conventions in This Guide	6
Related Documentation	6
Introduction	8
Location-Based Services Overview	9
Positioning Methods	9
Cell-based Positioning	9
Enhanced Positioning Methods	10
EOTD Positioning (Network-based)	11
AGPS Positioning (Handset-based)	11
Shapes Representing a Geographical Position	11
Ellipsoid Point	12
Point Described Using Geodetic Longitude and Latitude:	12
Ellipsoid Point with Uncertainty Circle	12
Location Studio Overview	13
LSt Services	13
System Components	14
Privacy & Access Management	16
Client Authentication	18
Client Profile and Authorization	18
Personalization	18
Subscriber Profile	18
Subscriber-to-Client Permission	19
Explicit and Implicit Permissions	19
Time-of-Day and Day-of-Week Filter	21
ID Protection Service	21
Client Interfaces	22
Advanced Application Interface (AAI)	22
Proxy Services	22
WAP	23
SMS	24
Notification Service	24
Sample Call Flows	26
SMS Call Flow	26
WAP Call Flow	28
Developing a Location-Intelligent Application	30
Method A—Using Openwave Classes	30
Step 1, Method A. Download the Openwave Location Studio SDK Package	30
Step 2, Method A. Write Your Client Code and Edit the Example Clients	31
Step 3, Method A. Compile Your Client and the Example Clients	32
Step 4, Method A. Run Your Client or the Example Clients	33
Step 5, Method A. Look for Expected Results	33
Method B—Generating Classes From WSDL	34
Step 1, Method B. Download the Openwave LSt SDK Package	34
Step 2, Method B. Download Glue	34

Step 3, Method B. Generate Client-Side Java Classes.....	35
Step 4, Method B. Write Your Client and Edit the Example Clients	35
Step 5, Method B. Compile Your Client and the Example Clients	36
Step 6, Method B. Run Your Client or the Example Clients	37
Step 7, Method B. Look for Expected Results.....	38
Method C.....	38
Step 1, Method C. Generate Client-Side Java or C++ Classes	38
Step 2, Method C. Write Your Client	38
Step 3, Method C. Compile and Run Your Client	38
Receiving Mobile-Originated Messages.....	39
The Openwave Demo Host	40
The LSt 2.0 SDK Download Package.....	41
Location Studio SDK	41
LST AAI ICD.....	41
MethodA and MethodB Directories	41
MethodB Directory	42
Negotiating an Agreement with a Mobile Operator.....	44
Troubleshooting	45
Where to Find More Information About Errors	45
Where to Find More Information About Glue Errors	45
Use the Example Client Applications for Debugging	45
Bad Data Values.....	45
Getting More Diagnostics from Glue.....	46
Glue wsdl2java and Registry.bind.....	46
WSDL URL vs. Connection URL.....	47
SSL–Secure Socket Layer	47
Glossary	48

About This Book

This guide describes the functions you need to perform when administering the Location Studio (LSt) application.

Audience

This guide is intended for use by personnel who need to develop location-intelligent applications to communicate with LSt using the Advanced Application Interface (AAI). To use this book profitably, you need the following experience:

- Experience with the LSt host platform being used
- UNIX
- Web Services and WSDL
- Java
- eXtensible Mark-up Language (XML)
- Wireless communications

How to Use This Book

This book provides conceptual and procedural information to develop client applications for use with LSt through AAI.

Table 1 Section descriptions for this book

Section	Description
Introduction	An overview of the intent and purpose of this document.
Location-Based Services Overview	An introduction to Location Based Services (LBS).
Location Studio Overview	An introduction to LSt.
Developing a Location-Intelligent Application	Describes two methods for developing a client application for use with LSt through AAI.
Receiving Mobile-Originated Messages	This section gives an example of how to receive a mobile-originated message in your client application.
The Openwave Demo Host	Describes how to access Openwave's demonstration host system for LSt client applications.
LSt. 2.0 SDK Website	Describes the files located on the LSt 2.0 SDK Website. Descriptions for each file are given.
Negotiating an Agreement with a Mobile Operator	Provides suggestions for arranging for a carrier to provide support for your application.

Troubleshooting	Information for performing basic troubleshooting on your client application.
Glossary	Describes terminology introduced in this book.

Conventions in This Guide

These conventions are observed in Openwave documentation.

- The Note format provides background information that is tangential to the primary discussion. Here is an example:

NOTE An alternative way to color a picture object is to call `getColor` and then call `fillPix`. Ordinarily, you simply call `colorPix`.

- The Important format draws the reader's attention to key considerations. Here is an example:

IMPORTANT You cannot upgrade directly from Release 4.0 of the software to Release 6.0. If you are running Openwave Hello 4.0, contact the Technical Assistance Center for special upgrade instructions.

- Monospace distinguishes programming elements (such as code fragments, objects, methods, parameters, utilities, configuration keys, and HTML tags) and system elements (such as file names, directories, paths, and URLs). Hanging indentation represents a line continuation.

```
CLASSPATH =classes:$GLIB/GLUE-STD.jar:$GLIB/jnet.jar:$GLIB/jsse.jar:
           $GLIB/servlet.jar:$GLIB/xerces.jar
```

- *Monospace italic* is used for placeholders, which are general names that you replace with names specific to your site, as in this example:

```
your_home_directory\dbadmin\dbupgrade\
```

Related Documentation

The following documents provide additional information necessary to use LSt:

- [1] W3C website on WSDL, <http://www.w3.org/TR/wsdl>
- [2] Glue User's Guide found at The Mind Electric website, <http://www.themindelectric.com/docs/glue/guide/index.html>
- [3] Sun's "Getting Started with Web Services" found at the Sun One website, <http://www.themindelectric.com/docs/glue/guide/index.html>
- [4] O'Reilly's "Web Services Essentials" by Ethan Cerami
- [5] "The Java Web Services Tutorial, The Java Series Enterprise Edition" by Armstrong, Bodoff, Carson, Fisher, Green and Haase, Sun Microsystems and Addison Wesley
- [6] Location Studio Release Notes provides information about new features, corrections, known problems, and last-minute information not included in other LSt documentation.
- [7] Location Studio Administrator Guide provides technical information on how LSt works and how to administer LSt, such as provisioning and using logs and SNMP.
- [8] Location Studio Provisioning Guide provides information on how to use the Location Studio Web Provisioning interface to provision clients and subscribers.

The following documents are available for download at the Openwave Developer Location Studio developer website http://developer.openwave.com/prod_tech/locationstudio.html.

- [9] Advanced Application Interface (AAI) Specification, System Version: Location Studio 2.0, Openwave Systems, 4 October 2002.
- [10] Openwave, "Location Studio White Paper," Version 1.0, June 2002.
- [11] Openwave, "Location Studio 1.0—Frequently Asked Questions," Version 1.4, August 12, 2002.
- [12] LIF, "Mobile Location Protocol Specification Version 3.0.0", June 3, 2002.
- [13] Openwave, "LIF MLP 3.0.0 Interface Statement of Compliance."

Introduction

This document is intended to lead a developer step-by-step through the process of compiling and running a client application that uses the Location Studio 2.0 Advanced Application Interface. This document also provides brief introductions to Location Based Services and Location Studio. Openwave provides this document for members of the Openwave Developers Program who intend to enable an existing application to use location data or to create a new location-based service.

As stated in the LSt Advanced Application Interface (AAI) Specification, the AAI is implemented over Web Services to provide maximum flexibility and ease of implementation for developers. The AAI Specification gives a high level view of the interface, shows where Location Studio Web Services fit in the wireless operator's network, includes example call flows and provides method and parameter definitions in the context of the business application. This document does not attempt to design an application or to repeat the information already documented in the AAI Specification.

This document only refers to the base functionality of Location Studio and does not include information about customized interfaces or functionality. To learn about customized interfaces and features, consult your mobile operator.

The interface specifications of Location Studio are subject to change. All changes will be available from the Openwave Developer website as soon as they are commercially released to the field. Participants in the Location Studio Beta programs may get early access to new releases. The latest files and information for the SDK are maintained at http://developer.openwave.com/prod_tech/locationstudio.html.

Location Studio is licensed to mobile operators by Openwave. Please consult your mobile operator to see whether Location Studio is available to you. 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 not expose all of the SDK functionality.

Location-Based Services Overview

By definition, a Mobile Location Service uses the location of a mobile subscriber (or mobile station) to enable or enhance the subscriber experience within an application. Location-based services have been around for decades—for example: fleet-tracking services facilitated by mobile data networks and GPS terminals—but the penetration of these services into the consumer market has been very limited. The emergence of cellular network-based location determination systems has propelled mobile location services into the mainstream, by making it easy and convenient to position standard mobile telephones.

Increasing numbers of mobile operators are implementing the capability to locate mobile phones in their wireless networks. Several different methods to accomplish the determination of the location of a phone exist. The most common and well-known positioning methods are briefly described in Positioning Methods. For additional information, see the Openwave document, “Location Technologies Presentation.”

Positioning Methods

Different methods have been developed to enable mobile operators to estimate the horizontal position of mobile subscribers. This section describes two of the most prominent methods: cell-based positioning and enhanced positioning.

Cell-based Positioning

Cell-ID provides a location coordinate based on the cell the subscriber is within. The location infrastructure contains information on the location coordinate of each cell centroid. This cell centroid location information is the information that is returned. Cell-ID operates in all networks—that is, GSM, TDMA, CDMA, and AMPS. It is the most common form of positioning.

Figure 1 Urban cell organization



Since the Mobile Station can be anywhere within the cell, or sector, the accuracy of this method depends on the cell size. Positioning is generally more accurate in urban areas with a dense network of smaller cells. Rural areas have a lower density of base stations. If micro cells are used, the cell-size may be reduced significantly—to the range of several hundred meters.

Figure 2 Cell-based positioning

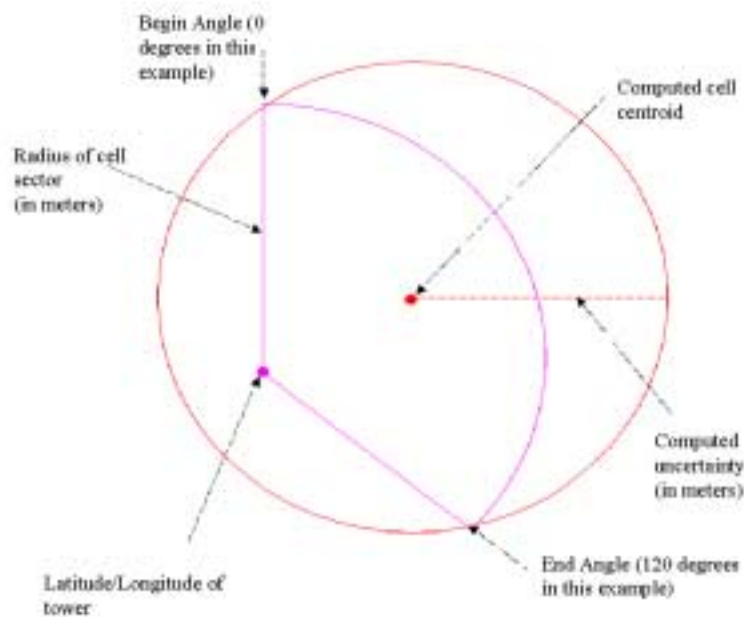


Table 2 contains typical accuracy values measured for cell-ID based positioning.

Table 2 Accuracy for cell-ID positioning

Geography	Typical CellID accuracy
Urban	100 – 400 meters
Suburban	400 – 2000 meters
Rural	1000 – 20000 meters

Enhanced Positioning Methods

High accuracy positioning methods, which provide a significant improvement over cell-based positioning, are based on triangulation (network-based), or with Assisted GPS systems (handset-based). These systems are much more expensive to deploy, and slower to deliver location. This currently limits the number of deployments of this technology, and the frequency with which high-accuracy locations may be requested. Operators generally charge a premium for high-accuracy locations, which is why it is important to only request the accuracy required for your specific application.

Table 3 contains typical accuracy values measured for enhanced positioning methods.

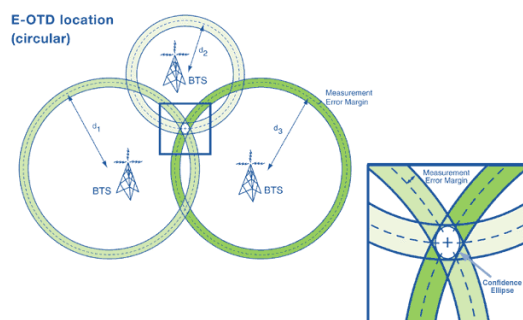
Table 3 Accuracy for enhanced positioning methods

Positioning Method	Typical Accuracy
EOTD	50 – 200 meters
AGPS	5 – 50 meters

EOTD Positioning (Network-based)

Enhanced Observed Time Difference (EOTD) of arrival systems employ handset measured time differences between the time of arrival of signals from surrounding base stations. These time differences are reported to a Serving Mobile Location Center (SMLC). The SMLC also receives reference measurements from Location Measurement Units (LMUs) at base station centers and calculates the coordinates of the phone based on these measurements and the knowledge of the location of the base stations. EOTD was developed for GSM networks.

Figure 3 EOTD positioning



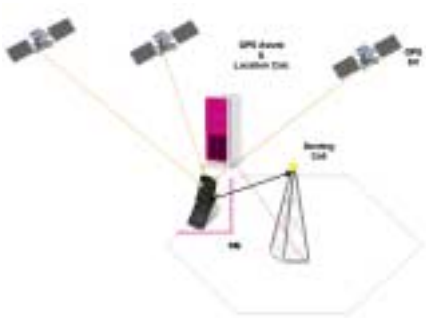
AGPS Positioning (Handset-based)

There are two basic approaches to AGPS: Snaptrack and SiRF.

The Snaptrack approach calls for a central server. The central server stays in contact with a few reference GPS receivers and also calculates the coordinate for a phone based on measurement data from the phone. This method is supported by Qualcomm chip sets for CDMAONE technology.

The SiRF approach (adopted by Motorola, Ericsson, and Nokia) calls for Location Measurement Units (LMUs) to be deployed at most base stations (for optimum performance). Coordinates are calculated in the phone based on GPS measurements and data received from the phone.

Figure 4 AGPS Positioning (Handset-based)



Shapes Representing a Geographical Position

There are a number of shapes used to represent a geographic area that describes where a mobile subscriber is located.

In the future, these shapes may also be used for defining triggering criteria to initiate a positioning when the mobile subscriber enters or leaves the geographical area that is described.

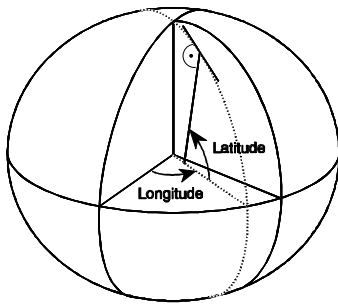
Ellipsoid Point

An ellipsoid point is a point on the surface of the ellipsoid. In practice, such a description can be used to refer to a point on the Earth's surface or close to Earth's surface. The point can be described using a number of co-ordinate systems.

Point Described Using Geodetic Longitude and Latitude:

Figure 5 illustrates a point on the surface of the ellipsoid and its co-ordinates. The latitude is the angle between the equatorial plane and the plane that is perpendicular to the tangent of the point at the ellipsoid surface. Positive latitudes correspond to the North hemisphere. The longitude is the angle between the half-plane determined by the Greenwich meridian and the half-plane defined by the point and the polar axis, measured eastward.

Figure 5 A point on the surface of the ellipsoid

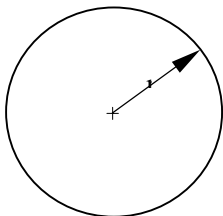


Ellipsoid Point with Uncertainty Circle

An ellipsoid point with uncertainty circle is characterized by the coordinates of an ellipsoid point (the origin) and a radius, "r." The ellipsoid point with uncertainty circle describes the set of points on the ellipsoid that are at a distance from the point of origin less than or equal to "r." This shape can be used to indicate points on the Earth surface or near the Earth surface. The typical use of this shape is to indicate a point when its position is known only with a limited accuracy.

It should be noted that a circle can be represented by an ellipse (see Figure 6) where the semi-major and semi-minor axes are both equal to r and the angle of orientation is immaterial.

Figure 6 A point on the surface of the ellipsoid

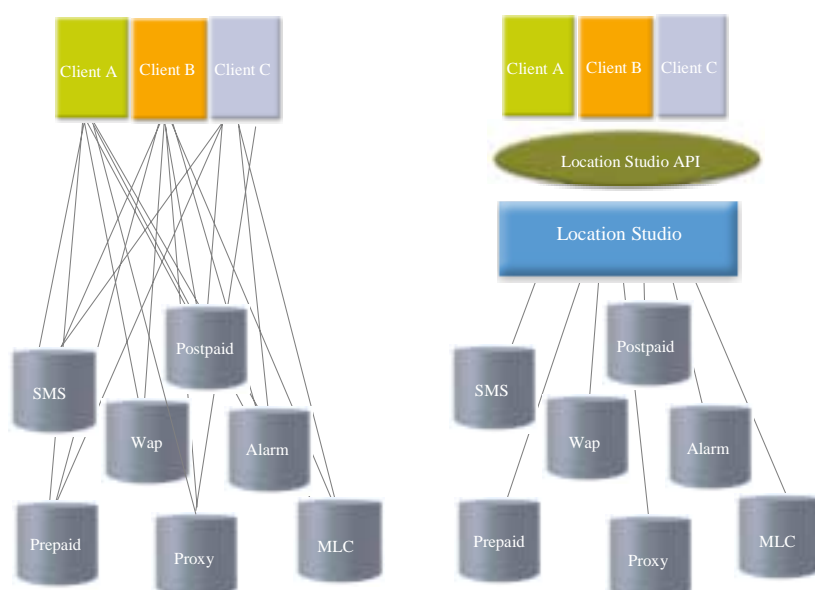


Location Studio Overview

Location Studio is a location middleware product deployed in a wireless operator's network. Location Studio implements access, security, and privacy rules on behalf of subscribers for each transaction between a location-based client application and LSt. A significant feature of LSt is that it enables subscribers to control which clients (location-based services) have access to their location. Privacy controls allow subscribers to control how, when, and under which circumstances client requests for their location are allowed. LSt can also be used as a proxy so that subscriber MSISDN/MIN IDs are not communicated to clients.

Location Studio makes it easier for you to integrate your application with internal operator infrastructure, including: location server, SMSC, WAP gateway, WAP push proxy, subscriber portal, customer care/activation system, billing systems, and operational support systems. Location Studio integrates with these elements once, so that individual LCS clients do not have to do so, as illustrated in Figure 7.

Figure 7 Deployment of a Location Service, without and with LSt



Location Studio is deployed commercially in both GSM and CDMA networks around the world.

LSt Services

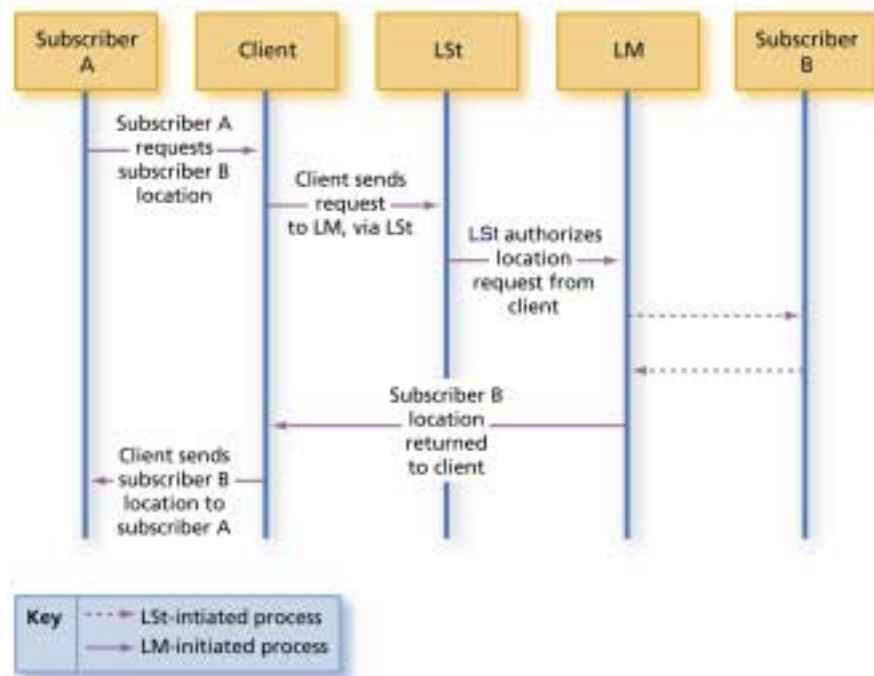
LSt provides the following functionality to subscribers, developers, and network operators:

- In networks with LSt, subscribers can choose the location-based services (LBS) clients that can request their location data. LSt offers filter options that enable subscribers to control how, where, and when their location data is shared with clients. The authorization features of LSt ensure that only authorized clients can request subscriber location data.
- For developers, LSt offers secure connections (SSL/HTTPS) to access a subscriber's mobile terminal location data from an operator's network.
- LSt enables network operators to define the location-based services (LBS) clients that subscribers can use. Operators can also define which clients are able to request subscriber location data. LSt also provides extensive logging and billing capabilities.

LSt handles requests for subscriber location from location-based clients with robust authentication and authorization capabilities that exceed GSM standards.

Following is a basic example of how LSt is used. Using a person finder client, such as Openwave's FriendFinder, subscriber A requests to find subscriber B.

Figure 8 LSt example

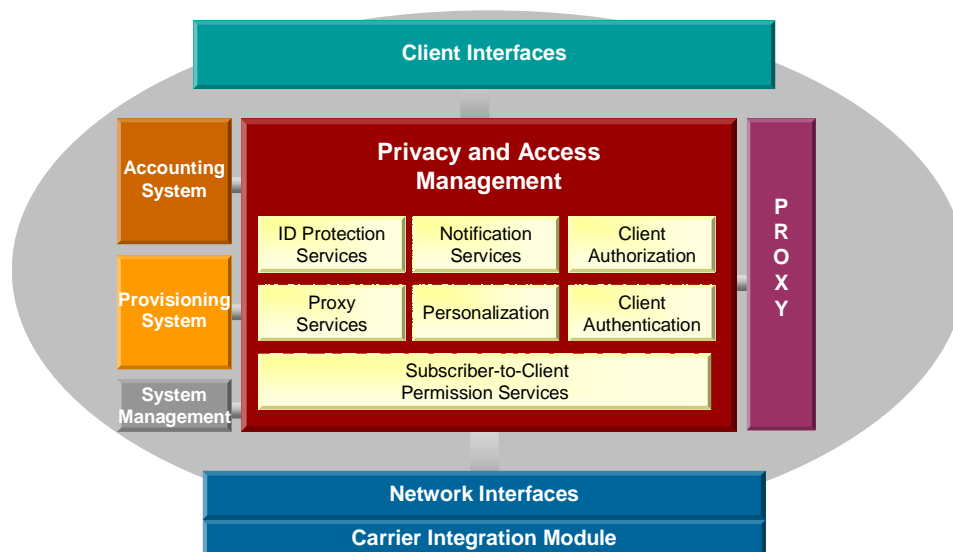


1. The person finder application (the client), receives subscriber A's request and requests subscriber B's location data from Location Manager (LM) via LSt.
2. LSt authorizes subscriber B's location data be returned to the client and the location request is sent to LM. Location Manager then is able to send subscriber B's location data to the client.
3. Lastly, the client sends subscriber B's location to subscriber A.

System Components

Figure 9 shows the functional composition of LSt:

Figure 9 Functional components of LSt

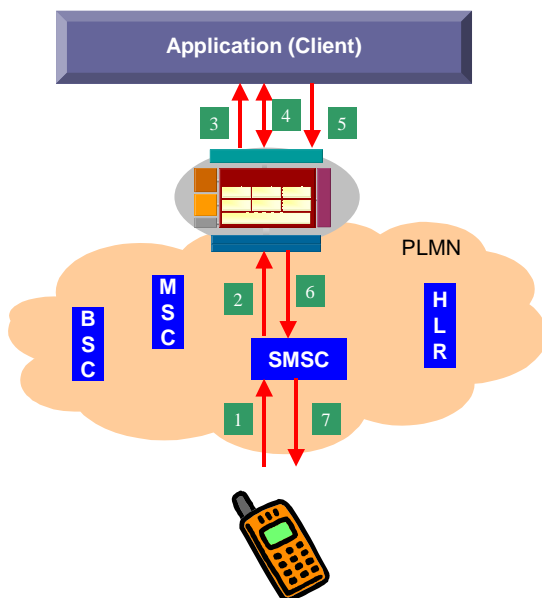


- **Privacy & Access Management:** The core value-added functionality provided by Location Studio. Privacy management at the middleware layer provides a consistent mechanism to control access to location on an application-by-application basis. The subscriber can personalize their privacy profile to allow different privacy and notification setting depending on the application. For the operator, LSt provides a single point of access to the network by LCS clients using privileges stored in a client-profile.
- **Client Interfaces:** Web-Services based interfaces providing access to core functionality required to implement location-based services. LCS client (application) interfaces are provided for the following services: Location Request, Messaging (e.g. SMS), Subscription Validation and Service Billing.
- **Proxy:** An SMS and WAP proxy that facilitates ID protection, single sign-on and stricter authorization on the client interfaces.
- **Accounting System:** Location Studio provides detailed accounting of all micro-transactions executed against the platform by LCS clients. Micro-transactions are defined as basic service requests - location, messaging and spatial/content toolkit services. Each client transaction is stored in a log file, formatted as a Transaction Detail record (TDR) that uniquely identifies the originating client, the date/time, the type of transaction, and the result. Log files are rolled off the system at periodic intervals and transferred to an external mediation system for statistical analysis and/or billing and reconciliation. Micro-transactions can also be correlated against service level billing events submitted by clients to identify a macro-transaction—one that is billable directly to the subscriber (such as “delivered premium content,” or “located three friends”).
- **Provisioning System:** The provisioning system of LSt provides interconnection to the operator’s customer activation and support systems. A combination of Web-based graphical user interfaces and a Web Services based provisioning API allow both manual and electronic modifications to subscriber and client profiles. Additional features support more advanced self-provisioning functions allowing the subscriber more direct control, and lessening the requirements for human interaction with operator support personnel.

- **Network Interfaces:** Consists of two parts: Upper layer and Carrier Integration Module (CIM). The CIM interfaces to value-added services infrastructure within the operator's network, including: SMSC, WAP-GW and Location Server.

Figure 10 illustrates a typical use case of LSt. In this example the subscriber would like to use a location based service through SMS (for example, “yellow pages”). The service is invoked by sending an SMS message with a service specific keyword (for example, “hotel”) to a short number. That short number is defined by the operator and configured in LSt (step 1).

Figure 10 Typical use case



The SMS message is received by the SMSC and forwarded to LSt (step 2).

Location Studio does not analyze the message, but based on the short number used, the message is converted into a http request and forwarded to the appropriate application (step 3)

For the application to deliver a location-based service the location of the subscriber needs to be resolved. The application will request the location from LSt. Before LSt forwards the location request to the operators Location Server (e.g. Openwave Location Manager or Ericsson MPS), the subscribers' privacy settings are checked. Location Studio will reject the location request if the privacy check was negative (step 4).

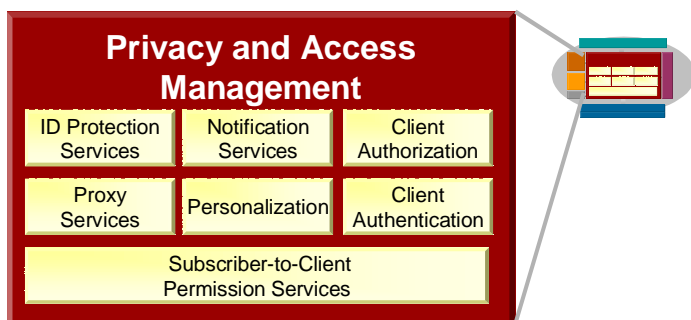
With the subscribers' location, the application is now able to deliver the location-based service. This is done by using the LSt messaging interface, which translates the message from the application to an SMS message and sends it directly to the subscriber (for example, “Closest Hotel is: Royal Viking, 1234 Mainstreet, telephone 7623615”).

Privacy & Access Management

The Privacy and Access features of LSt are the primary value-added feature provided by location middleware. These features provide the controls and safeguards necessary to ensure anonymity and privacy of the subscriber—a function critical to the adoption of location-based services.

There are a number mechanisms used in LSt to protect the subscribers privacy, outlined in the following and detailed in the remainder of this section.

Figure 11 Overview of Privacy and Access Management



ID Protection. To protect the anonymity of the subscriber the interfaces to LSt primarily use identifiers other than the MSISDN. These identifiers include the Temporary Subscriber ID (TSID), Persistent Subscriber ID (PSID) and Operator Subscriber ID (OSID). Of integral importance is the proxy functionality, which replaces the MSISDN or other incoming identifier with one or more of the above aliases before requests are forwarded to the clients.

Personalization. The subscribers can change many settings that are relevant for themselves and their relation to client applications either through WEB, WAP or SMS interfaces.

Client Authentication and Authorization. The client requests need to be authenticated and authorized before acted upon. The authorization is mainly focused on what subscribers, what information and what interfaces the client is allowed to access.

Notification. For the subscriber to feel that he is in control of his location information, a number of notification options can be configured, including “ask features” where the user explicitly must allow clients to make location requests by replying to a notification message.

The basic steps in a successful location request transaction against LSt are:

1. Location Request: Location is requested on any of the supported interfaces.
2. Password check: The client application is checked for a valid user id and password based on the authentication values in the client profile. The client application user ID and password must be present for each request.
3. Enabled account check: Checks the value of the enabled field in the client profile.
4. Client authorized for interface: Check if the client is authorized to make requests on the location request interface.
5. Client Privacy Bypass: Some clients such as the emergency services, lawful intercept, or internal applications have the right to override the subscriber privacy settings. If this is enabled in the client profile, the location request is made without further privacy checks.
6. Subscriber ID type allowed: Checks whether the client is allowed to use the type of subscriber IDs (PSID, TSID, OSID, or MSID) that are present in the request.
7. Priority allowed: verification that the priority the LCS client requests are allowed to be requested. If the LCS client tries a request with higher priority than allowed, an error message will be returned.
8. Master Subscriber Privacy: The operator or subscriber may at some time desire to be location invisible (deny all requests at this point) or visible (allow all requests at this point). If this parameter is not set, the explicit and default permissions are active. An explicit permission is a permission that

the subscriber has set himself, and default permission is defined on a client group level, and is used when the subscriber has not set an explicit one.

9. Is the LCS client Operator Enabled: Provides the ability to restrict the clients that can locate the subscriber. The client - subscriber permission set should have the operator enabled for the requested subscriber for this check to pass.
10. Is the LCS client Subscriber Enabled: This checks to see if the subscriber has added this client to the client-subscriber permission set. The client - subscriber permission set should have the subscriber enabled for the requested subscriber for this check to pass.
11. Time filters. The subscriber can restrict clients to only locate him during certain hours and/or days
12. Apply client profile: Some clients have restricted parameters on the request interface (for example, allowed request accuracy). These clients will have the restrictions applied at this point.
13. Notify subscriber: Check notification options for the subscriber. If the subscriber has the "notify with ask" feature enabled, the client may only proceed with the location request if the subscriber explicitly has allowed the client to locate him for the specific request.

The following components of LSt provide these critical features:

Client Authentication

All transactions against LSt require positive authentication of the requestor's credentials. A login ID and password must be embedded with each request. More advanced authentication may be incorporated using external third-party products—for example bi-directional authentication using SSL with client and server certificates. These advanced capabilities are not part of LSt—they are accessories to enhance the deployment.

Client Profile and Authorization

Once a client has successfully authenticated to LSt their request is authorized based on privileges established within the client profile. The client profile is a set of parameters used for identification and authorization of access by LCS clients. The client profile contains parameters that define what services a client may access from the network, the allowed quality of service, as well as physical identification data used for authentication and billing. The client profile is stored within LSt. Each client may have different settings established, depending on their business relationship or application type.

Personalization

A combination of a Web-based GUI and SMS and WAP interfaces that enable subscribers to personalize their services. Subscribers can typically edit their own settings using the subscriber profile and subscriber-to-client permissions

Subscriber Profile

The subscriber profile is a set of parameters used for identification, authorization, and privacy preferences of subscribers within the serving operator's network. The subscriber profile contains parameters that define what services the subscriber is allowed to access, and what services the subscriber authorizes to receive location. Permissions within this database are unique to specific subscriber-client relationships. The subscriber profile is stored in a database managed by LSt.

Subscriber-to-Client Permission

Subscriber-to-client permissions provide the ability to grant specific client applications unique authority to locate a certain subscriber. Newly provisioned clients are assigned a set of default privacy conditions, which then may be personalized by the subscriber according to their wishes for that specific service. Permissions can be uniquely assigned to each client application enabled in the subscriber's profile.

For each subscriber-to-client relation, several attributes are used to protect subscriber privacy and control access by subscribers to applications, including:

- Subscriber authorized to subscribe to and access the client application (granted by operator).
- Client allowed to locate, send messages to, and bill the subscriber (granted by subscriber and/or operator).
- Best allowed return accuracy in location responses for a specific client (set by subscriber), e.g. a dating service.

Irrespective of the established permissions, the subscriber may override all permissions, except the privacy override flag in the client profile, with a Global OFF privacy setting (the master privacy flag in the subscriber profile). This allows the ability to go invisible until the original permission sets are restored (Global ON privacy).

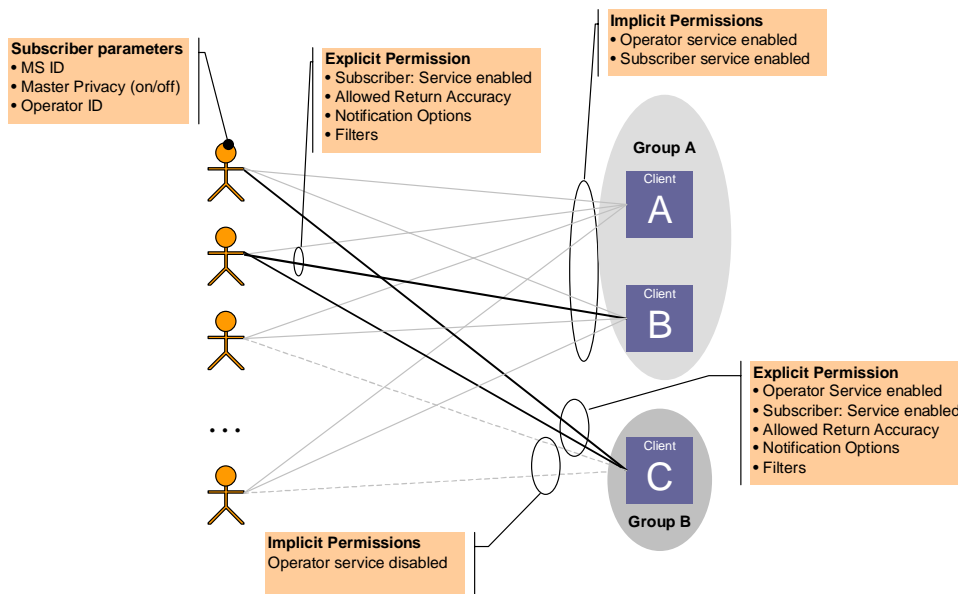
Explicit and Implicit Permissions

Permissions enable subscribers to filter how their location is made available to clients. Two types of permissions are used:

- **Inherited.** Location request permission is defined by each client's client group default permissions. These default permissions are used unless the subscriber has explicit permissions.
- **Explicit.** Permission is defined for a client according to the explicit settings in the client-subscriber permissions profile. Creating explicit permissions for a subscriber means that the inherited permissions (client group default permissions) are no longer applicable.

The following figure explains the difference between explicit and implicit permissions.

Figure 12 Permissions



In Figure 12 the subscribers are shown to the left. The subscriber has a few parameters that are the same for all clients, such as MSID, master privacy on/off and operator ID. When a new client application is provisioned in LSt, it inherits a number of parameters that are grouped together in a client group template. Of particular interest in this figure is the implicit permission settings are provisioned as if the client is first being created. After the first creation, the client profile may be modified. The default, or implicit, permission settings belong to the client group and are changed simultaneously for all clients that belong to a certain group. The implicit permission settings are the settings that hold for all users who have not explicitly set their permissions. For example, in Figure 12 clients A and B have the implicit setting operator service enabled and subscriber service enabled. This means that by default all users can use the services without being explicitly provisioned. This would typically be used for active information services. In addition, subscriber 2 has changed one of the settings that are associated with the permission, for example, position accuracy, notification option or filter option. Thus an explicit permission exists in the database for the relation between client B and subscriber 2, indicated by the black line. Client C has the operator service enabled flag set to false, which means that the subscribers who do not have an explicit permission with the operator service enabled flag set to true (provisioned only by the operator) are not allowed to access this service. Subscribers A and B have an explicit permission. This type of behavior may be desired e.g. for a colleague finder, where only the employees of a specific company, or companies, are allowed to join the service (and allowed to be positioned by the client).

In the contexts of explicit and implicit permissions, the two parameters Operator Service Enabled and Subscriber Service Enabled have slightly different meanings when they are implicit (default) and explicit. The following table explains the different meanings from an operator point of view, that is, how the different settings can be used. From a privacy enforcement point of view, (that is, when the privacy controls are checked in LSt) it does not matter if the subscriber-client relation is implicit or explicit. Both are treated the same way since the extraction of the permission from the database is separated from the use of it in the internal LSt logic. When you develop your client, use this information to make a recommendation to the wireless network operator how the default behaviour would best work for their application.

Table 4 Operator and Subscriber Service Enabled for explicit and implicit permissions

Field	Explicit Usage	Implicit (default) Usage
Operator Service Enabled	Indicates whether the subscriber is allowed to change the subscriber service enabled flag. In this case, the operator likely charges a fixed subscription fee for services. The subscriber is also allowed to subscribe to the service.	True: If this flag is set to true it means that all subscribers are allowed to set the subscriber service enabled flag.
Subscriber Service Enabled	<p>When this is explicitly set it means that the subscriber allows the corresponding client to position him (provided that no other filters are blocking this).</p> <p>In this case the operator charges a fixed subscription fee for services. The subscriber has also subscribed to the client service.</p>	<p>If this is true, the client is allowed to position all subscribers that have not explicitly disabled the client from doing this.</p> <p>In this case the operator charges a fixed subscription fee for services. The service is included in the base subscription fee.</p>

Time-of-Day and Day-of-Week Filter

The permission table described above is also linked to a filter that controls the time of day and day of week for the positioning allowed for a certain application. The attributes that control the time-filter are described in the following table.

ID Protection Service

Location Studio supports advanced features to support anonymous and private exchange of data with client applications. The primary method of protecting subscriber identity is through the use of aliases, from which the MIN/MSISDN is not easily derived. LSt supports a very flexible alias function, including:

Internally generated temporary and persistent identifiers. The temporary identifiers are only allowed to be used within a predefined time-period (default 30 seconds). These identifiers are referred to as TSID and PSID respectively.

Synchronization with external identifier managers (such as those generated within a WAP gateway, messaging gateways or operator portals). This kind of ID is called OSID (Operator Subscriber Identifier)

Distribution of identifiers (exchange with client applications) during subscriber-initiated service invocation and for community services originated from fixed-network (Internet) access

The use of subscriber aliases is mandatory by regulations or law in some jurisdictions.

The type of identifier used depends upon operator preference (the use of an external identifier vs. one generated by LSt), the client profile settings (specifies the valid identifier type for that client) or the type of positioning scenario (active vs. passive positioning).

A Temporary Subscriber Identifier (TSID) is generally used when the mobile station is engaged in an *active* session with an LBS application, and the subscriber has initiated a service requiring its location be provided to the client. Active sessions will use the Invocation Proxy feature of LSt to originate the LBS service using either WAP or SMS data services. Active sessions are commonly associated with

information-type services (e.g. find-the-nearest services) and some games/entertainment services where no persistent association between the subscriber and the end service is required.

A Persistent Subscriber Identifier (PSID) is generally used for *passive* positioning requests – when the request is made by the client application independent of any current action of the target mobile station. However, the PSID can also be used for active applications as way for the application to keep persistent personalization information about the subscriber between sessions. The mobile station may be in any state: off, idle, active with voice session, active with data session. Passive sessions are commonly associated with tracking or community types of services where a persistent association between the subscriber and the end service is required.

Client Interfaces

Advanced Application Interface (AAI)

Openwave's Location Studio provides the capability for LCS clients 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 as well as a less bug-prone technology (compared to other means such as XML). There is also a proxy part that handles actions that are Mobile Originated (i.e. initiated by the subscriber). The following functionality is available through the AAI:

Messaging. Location Studio provides SMS and WAP messaging functionality for LCS clients. Location Studio supports mobile terminated (MT-SMS and WAP push) and mobile originated (MO-SMS and WAP request, see **Proxy** below) messages. Mobile terminated messages are initiated by the LCS client, and sent to the end user through LSt.

Event Billing. The Wireless Billing feature in the LSt 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 LCS client. 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 sub-events are consolidated by the LCS client and requested on the event-billing interface as one event.

Location. Location Studio offers LCS clients the functionality to request subscriber location over the AAI interface. This is done so that LCS clients can utilize the easy to use Web Services technology; instead of implementing the rather complicated XML-based LIF MLP protocol.

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.

Proxy. Mobile originated messages are sent by end users to LCS clients through LSt (the proxy). The proxy performs a number of privacy checks, and forwards the messages (or WAP requests) to the client application.

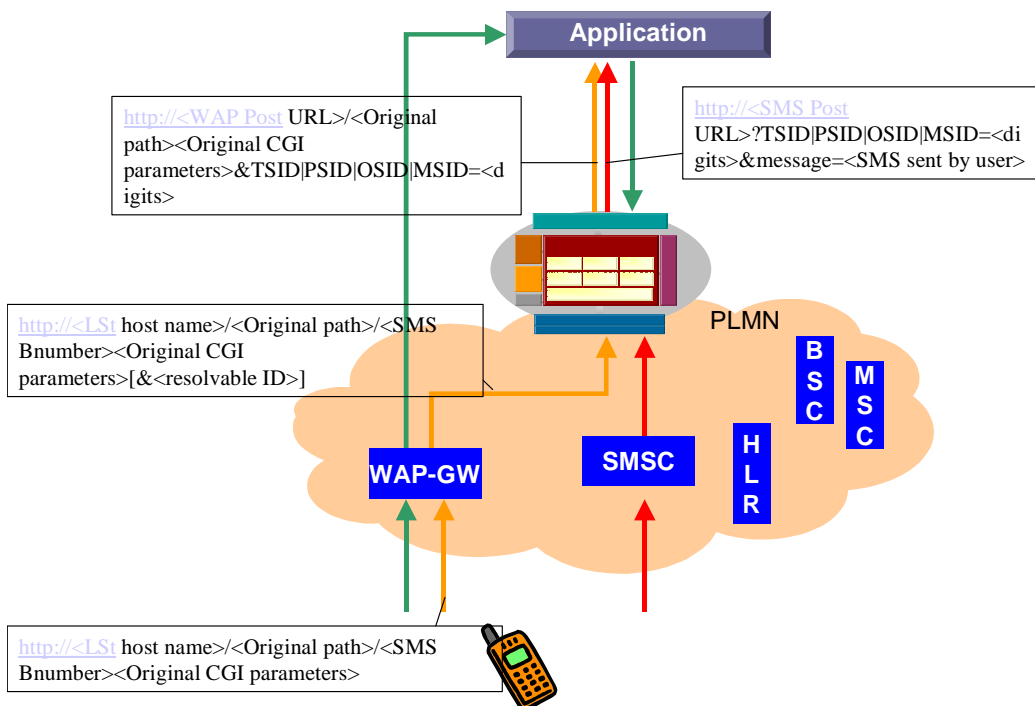
Proxy Services

Location Studio provides both WAP and SMS proxy functions for user invocation of WAP and SMS-based services. The WAP and SMS invocation proxies support both information and community types of

services, and provide key features necessary to support identification of subscriber-originated (active sessions) and applicable client access privileges.

The proxy functions are illustrated in the following diagram.

Figure 13 Proxy services



WAP

The Location Studio WAP proxy enables the following features:

Replacement of MSISDN or WAP gateway generated identifier with LSt-generated subscriber identifiers (TSID/PSID). The TSID/PSID may be used by the application on all the external interfaces i.e. MLP and AAI (consisting of WLI, WMP, WVP and WBP).

TSID/PSID identifiers enable anonymity (application does not know subscriber identity or MIN/MSISDN). The PSID is persistent from connection to connection and can provide personalization and automatic sign-on capabilities.

The WAP proxy feature requires that LSt proxy the traffic to and from the specified client that are related to a sign-on or a LSt related transaction, such as location requests. All other traffic does not have to pass through the proxy. LSt will proxy requests to the client, translating all incoming URLs to URLs corresponding to the client. Location Studio will derive the outgoing URL from the client profile.

For example, the incoming URL, after the addition of the MSISDN in the WAP gateway may look like:

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

The example follows the general LSt incoming proxy format:

```
http://LSt host name{:port}/locationstudio/waproxy/Original path/SMS
Bnumber Original CGI parameters[&MSISDN|WAP ID parameter|other
resolvable ID]
```

In this URL the [] part is added by the WAP GW and the {} part is optional (depending on configuration). This means that the URL without the [] part is the format that the LCS client should use when it wants to LCS enable the URL link.

The inclusion of the MSISDN will be different for different WAP gateways, for some it is the WAP ID that will be included, and for some the information will be in the header rather than the URL. Location Studio handles all these cases through a professional services adaptation of the proxy. The Location Studio proxy looks up the client and the corresponding Post URL field from the SMS BNumber part of the URL, and replaces the MSISDN parameter with the TSID/PSID/OSID depending on the TSID/PSID/OSID Allowed flags in the client profile. For example, if the application is Friendfinder at the operator M2, the outgoing URL may be:

```
http://wap.M2ff.com/dialog/start?action=12345&PSID=7656754215342364536253
```

The example follows the general Location Studio outgoing proxy format:

```
http://WAP Post URL/Original path Original CGI
parameters&TSID|PSID|OSID=digits
```

SMS

A key feature of LSt is the support of Mobile Originated SMS for LBS service invocation. This feature allows for creation of easy to use messages that will perform a function within an application and return an SMS message with the result. An example would be to send a command word of "FIND" followed by a word describing what you are looking for "Pizza" and the application could then return information about the 3 closest pizza restaurants. This interface can be utilized to SMS enable applications that currently are only Web and WAP enabled.

The Location Studio SMS proxy enables the following features:

- Delivery of temporary and persistent subscriber identifiers (TSID/PSID), which are used by the application to position the MS, send receive SMS messages, and generate billing events.
- TSID/PSID identifiers enable anonymity.
- The SMS proxy feature requires that LSt proxy all SMS traffic to and from the specified client. Location Studio will proxy requests to the client, translating MIN/MSISDN to TSID/PSID for each transaction.

The URL posted to the application has the following format:

```
http://SMS Post URL?TSID|PSID|OSID=digits&message=SMS sent by user
```

An example in the above Friendfinder case may be:

```
http://wap.M2ff.com?PSID=7656754215342364536253&message="Find Mat"
```

Notification Service

Location Studio supports notification options for each subscriber-to-client permission based on either SMS or WAP-push messaging. There are non-subscriber specific default permissions that apply to all clients recently provisioned into the subscriber's profile (authorized for access by the operator). The subscriber can further modify all permission options (including notification) on a client-by-client basis. The notification options provided are as follows:

- **Notify only.** The subscriber is notified asynchronously about the positioning event. This is configured on a permission basis to be done via either SMS or WAP.
- **Notify with response required.** A response is required from the subscriber to allow the client's request. If the subscriber has not answered within a certain time-period the client gets a negative response from LSt on his request. This is configured on a permission basis to be performed either via WAP push or SMS.
- **Do not notify.** The most commonly used option. Normally this would be used e.g. for all anonymous active services.

When developing your client, ensure that you develop the client in a method (WAP or SMS) suitable with how the network operator will notify subscribers. The basic sequence for an SMS or WAP notification without verification use case is as follows:

1. A client requests the position of a subscriber.
2. Location Studio receives the request and checks if the Default notification option is set to notify SMS or notify WAP. (Location Studio also checks if the Default notification option is set to notify WAP ask or notify SMS ask, but in this use-case this is not true). If this is the case LSt sends a message to the subscriber using the message stored in the client table database fields WAP_NOTIFICATION_MESSAGE or SMS_NOTIFICATION_MESSAGE, respectively
3. In this use case there is no ask feature set, and thus LSt retrieves the location from the network.
4. Location Studio responds to the original location request.

The basic sequence for an SMS or WAP notification with verification use case is as follows:

5. A client requests the position of a subscriber.
6. Location Studio receives the request and checks if the Default notification option is set to notify WAP ask or notify SMS ask.
7. In this use case either the WAP or SMS ask feature is set, and thus LSt sends a notification to the subscriber, using either WAP push or SMS. The actual message text is retrieved from the fields WAP_ASK_NOTIFICATION_MESSAGE or SMS_ASK_NOTIFICATION_MESSAGE in the client table respectively. The messages contain placeholders for the client name and a verification code that is generated in order to later on correlate the response with the original message sent out. The placeholders are filled in with appropriate information before the message is sent.
8. The subscriber responds to the message. In the WAP push case the user pushes a link provided in the message. In the SMS case, the user responds to the SMS message with the code that was provided in the received notification.
9. Location Studio receives the response from the subscriber. The response is received from a special-purpose WAP notification servlet in the WAP case and in the messaging component in the SMS case. If no response is returned within a pre-specified time-period, LSt assumes that the response was negative.
10. If the response was positive the location is retrieved from the network.
11. Location Studio responds to the original location request, either with the location or an appropriate error message.

Sample Call Flows

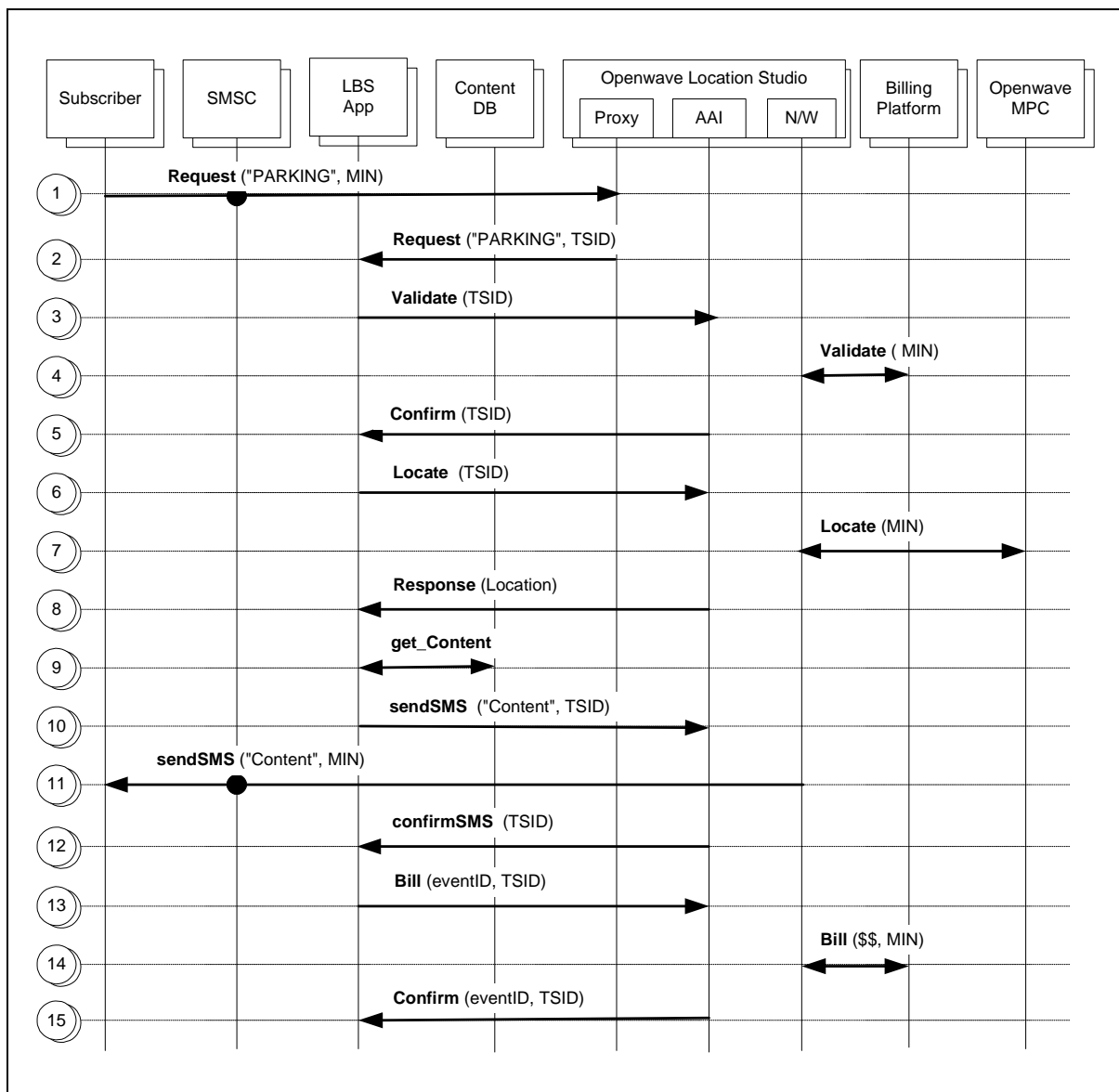
The call flows in Figure 14 and Figure 15 illustrates how LSt facilitates the delivery of a premium-billed, location-enabled service using either SMS or WAP as a bearer.

NOTE These call flows have been simplified for illustration purposes. More precise definition of service requests and required parameters may be found in the AAI Interface Control Document.

NOTE These call flows are designed to be illustrative by nature. Not all messages are required to location-enable a service. They are shown merely to indicate the breadth of services available if desired.

SMS Call Flow

Figure 14 SMS Call Flow



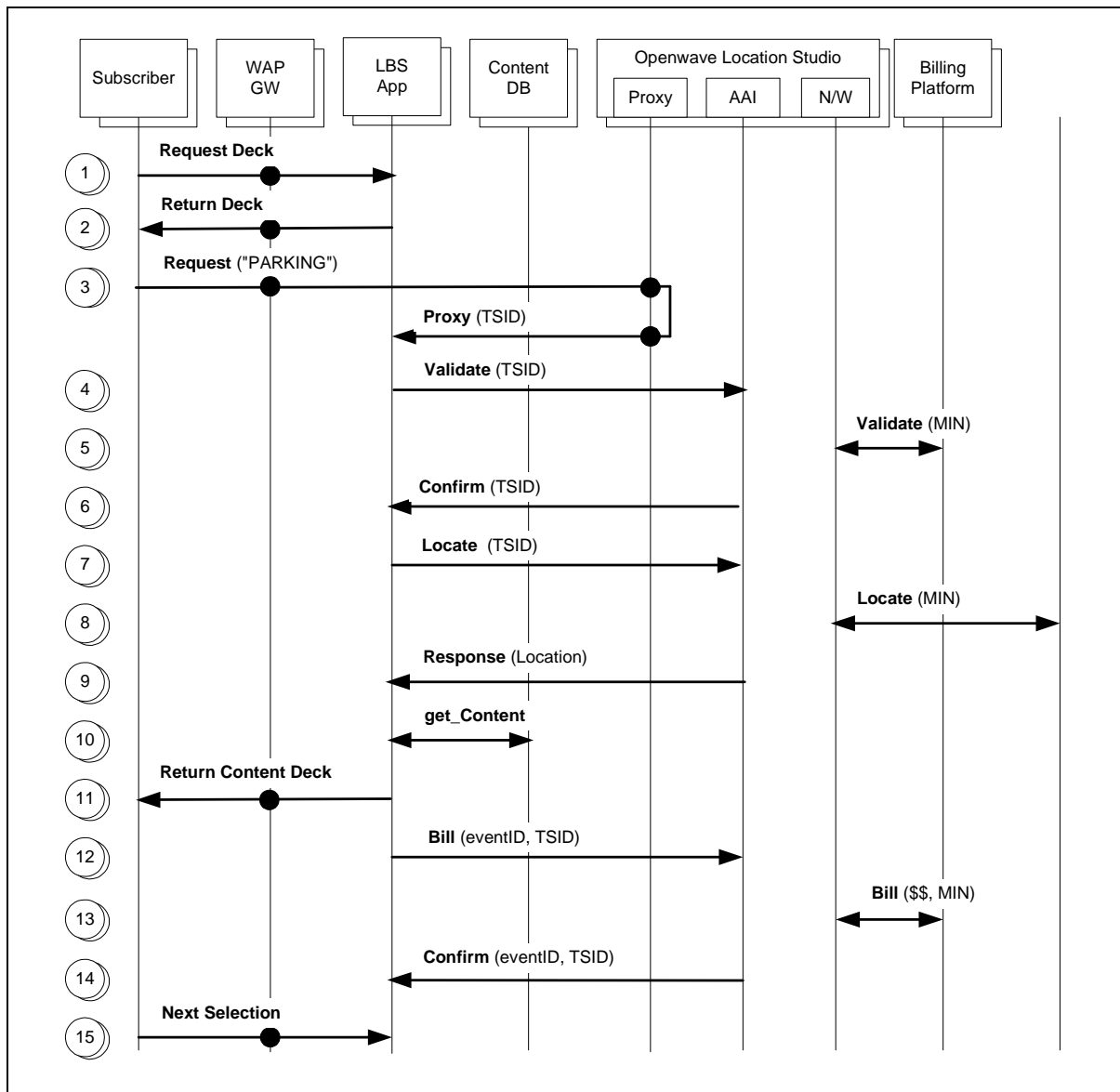
1. A mobile subscriber originates an SMS message with the keyword “Parking” in the text field. The SMS command is addressed to a short code, which terminates within the Location Studio SMS Proxy

and uniquely identifies the destination application (LBS App) for this service. The SMSC recognizes the mobile station by the Mobile Identification Number (MIN) and generates appropriate SMS bearer charging.

2. The Location Studio SMS Proxy substitutes the MIN with a Temporary Subscriber Identifier (TSID), and forwards the keyword “Parking” and TSID to the LBS App.
3. The LBS App submits a validation request to LSt, using the validation service of the AAI, to ensure that the subscriber is authorized to use the service.
4. Location Studio (LSt) validates that the subscriber identified by the TSID is authorized to use the requested service (LBS App) and, if the subscriber is a pre-pay customer, it is possible to extend this validation to test the pre-pay billing platform to qualify the subscriber status.
5. LSt responds to the validation request indicating the subscriber is authorized, and has a positive billing status.
6. The LBS App submits a location service request via the AAI using the TSID to identify the subscriber.
7. Location Studio requests and receives the subscriber location from the Openwave *Location Manager* Mobile Positioning Center (MPC).
8. Location Studio responds to the location service request with the required subscriber location.
9. The LBS App uses the subscriber’s location to search a content database for the nearest parking facilities that have available parking. The content database may be local to the LBS App (ie. in an Oracle spatial database), or it may be a remote database managed by a third-party content provider (or the operator).
10. The LBS App formats the available content as a text message and submits the text for SMS delivery to the subscriber using the messaging service of the AAI. The subscriber is once again identified in the request using the same TSID obtained from the SMS Proxy in step 2.
11. Location Studio submits the content received from the LBS app and formats it in an SMS text message to the SMSC for immediate delivery to the subscriber. The TSID ids used
12. Location Studio confirms submission of the SMS message to the SMSC, but not receipt of the message by the subscriber.
13. Upon successful delivery of service to the subscriber the LBS App submits a billing request to LSt, along with an event ID, using the billing service of the AAI. The event ID indicates to the billing system the total service charge.
14. Location Studio processes the request and interacts with the Billing Platform to debit funds for pre-pay customers, or for a post-pay subscriber, to create a CDR for the event.
15. Location Studio confirms the successful billing event.

WAP Call Flow

Figure 15 WAP Call Flow



1. A mobile subscriber originates a WAP session with the LBS Application.
2. A WML decks is presented to the subscriber with some cards that link to a service that requires a location update to complete, and some that do not require location information. The cards that require a location update link to the WAP Proxy in LSt.
3. The subscriber selects a card for the "Parking" service. The card contains a URL that terminates within the Location Studio WAP Proxy and contains parameters that indicate where to return the subscriber WAP session when the proxy function is complete. Location receives the identify of the mobile station (MIN) from the WAP Gateway and generates a TSID for use by the LBS App for subsequent (time limited) transactions with LSt. The TSID is returned as a parameter in the URL that is re-directed to the LBS App.
4. The LBS App submits a validation request to LSt, using the validation service of the AAI, to ensure that the subscriber is authorized to use the service.

5. Location Studio (LSt) validates that the subscriber identified by the TSID is authorized to use the requested service (LBS App) and, if the subscriber is a pre-pay customer, it is possible to extend this validation to test the pre-pay billing platform to qualify the subscriber status.
6. Location Studio responds to the validation request indicating the subscriber is authorized, and has a positive billing status.
7. The LBS App submits a location service request via the AAI using the TSID to identify the subscriber.
8. Location Studio requests and receives the subscriber location from the Openwave *Location Manager* Mobile Positioning Center (MPC).
9. Location Studio responds to the location service request with the required subscriber location.
10. The LBS App uses the subscriber's location to search a content database for the nearest parking facilities that have available parking. The content database may be local to the LBS App (ie. in an Oracle spatial database), or it may be a remote database managed by a third-party content provider (or the operator).
11. The LBS App formats the available content as a WML deck and returns the deck to the subscriber. As an alternative, or as an option within the deck, the LBS App may return the content formatted as a SMS text message, and may use the messaging service of the AAI to deliver the text message (not shown in this call flow).
12. Upon successful delivery of service to the subscriber the LBS App submits a billing request to LSt, along with an event ID, using the billing service of the AAI. The event ID indicates to the billing system the total service charge.
13. Location Studio processes the request and interacts with the Billing Platform to debit funds for pre-pay customers, or for a post-pay subscriber, to create a CDR for the event.
14. Location Studio confirms the successful billing event.
15. The subscriber continues navigation of the WML decks of the LBS Application. As in Step 2, some cards may require a location update to continue, in which case they are routed through the LSt WAP Proxy, and others simply interact directly with the LBS App and do not require use of the proxy.

Developing a Location-Intelligent Application

The following subsections contain step-by-step instructions for how to obtain java classes that will simplify the use of the AAI, how to compile your client application and how to run your application. Each subsection uses a different method and different tools. All subsections assume the following:

- You have already decided which LSt Web Services to use. If not, refer to the AAI Specification, reference [9]. You understand how your client application fits into your mobile operators network. You have already designed your client application using information in the AAI Specification. If not, please refer to the AAI and design your client application. If you do not have a client application in mind, you can follow the step-by-step instructions given here using one or more of the example client applications downloadable from the Openwave LSt 2.0 SDK website. These example client applications do not carry out any real business function; they simply demonstrate how to execute an AAI operation, how to set all the arguments of an AAI operation, how to catch exceptions generated by AAI operations and how to look at the values returned by a successful AAI operation.
- You understand the concepts of Web Services and WSDL, the Web Services definition language, specifically references [2], [3], [4], and [5]. The LSt AAI is based on Web Services technology and this document makes no attempt to explain that technology or its terminology.
- You have access to an LSt 2.0 installation, you know the name or IP address of the host on which it is running, you know the LSt port number on your LSt host and you have client, subscriber and permission entities properly provisioned in LSt. To facilitate the development and testing of new client applications, Openwave provides a LSt 2.0 instance running on an Openwave host. That host is currently named `lst-demo1.signalsoftcorp.com` and it uses the default port number 8080. A later section of this document describes the data entities already provisioned in the test system.
- The computer system on which you plan to develop and run your client application has a Java SDK already installed on it. The version of your Java SDK must be 1.3 or higher.

Method A—Using Openwave Classes

This section contains recommended step-by-step instructions for how to get your client application up and running. The instructions here use Openwave java classes provided in the `locationstudio_client.jar` file and associated Glue archives. If you have reason to not use the Openwave classes or prefer to use your own version of Glue, go to the Method B or Method C sections of this document. Note that the version of Glue archives is important; they **MUST** be the same version that was used in the creation of `locationstudio_client.jar`. Therefore, it is strongly recommended that you use the Glue archives that are included in the Openwave SDK package, even if you already have another Glue installation in your development environment.

Step 1, Method A. Download the Openwave Location Studio SDK Package

After downloading and expanding the Openwave SDK package, you should see a copy of this document and a copy of the AAI specification in the directory where you expanded the package. You should also see subdirectories named MethodA and MethodB.

The MethodA subdirectory of the Openwave LSt SDK package includes everything you need to compile and run an LSt client application except the Java SDK and a LSt web server installation. All the files you need to follow and execute the instruction in this Method A section of this document are contained in the MethodA subdirectory.

In the MethodA/lib subdirectory, you should see the locationstudio_client.jar file and all the Glue archives needed for method A. The locationstudio_client.jar file contains a set of java classes to simplify the use of the LSt Web Services. By providing these classes, we eliminate the need for you to generate client-side classes from the LSt WSDL. However, if you prefer to generate classes from WSDL, please proceed to method B.

The classes contained in locationstudio_client.jar differ from those generated from WSDL (as described in method B) in that they include get and set methods for private data members, they include constructors that support setting several data members, they have meaningful argument names, they include javadocs and they include a set of java constants (such as error codes) with meaningful names that eliminate the need to look up and hard-code constant values into your client application.

In addition, the locationstudio_client.jar file includes config.xml and standard.map files that Glue will use when your client application runs. Thus if you wish to modify Glue's configuration or mappings, you should start with the config.xml and standard.map files included in locationstudio_client.jar. One reason to modify Glue's configuration is to get more diagnostic and debug information at runtime. Please see Getting More Diagnostics from Glue and the "Applets" section of the Glue User's Guide, reference [2].

In the MethodA/docs subdirectory, you should see javadocs for all classes in locationstudio_client.jar.

In the MethodA/src subdirectory, you should see a number of example client applications, which will be described in more detail below.

The MethodA/classes subdirectory should be empty initially. This directory will be used to hold the .class files generated by the compile step described below.

In the MethodA directory, you should see a makefile, a makefile.bat file and an ant build file to support compiling in both UNIX and Windows environments. In addition, you should see several run scripts, which demonstrate how to run a client application. You should also see an example properties file that is used by all the example client applications.

Openwave provides locationstudio_client.jar and everything else in the MethodA subdirectory to make it faster and easier for you to write your own client application to take advantage of LSt's Web Services. This in no way implies that Openwave is not fully compliant with Web Services functionality. Openwave fully supports both Method A and Method B as described in this document.

Step 2, Method A. Write Your Client Code and Edit the Example Clients

Write your client application using the classes provided by Openwave. Browse the javadocs to see the classes, their methods, method arguments, response classes and error codes. Refer to the AAI specification for more explanation of operations and example call flows.

Note that your client application must include a call to Glue's Registry.bind method. Each of the example client application .java files includes such a call. If these examples are not sufficient, please refer to the Glue User's Guide, reference [2], for more explanation.

Even if you are writing your own client application, you may wish to look at the example client applications to see the code for a call to each AAI operation. There is one example application for each AAI operation. The source code files for the example client applications are named src/Example*.java where * is the name of an AAI operation. For example, if you wish to use the sendSmsMessage operation, you will find an example of its use in the java source code file named ExampleSendSmsMessage.java in the MethodA/src subdirectory. Since there are 2 incarnations of the getLocation operation, there are 2 java source code files, named ExampleGetLocationMLP.java and ExampleGetLocationSimple.java.

If you are going to actually run the example client applications, you need to edit the variables used by the applications. Each example application reads its variable values from the properties file named example_variables.properties. You need to edit the values in the properties file to contain values that work for your LSt installation. Each example client application uses a variable named LStURL, whose

original value in the properties file includes the hostname of the Openwave demo host. This demo host is provided by Openwave to facilitate the development and testing of new client applications. But if you want to use your own LSt installation, then change the hostname in the LStURL property to the hostname or IP address of the host on which your LSt instance is running. For example, the `example_variables.properties` file contains the line

```
LStURL=http://lst-  
demo1.signalsoft.com:8080/locationstudio/webservices/aai.wsdl
```

Where `lst-demo1.signalsoftcorp.com` is the hostname of the Openwave demo host. If you want to use your own LSt installation, change `lst-demo1.signalsoftcorp.com` to the hostname or IP address of the host on which your LSt installation is running.

The values for other variables in the properties file are valid for the LSt instance running on the Openwave demo host. If you changed the hostname to use your LSt host, then you should also edit the other variables to contain values that work for your installation. In either case, the example client applications should run and LSt may return an error message if the variable values are not consistent with the LSt installation.

Optionally, you can create your own properties file and give its name as the only command line argument when running the example client applications. In this case, the `example_variables.properties` file will not be read.

Complete definitions and explanations of all parameters to all operations are given in both the javadocs and in the AAI specification, reference [9]. Please use the javadocs and AAI specification as your references. The values in `example_variables.properties` are simply examples.

Note that the Openwave SDK package provides 2 sets of example client applications, one in the `MethodA` subdirectory and another in the `MethodB` subdirectory. Those in `MethodA` use the Openwave client classes provided in `locationstudio_client.jar`. Those in `MethodB` use classes that are generated by running `wsdl2java` for an internal LSt 2.0 installation. They are not the same so do not attempt to mix and match.

Step 3, Method A. Compile Your Client and the Example Clients

To successfully compile your client application, you need to know

- The path to your Java SDK directory,
- The path to the `locationstudio_client.jar` file and the Glue jar files, which should be `./lib`

To compile your client application, your classpath needs to include the following Glue .jar files, which are included in the Openwave SDK package: `GLUE-STD.jar`, `jnet.jar`, `jsse.jar`, `servlet.jar` and `xerces.jar`. For example, to compile the example client application found in `src/ExampleSendSmsMessage.java` in a UNIX environment, do the following:

```
/bin/javac -classpath lib/GLUE-  
STD.jar:lib/jnet.jar:lib/jsse.jar:lib/servlet.jar:lib/xerces.jar:lib/loc  
ationstudio_client.jar -d classes src/ExampleSendSmsMessage.java
```

Note that the above command puts the resulting `ExampleSendSmsMessage.class` file in the `classes` subdirectory.

The `MethodA` directory contains an example makefile that can be used by `gmake` to compile all the example applications. Before using this makefile, be sure to edit the `JAVAHOME` make variable and then run it in the `MethodA` directory.

The `MethodA` directory also contains an example `build.xml` file that can be used by `ant` to compile all the example applications.

Both the makefile and the build.xml file expect to find the .java files in the src subdirectory, the necessary .jar files in the lib subdirectory and they will put the resulting .class files in the classes subdirectory.

For help compiling in a Windows environment, see the makefile.bat file in the MethodA directory. Again, be sure to edit the JAVAHOME variable to be valid in your environment. See also the build.xml file, which can be used by ant to compile the example applications in a Windows environment. Again, both the makefile.bat and the build.xml file expect to find the .java files in the src subdirectory, the necessary .jar files in the lib subdirectory and they will put the resulting .class files in the classes subdirectory.

Step 4, Method A. Run Your Client or the Example Clients

To successfully run your client application, you need to know the paths to your Java SDK and .jar files as described in the previous step. And your CLASSPATH environment variable needs to include the same .jar files as described in the previous step plus the directory containing the .class files created in the previous step.

If you are going to run one of the example client applications, first remember to edit the variable values in the example_variables.properties file, as described previously.

For example, to run the ExampleSendSmsMessage client application in a UNIX environment, do the following:

Edit example_variables.properties to contain your variables values.

```
CLASSPATH=classes:lib/locationstudio_client.jar:lib /GLUE-  
STD.jar:lib/jnet.jar:lib/jsse.jar:lib/servlet.jar:lib/xerces.jar  
/bin/java ExampleSendSmsMessage
```

All example client applications will recognize one command line argument and use it as the name of a properties file to read. If a properties file name is given as a command line argument, then the example_variables.properties file will not be read.

The MethodA directory contains an example shell script, named runExample.sh that can be used to run any of the example client applications in a UNIX environment. Before using this script, be sure to edit the JAVAHOME variable. Then run the script in the MethodA directory and enter the name of any example application as an argument to the script. Any command line argument given to this shell script will be passed to the client application. For example, to run the ExampleSendSmsMessage client application, do the following:

```
runExample.sh ExampleSendSmsMessage
```

For help running the example client applications in a Windows environment, see the runExample.bat file in the MethodA directory. Again, be sure to edit the JAVAHOME variable to be valid in your environment. And, again, remember to edit the variable values in the example_variables.properties file if you are running one of the example client applications.

Step 5, Method A. Look for Expected Results

If you are running the example client applications, there should be enough information printed to standard output that you can see what happened and any errors returned. If there are errors, see the AAI section about error codes and go to the Troubleshooting section of this document for more help.

If you are attempting to debug your own client application and are not getting the expected results, you can use one or more of the example client applications, edit the variable values in the properties file to use your data and run the example to see the returned values.

Method B—Generating Classes From WSDL

This section contains step-by-step instructions for how to get your client application up and running. The instructions here use Glue tools to generate java classes from the LSt 2.0 WSDL. To be successful with this method, you are not restricted to any particular Glue version. If you have reason to not use Glue at all or prefer to use a different 3rd party tool to generate java classes from WSDL, go to the Method C section of this document.

Step 1, Method B. Download the Openwave LSt SDK Package

After downloading and expanding the Openwave SDK package, you should see a copy of this document and a copy of the AAI specification in the directory where you expanded the package. You should also see subdirectories named MethodA and Method B.

The MethodB subdirectory of the Openwave LSt SDK package includes everything you need to compile and run an LSt client application except the Java SDK, a Glue Standard Package and a LStn web server installation. Everything you need to follow and execute the instruction in this Method B section of this document is contained in the MethodB subdirectory.

In the MethodB/src subdirectory, you should see a number of example client applications, which will be described in more detail below.

The MethodB/classes subdirectory should be empty initially. This directory will be used to hold the .class files generated by the compile step described below.

In the MethodB directory, you should see a do_wsdl2java shell script and bat file to support the generation of client-side java classes from LSt's WSDL in both the UNIX and Windows environments. In addition, you should see a makefile, a makefile.bat file and an ant build file to support compiling in both UNIX and Windows environments. In addition, you should see several run scripts, which demonstrate how to run a client application. You should also see an example properties file that is used by all the example client applications.

The contents of the MethodB directory in the download package are described in [The LSt 2.0 SDK Download Package](#)~~The LSt 2.0 SDK Download Package~~.

Step 2, Method B. Download Glue

If you do not already have a version of Glue, download it from The Mind Electric website. The top level Glue link is <http://www.themindelectric.com/glue/index.html>. From there, select the “Developers” link, at which you can see the “Download” link. The “users guide” link takes you to a large, well-written set of documentation about all aspects of Web Services. While the majority of that documentation is for web application *server* developers, the section titled “Invoking Webservices” is written specifically for you, the client application developer. That section is recommended reading, even if you think you know what you’re doing. The “Binding and Invoking” subsection of the “Invoking Webservices” section shows how to use the *wsdl2java* tool to generate client classes from WSDL and includes a link to more information about the *wsdl2java* utility.

Note that the Glue link shown above and subsequent link names at the Glue website were correct at the time this document was written. If The Mind Electric reorganizes their website, these link names may no longer be correct.

Step 3, Method B. Generate Client-Side Java Classes

Use the *wsdl2java* Glue utility to generate java classes from your LSt installation's WSDL. To successfully use *wsdl2java* in a UNIX environment, you must first:

- Set the `ELECTRIC_APPHOME` environment variable to the path to your Glue home directory,
- Set the `CLASSPATH` environment variable to include the .jar files found in `$ELECTRIC_HOME/lib`,
- Find the URL of the LSt 2.0 installation that serves the LSt WSDL, typically:

```
http://host:8080/locationstudio/webservices/aai.wsdl,
```

Where `host` is replaced with the hostname or IP address of the machine where your LSt instance is running. Note that the LSt port number is typically 8080 but may be different on your LSt installation. To facilitate client application development and testing, Openwave provides an LSt instance running on a host named `lst-demo1.signalsoftcorp.com`. If you use the `lst-demo1` demo host to generate java classes, then you must also use `lst-demo1` to run those classes. If you later switch to your own LSt installation, you must regenerate your java classes using the WSDL served up by your LSt installation.

- Then run the *wsdl2java* utility as follows:

```
$ELECTRIC_APPHOME/bin/wsdl2java -d src -c  
http://host:8080/locationstudio/webservices/aai.wsdl
```

This should result in the creation of 27 .java files and a file named `AAIService.map`. Using the above command, the .java files will be created in the `src` subdirectory but the `AAIService.map` file will be created in the current working directory. The `AAIService.map` file is required by Glue when the final client application is run.

IMPORTANT Do not delete or move the `AAIService.map` file. It must be in the current working directory when the client application is run.

Note that the `MethodB` subdirectory contains a shell script named `do_wsdl2java.sh` that encapsulates the above commands. Before using this shell script, be sure to change the values of the `GLIB` and `ELECTRIC_APPHOME` variables to valid paths on your system. The shell script uses the Openwave demo host, so if you want to use your own LSt installation, you **MUST** also edit the hostname in the shell script.

The `MethodB` subdirectory also contains an example `build.xml` file that can be used by `ant` to run the `do_wsdl2java` shell script for you. Since it uses the shell script, be sure to edit the shell script as described above before running `ant`. In addition, be sure to override the `glue.home` property value in `build.xml` to define the path to your Glue jar files.

For help running the Glue *wsdl2java* utility in a Windows environment, see the `do_wsdl2java.bat` file in the `MethodB` subdirectory. Again, be sure to edit the `ELECTRIC_APPHOME` and `GLIB` variables and hostname to be valid in your environment. Again, the `MethodB` subdirectory also contains an example `build.xml` file that can be used by `ant` to run the `do_wsdl2java.bat` file for you. Since it uses the shell script, be sure to edit the shell script as described above. In addition, be sure to override the `glue.home` property value in `build.xml` to define the path to your Glue jar files.

Step 4, Method B. Write Your Client and Edit the Example Clients

Write your client application using the classes created in the previous step. Note that the *wsdl2java* utility does not generate javadocs so you will have to look directly at the generated .java files in the `src` subdirectory. Also, refer to the AAI specification for more explanation of operations and example call flows.

Note that your client application must include a call to Glue's Registry.bind method. Each of the example client application .java files includes such a call. If these examples are not sufficient, please refer to the Glue User's Guide, reference [2], for more explanation.

Even if you are writing your own client application, you may wish to look at the example client applications to see an example call to each AAI operation. The example client applications can be found in the MethodB subdirectory of the SDK package. There is one example application for each AAI operation. The source code files for the example client applications are named src/Example*.java, where * is the name of an AAI operation. For example, if you wish to use the sendSmsMessage operation, you will find an example of its use in the java source code file named ExampleSendSmsMessage.java in the MethodB/src subdirectory. Since there are 2 incarnations of the getLocation operation, there are 2 java source code files, named ExampleGetLocationMLP.java and ExampleGetLocationSimple.java.

If you are going to actually run the example client applications, you need to edit the variables used by the applications. Each example application reads its variables from a properties file named example_variables.properties. You need to edit the values in the properties file to contain values that work for your LSt installation. Each example client application uses a variable named LStURL, whose original value in the properties file includes the hostname of the Openwave demo host. This demo host is provided by Openwave to facilitate the development and testing of new client applications. But if you want to use your own LSt installation, then change the hostname in the LStURL property to the hostname or IP address of the host on which your LSt instance is running. For example, the example_variables.properties file contains the following line:

```
LStURL=http://lst-  
demo1.signalsoft.com:8080/locationstudio/webservices/aai.wsdl
```

Where lst-demo1.signalsoftcorp.com is the hostname of the Openwave demo host. If you want to use your own LSt installation, change lst-demo1.signalsoftcorp.com to the hostname or IP address of the host on which your LSt installation is running.

The values for other variables in the properties file are valid for the LSt instance running on the Openwave demo host. If you changed the hostname to use your LSt host, then you should also edit the other variables to contain values that work for your installation. In either case, the example client applications should run and LSt may return an error message if the variable values are not consistent with the LSt installation.

Optionally, you can create your own properties files and give its name as the only command line argument when running the example client applications. In this case, the example_variables.properties file will not be read.

Complete definitions and explanations of all parameters to all operations are given in the AAI Specification, reference [9]. The values in example_variables.properties are simply examples.

Note that the Openwave SDK package provides 2 sets of example client applications, one in the MethodA subdirectory and another in the MethodB subdirectory. Those in MethodA use the Openwave client classes provided in locationstudio_client.jar. Those in MethodB use classes that are generated by running *wsdl2java* for an internal LSt 2.0 installation. They are not the same so do not attempt to mix and match.

Step 5, Method B. Compile Your Client and the Example Clients

To successfully compile the classes created in the previous step and your client application, you need to know

- The path to your Java SDK directory,
- The path to your Glue library directory, from here on referred to as \$GLIB,
- The path to the classes created in the previous step, which should be src

To compile, your classpath needs to include the following .jar files from your Glue installation: GLUE-STD.jar, jnet.jar, jsse.jar, servlet.jar and xerces.jar. For example, to compile the example client application found in src/ExampleSendSmsMessage.java, do the following:

```
/bin/javac -classpath src:$GLIB/ GLUE-STD.jar:$GLIB/jnet.jar:$GLIB/jsse.jar:$GLIB/servlet.jar:$GLIB/xerces.jar -d classes src/ExampleSendSmsMessage.java
```

Note that the above command puts the resulting ExampleSendSmsMessage.class file in the classes subdirectory.

The MethodB directory contains an example makefile that can be used with gmake to compile all the example applications. Before using this makefile, be sure to edit the JAVAHOME and GLIB make variables and then run it in the MethodB directory.

The MethodB directory also contains an example build.xml file that can be used by ant to compile all the example applications. Before using this file, be sure to override the glue.home property value to define the path to your Glue jar files.

Both the makefile and the build.xml file expect to find the .java files in the src subdirectory and they will put the resulting .class files in the classes subdirectory.

For help compiling in a Windows environment, see the makefile.bat file in the MethodB directory. Again, be sure to edit the JAVAHOME and GLIB variables to be valid in your environment. See also the build.xml file, which can be used by ant to compile the example applications in a Windows environment. Again, be sure to override the glue.home property value to define the path to your Glue jar files. Again, both the makefile.bat and build.xml file expect to find the .java files in the src subdirectory and they will put the resulting .class files in the classes subdirectory.

Step 6, Method B. Run Your Client or the Example Clients

To successfully run your client application, you need to know the path to your Java SDK and the same \$GLIB as described in the previous step. And your CLASSPATH environment variable needs to include the same .jar files as described in the previous step plus the directory containing the .class files created in the previous step.

If you are going to run one of the example client applications, first remember to edit the variable values in the example_variables.properties file, as described previously.

For example, to run the ExampleSendSmsMessage client application in a UNIX environment, do the following:

Edit example_variables.properties to contain your variables values.

```
GLIB=/export/GLUE3.2.2/electric/lib
CLASSPATH=classes:$GLIB/GLUE-STD.jar:$GLIB/jnet.jar:$GLIB/jsse.jar:
$GLIB/servlet.jar:$GLIB/xerces.jar
/bin/java ExampleSendSmsMessage
```

NOTE Glue requires the AAIServices.map file to be in the current directory when you run the client application. This file was created by the wsdl2java utility in a previous step.

All example client applications will recognize one command line argument and use it as the name of a properties file to read. If a properties file name is given as a command line argument, then the example_variables.properties file will not be read.

The MethodB directory contains an example shell script, named runExample.sh that can be used to run any of the example client applications in a UNIX environment. Before using this script, be sure to edit the JAVAHOME and GLIB variables. Then run the script in the MethodB directory and enter the name of any example app as an argument to the script. Any command line argument given to this shell script will

be passed to the client application. For example, to run the ExampleSendSmsMessage client application, do the following:

```
runExample.sh ExampleSendSmsMessage
```

For help running the example client applications in a Windows environment, see the runExample.bat file in the MethodB directory. Again, be sure to edit the JAVAHOME and GLIB variables to be valid in your environment. And, again, remember to edit the variable values in the example_variables.property file if you are running one of the example client applications.

Step 7, Method B. Look for Expected Results

If you are running the example client applications, there should be enough information printed to standard output that you can see what happened and any errors returned. If there are errors, see the AAI section about error codes and go to Troubleshooting for more help.

If you are attempting to debug your own client application and are not getting the expected results, you can use one or more of the example client applications, edit the variable values in the properties file to use your data and run the example to see the returned values.

Method C

This section contains limited information about getting your client application up and running. The information here is simply a general outline of steps that you can investigate on your own.

Step 1, Method C. Generate Client-Side Java or C++ Classes

Use a tool or utility to generate client java classes from the WSDL provided by your LSt web server. Note that you can browse to your LSt web server to view the WSDL definition of the Web Services provided there. You can also use a utility to generate C++ client classes rather than java client classes. And if you know enough about SOAP and WSDL, you can also write your own code to encode/decode and send/receive SOAP messages to/from your LSt web server.

Step 2, Method C. Write Your Client

Typically, you will use the Java or C++ classes created in the previous step.

Step 3, Method C. Compile and Run Your Client

Compile both your client application code and the classes generated in the previous steps. After compiling, run your client application.

Receiving Mobile-Originated Messages

This section gives an example of how to receive a mobile-originated message in your client application. In the AAI Specification, this is called “Upstream” communications. Sections 5.1.2 and 5.1.3 in the AAI Specification, titled “Upstream SMS Proxy Example” and “Upstream WAP Proxy Example” show typical use cases in which the user of a mobile phone is the initiator of a communication with your client application. The AAI specification includes fully annotated call flows for these cases. Please refer to that document to understand the context in which this discussion applies.

This topic is not part of the previous section because the previous section covered how to use LSt web service operations and there is not an LSt web service operation to provide upstream communication. However, LSt functionality includes a proxy to help the client application receive mobile-originated messages.

In these cases, an LSt proxy will receive the mobile-originated message and forward it to your client application. Your client application would typically carry out some service and respond back to the mobile phone with an SMS or WAP message using one of the LSt Web Services described in the previous section. But receiving the initial communication from the mobile phone cannot be implemented as a web service since it is initiated by the mobile phone rather than by your client application. In this case, your client application has to be waiting to receive the mobile-originated message. It can be implemented as an http servlet, which waits for an http POST. The LSt proxy will forward the mobile-originated message to your client application servlet as an http POST.

The AAI Specification describes the contents of the http Post your client application servlet should receive.

An example client application servlet is included in the MethodA/src directory of the Openwave SDK package. Look at the file MethodA/src/LstDemoServlet.java. This example is included in the MethodA subdirectory because it uses the Openwave java classes provided in the locationstudio_client.jar file. As you can see, the init() method of this servlet reads variable values from the example_variables.property file and then waits for a request to be posted to its doPost() method. The doPost() method looks at the parameters of the posted request and uses them in a call to the LSt getLocation Web Services operation. The getLocation response gives you the location of the mobile phone that initiated the communication. This location can then be used by your client application servlet to provide some location-based service. The example servlet then sends an SMS message back to the mobile phone by using the sendSmsMessage web service operation provided by LSt.

The Openwave Demo Host

Openwave provides a demonstration instance of LSt running on an Openwave host. This demo installation is made available to facilitate the development and testing of new client applications that want to use Location Studio's Web Services.

Using the demo host, you can develop and test your new client application before you have access to your own LSt installation. You can also use the demo host to help you debug problems in your own LSt installation. The test host also provides a set of test data.

For information on accessing the test host, please go to http://developer.openwave.com/prod_tech/locationstudio.html.

The LSt 2.0 SDK Download Package

This section describes the files for the Location Studio SDK document on the LSt SDK Website. All of the files are contained in a single ZIP file. The contents are described in the following sections.

Location Studio SDK

This document (with file name lsdg_20a_001.doc).

LST AAI ICD

This Word document contains the Advanced Applications Interface Specification, reference [9].

MethodA and MethodB Directories

The files for methods A and B are organized in two directories: MethodA and MethodB. The contents of each directory is described in the following tables.

Table 5 MethodA directory files

lib	Contains the LSt and Glue jar files.
lib/locationstudio_client.jar	This jar file contains .class files for the Openwave java classes that simplify the use of the AAI.
lib/glue .jar files	These .jar files are redistributed from The Mind Electric. They are included here because a client application that uses locationstudio_client.jar needs to use the same Glue jar files that were used in the creation of locationstudio_client.jar. The following Glue archives are included here: GLUE-STD.jar, jnet.jar, jsse.jar xerces.jar and servlet.jar.
javadocs	Contains the html files that make up the javadocs for the Openwave java classes in locationstudio_client.jar.
src	<p>This subdirectory contains Openwave example client applications. There one example client application for each of the AAI operations. Source code files are named Example*.java where * is the name of an AAI operation. For example, to find the example client application code that demonstrates the use of the sendSmsMessage operation, look in the source code file named ExampleSendSmsMessage.java.</p> <p>Source code for the example client application servlet discussed in the "Receiving Mobile-Originated Messages" section is also included here.</p> <p>Source code for a minimal client application, named nullTest.java is also contained here. It demonstrates LSt output if no AAI parameters are set. This application can be used to verify that communications can be established with your LSt installation.</p>

example_variables.properties	This properties file contains values for each variable used by all the example client applications. By default this file is read by each of the example client applications.
makefile	A gmake input file to compile all the example client applications in a UNIX environment.
build.xml	An ant buildfile to compile all the example client applications in both UNIX and Windows environments.
runExamples.sh	A shell script that can be used to run any of the example client applications in a UNIX environment.
runAllExamples.sh	A shell script that will run all the example client applications one after another in a UNIX environment.
makefile.bat	A bat file to compile all the example client applications in a Windows environment.
runExamples.bat	A bat file that can be used to run any of the example client applications in a Windows environment.
runAllExamples.bat	A bat file that will run all the example client applications one after another in a Windows environment.

MethodB Directory

The MethodB directory contains the following subdirectories and files.

Table 6 MethodB directory files

src	Contains examples of Openwave client applications. There one example client application for each of the AAI operations. Source code files are named Example*.java where * is the name of an AAI operation. For example, to find the example client application code that demonstrates the use of the sendSmsMessage operation, look in the source code file named ExampleSendSmsMessage.java.
do_wsdl2java.sh	An example shell script that shows how to use Glue's <i>wsdl2java</i> utility in a UNIX environment.
do_wsdl2java.bat	An example bat file that shows how to use Glue's <i>wsdl2java</i> utility in a Windows environment.
example_variables.properties	This properties file contains values for each variable used by all the example client applications. By default this file is read by each of the example client applications.
makefile	A gmake input file to compile all the example client applications in a UNIX environment.

build.xml	An ant buildfile to run the do_wsdl2java script and compile all the example client applications in both UNIX and Windows environments.
runExamples.sh	A shell script that can be used to run any of the example client applications in a UNIX environment.
runAllExamples.sh	A shell script that will run all the example client applications one after another in a UNIX environment.
Makefile.bat	A bat file to compile all the example client applications in a Windows environment.
runExamples.bat	A bat file that can be used to run any of the example client applications in a Windows environment.
runAllExamples.bat	A bat file that will run all the example client applications one after another in a Windows environment.
example_aai.wsdl	<p>An example of the WSDL defining the LSt Web Services. Note that this file should NOT be used as input to the wsdl2java utility program and is included here as an example only. Rather, the input to wsdl2java should always be the URL of the actual LSt installation you will be using. To easily view the WSDL that your LSt installation provides, simply browse to the URL, which should be similar to the following:</p> <p>http://host:8080/locationstudio/webservices.aai.wsdl</p> <p>Where <host> is replaced with the name or IP address of the host on which your LSt installation is running. Most browsers will display the resulting WSDL in a readable form. Again example_aai.wsdl is provided here as an example only and should not be used to generate client classes.</p>

Negotiating an Agreement with a Mobile Operator

Although the variety of services that can be created and location enabled is limited only by the imaginations of the application developers, the mobile carrier has complete control over what kinds of services are delivered in their network, and which developer partners have access to the location infrastructure. This means that you must arrange for a carrier to provide support for your application, and to agree on terms for revenue sharing. There are many models for this kind of agreement. One common arrangement is for the carrier to provide access to your service and to bill subscribers a monthly premium (subscription pricing) for access to a single or bundle of services. In exchange, the carrier may provide a percentage of the price charged per active subscriber per month.

Before you can start offering a location-enhanced service, the carrier needs to do the following:

- Purchase and configure location-serving infrastructure (such as Openwave's Location Manager and Location Studio products)
- Provide you with a username and password in their network, specific to your application
- Provide connectivity to LSt if it is not openly accessible to the Internet
- Provide a URL to a test server where you can "certify" your application in their network (including location requests and services required such as SMS or WAP bearers)
- Provide a URL to the commercial location server that will be used when you launch your application
- Arrange for billing and revenue sharing as applicable to the terms arranged

Troubleshooting

Where to Find More Information About Errors

If your client application is receiving error responses from LSt, look at the Error Codes section of the AAI specification, reference [9] and browse the javadocs for the classes in `locationstudio_client.jar` that are provided by Openwave. Even if you are not using the Openwave classes in `locationstudio_client.jar`, these classes include java constants for error codes that are documented in the associated javadocs. If you generated your own java classes using the LSt WSDL, your generated classes are probably similar to those provided by Openwave and browsing Openwave's javadocs may provide help.

Where to Find More Information About Glue Errors

First remember that if you are using Method A, you must use the Glue .jar files that are provided by Openwave in the LSt SDK package. If you attempt to use the Openwave classes provided in `locationstudio_client.jar` with other Glue .jar files, you may experience a version incompatibility problem. Be sure to use the correct Glue .jar files.

If you are still experiencing errors reported by Glue, check the Glue documentation at The Mind Electric website for more help. The top level Glue link is <http://www.themindelectric.com/glue/index.html>. From there, select the “Developers” link. The “users guide” link takes you to a large, well-written set of documentation about all aspects of Web Services. While the majority of this documentation is for web application *server* developers, the section titled “Invoking Webservices” is written specifically for you, the client application developer. This is recommended reading, even if you think you know what you’re doing. The “Binding and Invoking” subsection of the “Invoking Webservices” section tells you to use the `wsdl2java` tool to generate client classes from WSDL.

Note that the Glue link shown above and subsequent link names at the Glue website were correct at the time this document was written. If The Mind Electric reorganizes their website, these link names may no longer be correct.

Use the Example Client Applications for Debugging

If your client application is getting unexpected results from LSt, you can use one or more of the example client applications to test out your data values. Edit the variables in the properties file to use your data values and then run the example client application to see what results it prints.

Bad Data Values

If your client application is getting unexpected results from LSt, be sure to verify that your client ids, client passwords and subscriber ids have been properly provisioned in your LSt web server. Also verify that permission relationships have been properly provisioned between your clients and subscribers. A common source of errors is incorrectly provisioned permission relationships between clients and subscribers. Refer to the LSt user's guides, whose voluminous information cannot be repeated here.

Getting More Diagnostics from Glue

If you are using Glue as described previously in methods A and B, you can configure Glue to output more diagnostic information. To do this, first find Glue's config.xml file. Its location will depend on which method you are using:

- If using Method A, you will have to use the config.xml file that is contained in the locationstudio_client.jar file. First, go to the MethodA./lib subdirectory and extract the file as follows:

```
cd MethodA/lib; jar xf locationstudio_client.jar electric/common/WEB-INF/config.xml
```

Now you should find Glue's config.xml file in MethodA/lib/electric/common/WEB-INF.

- If using Method B, it should be located in \$GLIB/common/WEBINF. In the previous examples of compiling with Glue libraries, we set \$GLIB to /export/GLUE3.2.2/electric/lib because we installed Glue version 3.2.2 into /export/GLUE3.2.2/electric. Therefore we should find Glue's configuration file in /export/GLUE3.2.2/electric/common/WEBINF/config.xml.

Now that you have found the Glue config.xml file, look in it for a section titled LOGGING, which contains lines that look like

```
<!--<log>EXCEPTION</log>-->
```

Most of these lines are originally commented out (using the leading “<!--“ and trailing “-->”) turn the line into a comment. By removing the comment tags in this section, Glue will generate more diagnostic information. For example, if you un-comment the EXCEPTION line to look like the following:

```
<log>EXCEPTION</log>
```

Then, Glue will print out much more information about exceptions, which may help you identify where a problem originates. If you remove the comment tags for the SOAP line, you can see the actual SOAP messages that are sent between the client application and LSt, which can help you determine if a problem occurred before the SOAP message was sent and if the client application actually received the SOAP message in response from LSt.

If you remove the comment tags for the MAPPING line, you can see if all the expected .map files are read or not. If using method B, Glue requires that the file AAIServices.map be read. You can verify that your client application is really finding and reading this map file.

After editing the Glue config.xml file, you can simply rerun your client application and the additional diagnostic information will be written to standard out. There is no need to recompile your application code.

Glue wsdl2java and Registry.bind

If you followed method B described above, you used the Glue *wsdl2java* utility to generate client side code from LSt's WSDL and then your client application used Glue's Registry.bind method to connect to LSt at run time. Be sure that you use the same URL in both cases. You cannot generate code from a WSDL file that does not exactly match the WSDL that LSt will use at run time.

If you used Method B and generated your java classes using wsdl2java, you may have generated your classes using the LSt test installation on the Openwave demo host. If so, then you can use ONLY the same LSt installation when running your client application. If you want to run your client application using your own LSt installation, then you must go back and regenerate your java classes using your own LSt installation.

See also the next section about WSDL URLs vs. the connection URL.

In the MethodB subdirectory of the SDK package, there is an example WSDL file. This is provided as an example only and should not be used to generate client code for your LSt installation.

This does not imply that Openwave will be constantly changing its Web Services interface definition. This implies simply that URLs used at the soap level are embedded in the WSDL file and these may be installation-dependent.

WSDL URL vs. Connection URL

In some LSt installations, the URL that your client application needs to connect to and the URL that serves up the WSDL may not be the same. For example, your client application may need to connect to a proxy between it and LSt. In this case, the URL that servers up the WSDL is LSt's Web Services URL but the connection URL is that of a proxy servlet. As another example, in some LSt installations, your client application may need to connect to a load-balancing servlet rather than connecting directly to LSt. In this case, the load-balancing servlet manages traffic for several LSt instances and your client connects to the servlet rather than with a particular LSt instance.

In both these examples, your client application needs to use what Glue calls an "endpoint override." This concept is explained in the "Endpoint Override" subsection of the "Invoking Web Services" section of the Glue User's Guide, reference [2]. For example, if the ExampleAuthenticateClient application needs to connect to a load-balancing servlet using Glue's "endpoint override", the registry bind code might look like the following:

```
String lstUrl =
    "http://localhost:8080/locationstudio/webservices/aai.wsdl";
String proxyUrl = "http://localhost:8080/loadbalanceproxy";
Context context = new Context();
Context.setProperty( "endpoint", proxyUrl);
IWVValidation vif = (IWVValidation)Registry.bind( lstUrl,
    IWVValidation.class, context);
```

By using a Context object, setting the "endpoint" property of that Context and passing the Context as the third argument to the Glue Registry.bind file, Glue will find the WSDL at LSt's Web Services URL but connect to the load-balancing servlet at a different URL.

SSL—Secure Socket Layer

If LSt is hosted with HTTPS enabled and the server has a certificate signed by a known Certificate Authority such as Verisign or Thawte, then communication between your client application and LSt can take place over a secure connection simply by changing http: to https: in all URLs.

However, if the server uses a self-signed certificate, e.g. for testing purposes, then that certificate has to be imported into the cacerts file in the JRE using the keytool utility. See the keytool documentation on your system for additional help.

Glossary

Term	Definition
AAI	Advanced Application Interface, see reference [9].
AFLT	Advanced Forward Link Trilateration
AGPS	Assisted Global Positioning System
API	Application Programming Interface
CDMA	Code division multiple access
Client	A client in LSt is an entity that wishes to use an LSt service, for example, a client attempts to get the location of a subscriber. Clients are uniquely identified in LSt by a client id and each client is provisioned with a password. LSt manages the permission relationships between clients and subscribers.
Client Application	A client application is a program or browser that attempts to use the AAI to access LSt Web Services. All attempts by a client application to use an LSt web service must include a client id and password. LSt will not carry out a request from a client application if the associated client ID and password are not recognized by LSt or if LSt determines that the client does not have permission to make such a request.
Community Services	An application that has relations between different subscribers. That is, a certain subscriber's actions causes the positioning of another subscriber.
EOTD	Enhanced Observed Time Difference
Fleet Services	An application where an independent third party is allowed to position a certain group of subscribers.
GLUE	Glue is a 3rd party package from The Mind Electric that can be used to simplify development of a client.
GMLC	Gateway Mobile Location Center
GLUE	Glue is a third party package from The Mind Electric that can be used to simplify development of a client.
GSM	Global System for Mobile communications
HTTP	Hypertext Transport Protocol
Information Services	An application where the subscriber requests information that relates only to his own position. That is, the subscriber's actions can only cause positioning of himself or herself.
IP	Internet Protocol
IPSec	Internet Security protocol for VPN

LBS	Location-Based Services
LIF MLP	The LIF MLP specification defines the parameters of a location request and the associated response. One of the LSt web service methods conforms to the LIF MLP specifications.
LMU	Location Measurement Unit
LSt	Location Studio
MIN	Mobile Identification Number
MLP	Mobile Location Protocol
MPC	Mobile Positioning Center
MS	Mobile Station
MSID	Mobile Station Identifier
MSISDN	Mobile Subscriber ISDN
OSID	Operator Subscriber ID. A subscriber ID used by clients for location requests in place of the subscriber's MSID in order to support anonymous and private data exchange between clients and subscribers. The OSID is generated in an application external to LSt, such as an operator's messaging or WAP gateway, and used synchronously through the operator's identification manager.
PDE	Position Determination Equipment
PSID	Persistent Subscriber ID. A subscriber ID used by clients for location requests in place of the subscriber's MSID in order to support anonymous and private data exchange between clients and subscribers. Typically, PSIDs are used for passive positioning requests. Each PSID is generated in LSt, and used only once with a limited lifetime.
QoP	Quality of Position
SDK	Software Development Kit. Openwave provides this document and additional files downloadable from the Openwave LSt 2.0 SDK website to assist developers with implementation of client applications that use the LSt Web Services. See a later section of this document for a complete list of what is downloadable from the website.
SLIR	Standard Location Immediate Request
SMLC	Serving Mobile Location Center
SSL	Secure Socket Layer
Subscriber	A subscriber is an LSt entity that is typically the target of an AAI request. For example, you can request the location of a subscriber or send a message to a subscriber. A subscriber is uniquely identified by a subscriber id, which includes both a number and type. See section 8 of the AAI Specification, reference [9], for a

	description of the various subscriber types. Any AAI request that involves a subscriber must include a valid subscriber id. For example, if a client application wants to send a message to a mobile phone, the application can use the sendSmsMessage Web Service request and pass it the subscriber ID associated with the mobile phone.
Subscriber identity	Refers to MIN, IMSI, MSISDN, and MDN, as well as the LSt Identifiers TSID, PSID and OSID.
TDMA	Time Division Multiple Access
TSID	Temporary Subscriber ID. A subscriber ID used by clients for location requests in place of the subscriber's MSID in order to support anonymous and private data exchange between clients and subscribers. Typically, TSIDs are used when subscribers are actively engaged with a client. Each TSID is generated in LSt, and used only once with a limited lifetime.
URL	Uniform Resource Locator
VPN	Virtual Private Network
WAP	Wireless Application Protocol
WSDL	Web Services Definition Language—Web Services provided by a particular LSt instance are precisely defined by a WSDL file which is accessible by browsing to http://host:8080/locationstudio/webservices/aai.wsdl where host is the hostname on which the LSt instance is running. Note that the LSt port number is typically 8080 but may be different on your LSt installation.