Oracle® Communications Diameter Signaling Router

Subscriber Data Server Provisioning Interface

E57485 Revision 01

September 2013

ORACLE®

Oracle® Communications Subscriber Data Server Provisioning Interface

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Topics:

This chapter contains general information about the XML/SOAP provisioning documentation, the organization of this manual, and how to get technical assistance.

Overview

This document presents the Representational State Transfer (REST) Provisioning interface to be used by local and remote provisioning client applications to administer the Provisioning Database of the Oracle Communications User Data Repository (UDR) system. Through REST interfaces, an external provisioning system supplied and maintained by the network operator may add, change, delete or retrieve subscriber/pool information in the UDR database.

Manual Organization

This document is organized into the following chapters:

- *Introduction* contains general information about the SDS documentation, the organization of this manual, and how to get technical assistance.
- *System Architecture* gives an overview of XML/SOAP system architecture.
- *Interface Description* provides a high level overview of the interface provided by the XML Data Server (XDS) and the SOAP server.
- *SOAP Message Definitions* describes the SOAP operations syntax and parameters.
- *XML Message Definitions* describes XML requests and responses syntax and parameters.
- *SDS Response Message Error Codes* describes the XML/SOAP error codes that are returned by the XDS/SOAP server.
- *XML/SOAP Interface System Variables* describes the XML/SOAP interfaces that have a set of system variables that affect the operation as it runs.
- *Database Object Model* describes the database object model and shows all tables associated with SDS provisioning.

Scope and Audience

This manual is intended for customers, Tekelec customer service, software development, and product verification organizations, and any other Tekelec personnel who need to understand the XML or SOAP interfaces. Users of this manual and the others in the SDS family of documents must have a working knowledge or telecommunications and network installations.

Documentation Admonishments

Admonishments are icons and text throughout this manual that alert the reader to assure personal safety, to minimize possible service interruptions, and to warn of the potential for equipment damage.

Table 1: Admonishments

Icon	Description
	**Danger**: (This icon and text indicate the possibility of *personal injury*.)
	**Warning**: (This icon and text indicate the possibility of *equipment damage*.)
	**Caution**: (This icon and text indicate the possibility of *service interruption*.)
	**Topple**: (This icon and text indicate the possibility of *personal injury* and *equipment damage*.)

My Oracle Support (MOS)

MOS (*https://support.oracle.com*) is your initial point of contact for all product support and training needs. A representative at Customer Access Support (CAS) can assist you with MOS registration.

Call the CAS main number at **1-800-223-1711** (toll-free in the US), or call the Oracle Support hotline for your local country from the list at *http://www.oracle.com/us/support/contact/index.html*. When calling, make the selections in the sequence shown below on the Support telephone menu:

1. Select **2** for New Service Request
2. Select **3** for Hardware, Networking and Solaris Operating System Support
3. Select one of the following options:
   - For Technical issues such as creating a new Service Request (SR), Select **1**
   - For Non-technical issues such as registration or assistance with MOS, Select **2**

You will be connected to a live agent who can assist you with MOS registration and opening a support ticket.

MOS is available 24 hours a day, 7 days a week, 365 days a year.

Emergency Response

In the event of a critical service situation, emergency response is offered by the Customer Access Support (CAS) main number at **1-800-223-1711** (toll-free in the US), or by calling the Oracle Support hotline for your local country from the list at *http://www.oracle.com/us/support/contact/index.html*. The emergency response provides immediate coverage, automatic escalation, and other features to ensure that the critical situation is resolved as rapidly as possible.

A critical situation is defined as a problem with the installed equipment that severely affects service, traffic, or maintenance capabilities, and requires immediate corrective action. Critical situations affect service and/or system operation resulting in one or several of these situations:

- A total system failure that results in loss of all transaction processing capability
- Significant reduction in system capacity or traffic handling capability
- Loss of the system's ability to perform automatic system reconfiguration
- Inability to restart a processor or the system
- Corruption of system databases that requires service affecting corrective actions
- Loss of access for maintenance or recovery operations
- Loss of the system ability to provide any required critical or major trouble notification

Any other problem severely affecting service, capacity/traffic, billing, and maintenance capabilities may be defined as critical by prior discussion and agreement with Oracle.

Related Publications

The Diameter Signaling Router (DSR) documentation set includes the following publications, which provide information for the configuration and use of DSR and related applications.

Getting Started includes a product overview, system architecture, and functions. It also explains the DSR GUI features including user interface elements, main menu options, supported browsers, and common user interface widgets.

Feature Notice describes new features in the current release, provides the hardware baseline for this release, and explains how to find customer documentation on the Oracle Customer Support Site.

Roadmap to Hardware Documentation provides links to access manufacturer online documentation for hardware related to the DSR.

Operation, Administration, and Maintenance (OAM) Guide provides information on system-level configuration and administration tasks for the advanced functions of the DSR, both for initial setup and maintenance.

Communication Agent User's Guide explains how to use the Communication Agent GUI pages to configure Remote Servers, Connection Groups, and Routed Servers, and to maintain configured connections.

Diameter and Mediation User's GuideDiameter User's Guide explains how to use the Diameter GUI pages to manage the configuration and maintenance of Diameter Configuration components, including Local and Peer Nodes, Connections, Configuration Sets, Peer Routing Rules, Application Route Tables,

System Options, and DNS options; describes the functions of Diameter Message Copy; explains how to configure and use Diameter Mediation; and describes DSR capacity and congestion controls.

Diameter Mediation User's Guide describes the functions of Diameter Mediation, and explains how to use the Diameter Mediation GUI pages (nested inside the Diameter GUI folder) to configure and test Rule Templates, how to use the Formatting Value Wizard, and how to configure Rule Sets.

IP Front End (IPFE) User's Guide explains how to the use the IPFE GUI pages to configure IPFE to distribute IPv4 and IPv6 connections from multiple clients to multiple nodes.

Range-Based Address Resolution (RBAR) User's Guide explains how to use the RBAR GUI pages to configure RBAR to route Diameter end-to-end transactions based on Diameter Application ID, Command Code, Routing Entity Type, and Routing Entity address ranges and individual addresses.

Full-Address Based Resolution (FABR) User's Guide explains how to use the FABR GUI pages to configure FABR to resolve designated Diameter server addresses based on Diameter Application ID, Command Code, Routing Entity Type, and Routing Entity addresses.

Charging Proxy Application (CPA) and Offline Charging Solution User's Guide describes the Offline Charging Solution and explains how to use the CPA GUI pages to set System Options for CPA, configure the CPA's Message Copy capability, and configure the Session Binding Repository for CPA.

Policy DRA User's Guide describes the topology and functions of the Policy Diameter Routing Agent (Policy DRA) DSR Application and the Policy Session Binding Repository, and explains how to use the GUI pages to configure Policy DRA.

Gateway Location Application (GLA) User's Guide describes the functions of retrieving subscriber data stored in Policy Session Binding Repository (pSBR) provided by Policy DRA and explains how to use the GUI pages to configure GLA.

DSR Alarms, KPIs, and Measurements Reference provides detailed descriptions of alarms, events, Key Performance Indicators (KPIs), and measurements; indicates actions to take to resolve an alarm, event, or unusual Diameter measurement value; and explains how to generate reports containing current alarm, event, KPI, and measurement information.

DSR Administration Guide describes DSR architecture, functions, configuration, and tools and utilities (IPsec, Import/Export, DIH, and database backups); and provides references to other publications for more detailed information.

Locate Product Documentation on the Oracle Technology Network Site

Oracle customer documentation is available on the web at the Oracle Technology Network (OTN) site, *http://docs.oracle.com*. You do not have to register to access these documents. Viewing these files requires Adobe Acrobat Reader, which can be downloaded at *www.adobe.com*.

1. Log into the Oracle Technology Network site at *http://docs.oracle.com*.
2. Select the **Applications** tile.
   The **Applications Documentation** page appears.
3. Select **Apps A-Z**.
4. After the page refreshes, select the **Communications** link to advance to the **Oracle Communications Documentation** page.
5. Navigate to your Product and then the Release Number, and click the **View** link (note that the Download link will retrieve the entire documentation set).

6. To download a file to your location, right-click the **PDF** link and select **Save Target As**.

Chapter 2

System Architecture

Topics:

This chapter provides an overview of XML/SOAP system architecture.

System Architecture Overview

Figure 1: SDS Architecture Overview provides an overview of the SDS architecture.

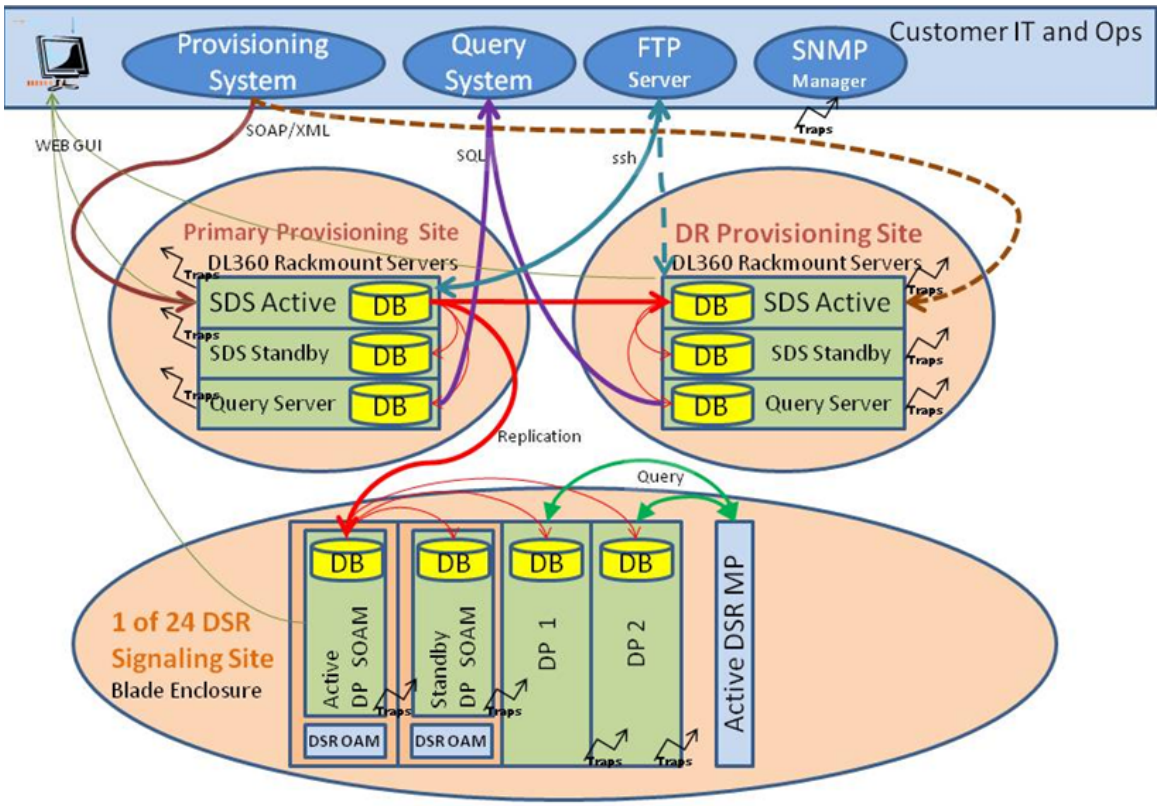

Figure 1: SDS Architecture Overview

The SDS system consists of a Primary Provisioning Site, a Disaster Recovery (DR) Provisioning Site, and up to 24 DSR Signaling Site servers with redundant data processor Site Operation Administration and Maintenance (SOAM) servers and up to 2 data processing blades. Each provisioning site has an active/standby pair of servers in a High Availability (HA) configuration and a third server configured as a Query Server.

The SDS system is built on a platform that provides a variety of services such as site-based GUI, HA capabilities (active/standby switchover and disaster recovery switchover), and database functionality (replication, backup, restore).

Every server within the SDS system collects measurements, alarms, and events data. Every server sends its traps directly to the Customer SNMP Manager.

Every server can also collect measurement data. Data processing measurements are sent to the active SOAM server, which sends the measurements from all data processing servers and itself to the Active SDS Server on the Primary Provisioning Site. The measurements can be viewed on the GUIs for the Active SDS Server on the Primary Provisioning Site and the DP SOAM server on the DSR Signaling Site.

SDS/HLRR Architecture Overview

Figure 2: SDS/HLRR Architecture Overview shows a high level overview for the SDS, HLRR and DSR products.

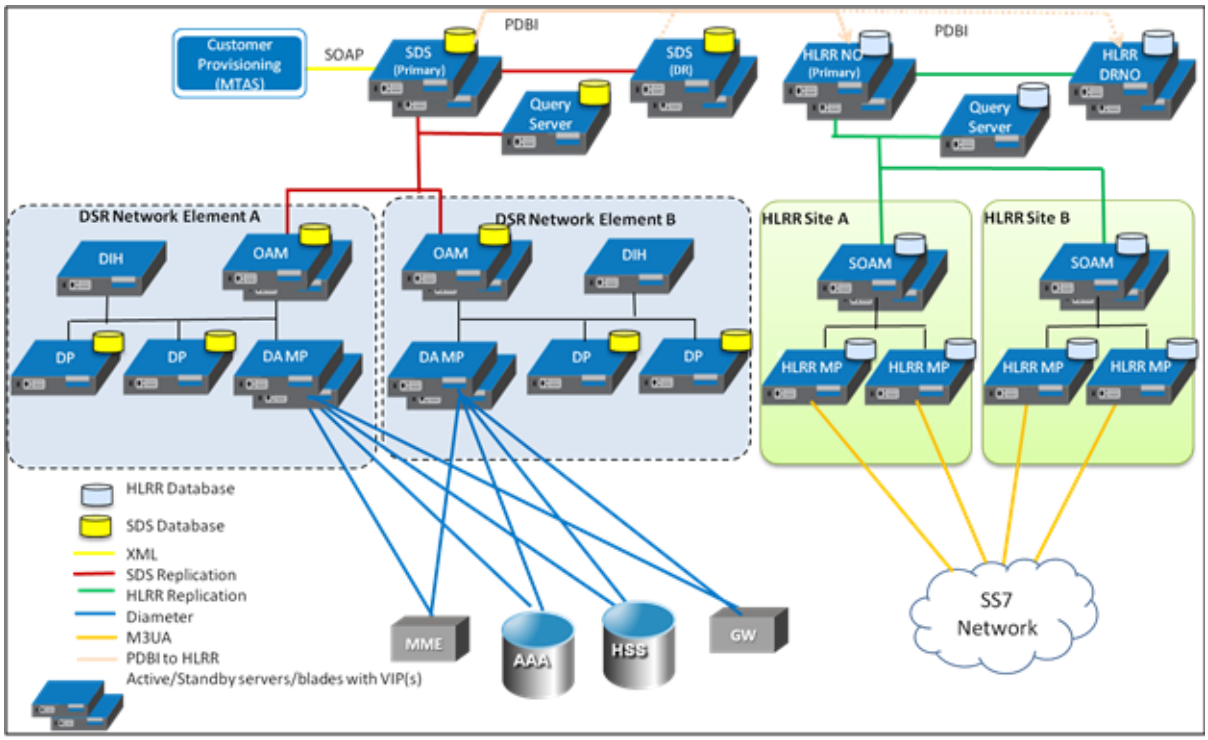

Figure 2: SDS/HLRR Architecture Overview

The SDS for HLRR components consist of an SDS Primary Provisioning Site, an SDS DR Provisioning Site, an HLRR Primary NO Site and an HLRR Disaster Recovery NO Site. The SDS sites replicate data to other SDS systems (such as Query Server and DP SOAM). The HLRR sites replicate data to other HLRR systems (such as Query Server and SOAM).

The SDS Primary Provisioning Site relays specific data to the HLRR Primary NO site and can send data to the HLRR DR NO site. The SDS DR Provisioning Site can also relay specific data to the HLRR Primary or DR NO site, and will do so if it becomes the SDS Primary Provisioning Site.

The type of data that can be relayed and additional information on this process is described in *Relaying data to the HLR Router*. The data is relayed over HLRR PDBI interface by the **pdbrelay** and **pdbaudit** (for remote audit) processes.

Customer IT and Ops

The Customer IT and Ops layer contains the customer provisioning system, query system, Web GUI, FTP Server, and SNMP agent. These components belong to the customer and are external to the Tekelec SDS system. The customer is responsible for configuring their systems to connect to the SDS system.

Web GUI

The customer uses the Virtual IP address (VIP) for the Active SDS Server on the Primary Provisioning Site to access the SDS GUI and the VIP for the appropriate SOAM to access the SOAM GUI for the DSR Signaling Site Server.

To connect to the SDS application:

1. Launch Internet Explorer 7.x, 8.x or 9.x and connect to the VIP assigned to the Active SDS Server on the Primary Provisioning Site.
2. Login to the GUI using your username and password.

The VIP address of the desired server is used to connect to the Active SDS Server on the DR Provisioning Site or to an SOAM server on a DSR Signaling Site. Data can only be viewed on these servers.

Provisioning System

The customer provisioning system must be configured so that it can have SOAP and/or XML connections to the Primary and DR Provisioning sites. The provisioning system establishes active connections to the active site (usually the Primary Provisioning Site). The VIP addresses are used to connect to the Primary and DR Provisioning sites. The SOAP and/or XML ports can also be configured.

Query System

A MySQL client must be installed on the customer system. The customer system can connect to the Query servers on the Primary and/or DR Provisioning Sites using the Query server IP address and IP port=15616.

FTP Server

The customer FTP server is used by the import and export processes and to store performance data.

SNMP Manager

The customer SNMP Manager is used to accept traps for the servers. All servers send SNMP traps to the SNMP Manager for alarms and events.

Primary Provisioning Site

The Primary Provisioning Site is used for SDS OAM. All provisioning is done using a Web GUI or from the customer provisioning system, using a SOAP and/or XML interface.

The Primary Provisioning Site uses three rack mount servers:

- Active SDS Server
- Standby SDS Server
- Query Server

Each server has the identical software but a different role. Only the SDS server runs the XML Data Server (XDS) and SOAP Server applications. These applications run within the same process referred to as XDS.

Active SDS Server

The Active SDS Server on the Primary Provisioning Site accepts input from the Web GUI and from a SOAP and/or XML interface. The Active SDS Server is responsible for applying all database updates (adds, changes, and deletes) and replicating appropriate updates to the:

- Primary Provisioning Site Standby SDS Server
- Primary Provisioning Site Query Server
- DR Provisioning Site Active SDS Server
- All subtending DP SOAMs at the DSR Signaling Sites

The Active SDS Server on the Primary Provisioning Site provides a GUI which is used for configuration, user administration, and viewing of alarms and measurements. The Active SDS Server distributes all successful incoming subscriber provisioning data, independent of source, to all downstream Network Elements (Query Server and DP SOAMs on the DSR Signaling Sites) and the DR Provisioning Site.

To ensure that the database levels of the Network Elements are no more recent than the database levels of the SDS Servers on the Primary and DR Provisioning Sites, the Active SDS Server on the Primary Provisioning Site provisions the Active SDS Server on the DR Provisioning Site prior to updating the DSR Signaling Sites (DP SOAM and DPs).

Both the Active and Standby SDS Servers share a VIP address. The Active SDS Server owns the VIP address. If the current Standby SDS Server becomes active, it acquires the VIP address.

Standby SDS Server

The Standby SDS Server receives updates from the Active SDS Server, keeping the Active SDS Server and Standby SDS Server in sync. If the Active SDS Server fails, then the Standby SDS Server automatically performs a switchover, becomes the Active SDS Server, and acquires the VIP address.

Query Server

The SDS Query Server provides a secure MySQL interface that allows the customer to query subscriber data using the previously configured *Query System*.

The SDS Query Server accepts replicated subscriber data from the Active SDS Server and stores it in a customer-accessible MySQL database. The SDS Query Server provides a free-form read-only query capability using the MySQL interface and limited MySQL user management. The SDS Query Server is located in the same physical frame as the SDS server components at the Primary and DR Provisioning Sites.

Disaster Recovery Provisioning Site

The Disaster Recovery (DR) Provisioning Site is an SDS Provisioning Site. Configuring a DR Provisioning Site is optional. If the site is configured, then a geo-diverse DR Provisioning site is recommended.

The DR Provisioning Site has the same hardware configuration and network accessibility as the Primary Provisioning Site. The Primary and DR Provisioning Sites have different VIP addresses for their Active SDS Servers.

The Active SDS Server on the DR Provisioning Site accepts updates from the Active SDS Server on the Primary Provisioning Site. The DR Provisioning Site does not have an active SOAP or XML connection opened. This connection can be established when the DR Provisioning Site is promoted to be the Primary Provisioning Site.

DR Provisioning Site databases are kept current through real-time replication of subscriber and application data from the Active SDS Server on the Primary Provisioning Site. Under normal operating conditions, the Active SDS Server on the DR Provisioning Site does not provision any downstream systems. If this server is made Active, then the server takes over all functions of the Active SDS Server on the Primary Provisioning site, including the provisioning interfaces and database replication to subtending SOAMs. If the Active and Standby SDS Servers on the Primary Provisioning Site fail, then the customer must manually force a switchover to the Active SDS Server on the DR Provisioning Site.

DP SOAM

The data processing (DP) SOAM is the single point of entry for the replication stream of subscriber data into a DSR Signaling Site. The DP SOAM consists of a combination of an active and a standby server running the DP SOAM application and operating in a high availability configuration.

The active DP SOAM Server receives subscriber data replicated from the Active SDS Server on the Primary Provisioning Site and replicates the data to the standby DP SOAM Server and to all subtending data processors located in the same physical frame. Provisioning data, alarms, and measurements can be viewed or queried using a GUI connected to the VIP address for the DP SOAM.

Data Processors

Data Processors are servers that are configured for DP functionality. These servers accept replicated subscriber data from the local DP SOAM and store it in a subscriber database. The data processors are used for processing queries from the DSR Message Processor for destination address resolution.

The data processor receives database queries that include user identities such as MSISDN, IMSI, or URI and destination types and returns the resolved destination address FQDN and/or realm values.

Each DSR Signaling Site can support up to 10 DP servers deployed in a single frame in order to scale query capacity. Two DP servers are supported currently.

Each DP server contains a copy of the same SDS data, configured in an active/active mode. The DSR Message Processor is responsible for load-balancing requests across DP servers.

Each DP server runs on a HP C-Class blade. The DP Server is configured in an active/active mode and is deployed at each DSR Signalling Site on blades with *n+m* redundancy. Initially, *n=1* and *m=1*.

A GUI is not available for DP servers. A GUI can be connected to the DP SOAM VIP address to view or query provisioning data, alarms, and measurements.

Chapter 3

Interface Description

Topics:

This chapter provides an overview of the interface provided by the XML data server and the SOAP server.

Provisioning Interface Overview

Data can be provisioned or exported using one of the following interfaces:

- GUI - Add, change, delete, and query routing-related data and internal tables. Internal tables contain information such as NPA-NXX split data and export schedules.
- SOAP - Add, change, delete, and read MSISDN, IMSI and NAI user data.
- XML - Add, change, delete, and read MSISDN, IMSI and NAI user data.
- SQL - View or query routing-related data. This interface is provided by the SDS Query server.
- Import - Import data in CSV or XML format.
- Export - Export data in CSV, XML, or HLRR format.
- PDB Relay - Add, change, and delete MSISDN and IMSI routing entities with destinations that have an E.164 network entity value on SDS. Automatically send those provisioning commands from SDS to HLR Router (HLRR)

The method used to provision data varies, depending on the type of data. All data can be provisioned using the GUI. *Table 2: Data Provisioning Interfaces* shows which interfaces are available for each type of data.

Table 2: Data Provisioning Interfaces

Data Type	GUI	SOAP/XML	XML Import/Export	CSV Import/Export	HLRR Export and PDB Relay	SQL Query
MSISDN	yes	yes	yes	yes	yes	yes
IMSI	yes	yes	yes	yes	yes	yes
NAI User	yes	yes	yes	yes	no	yes
Wildcard NAI User	yes	no	no	yes	no	yes
NAI Host	yes	no	no	yes	no	yes
Destination	yes	no	no	yes	no	yes
Destination Map	yes	no	no	no	no	yes
MSISDN Prefix	yes	no	no	yes	no	yes
IMSI Prefix	yes	no	no	yes	no	yes
MSISDN Blacklist	yes	no	no	yes	no	yes
IMSI Blacklist	yes	no	no	yes	no	yes
Subscriber	yes	yes	yes	yes	no	yes

Note: Only MSISDN and IMSI routing entities with destinations that have an E.164 network entity value can be relayed or exported in HLRR format.

A history of the commands and their responses can be viewed from the SDS GUI. See the *SDS Online Help* for more information. All provisioning requests are stored in the Command Log for all interfaces.

Customer Provisioning System to SDS Overview

Each SDS server has identical software. Two of the applications that exist on each SDS server are the XML Data Server and the SOAP Server. These applications run within the same process named XDS.

The customer's provisioning system must be configured so that it can have SOAP and/or XML connections to the Primary and Disaster Recovery (DR) Provisioning sites by using the Virtual IP (VIP) address for each site. The SOAP and/or XML ports are also configurable.

The customer's provisioning system only establishes active connections to the active site (usually the Primary Provisioning Site). In the event of a failure of the Active SDS Server, the Standby SDS Server is activated, and the VIP is moved over to that server.

In the event of a failure of the Primary Provisioning Site, the DR Site becomes active. The Client Provisioning Systems must manually switch over from the Primary SDS VIP to the DR SDS VIP.

XML Data Server

The XML Data Server runs in the XML Data Server (XDS) process on the Active SDS Server on the Primary Provisioning Site. The XML Data Server implements XML over a TCP interface.

Each XML request and response message (see *XML Message Definitions*) consists of a 4-byte binary length value, followed by the indicated number of ASCII characters that form the XML request. There is no need to terminate the XML request with any terminating character(s).

The XML Data Server is responsible for:

- Accepting and authorizing XML/TCP provisioning client connections.
- Processing and responding to XML requests received from provisioning clients.
- Updating and maintaining the provisioning database, located on the Active SDS server on the Primary Provisioning Site. MSIDN, IMSI, and NAI user routing entities can be read and provisioned, including destinations for the routing entities.

  Note: All specified destinations and NAI Hosts must already be defined (using the GUI or CSV import).

XML provisioning can occur via an XML client or an XML import file. SDS also supports exporting MSISDN, IMSI and NAI User data into an export file.

SOAP Server

The SOAP server runs in the XDS process on the Active SDS Server on the Primary Provisioning Site. The SOAP server implements SOAP over an HTTP interface.

The SOAP server is responsible for:

- Accepting and authorizing SOAP/HTTP provisioning client connections.
- Processing and responding to SOAP requests received from provisioning clients.
- Updating and maintaining the provisioning database, located on the Active SDS Server on the Primary Provisioning Site. MSISDN, IMSI, and NAI user routing entities can be read and provisioned, including the destinations for these routing entities.

  Note: All specified destinations and NAI Hosts must already be defined (using the GUI or CSV import).

Provisioning Clients

The provisioning clients, which are owned by the customer, establish TCP/IP connections to the XML Data Server or SOAP server, using the VIP for the Active SDS Server on the Primary Provisioning Site. The provisioning clients use XML or SOAP to send requests to manipulate and query data in the Provisioning Database and then process the XML or SOAP response messages.

Provisioning clients must re-establish connections with the XML Data Server or SOAP server using the Primary SDS VIP on switchover from the Primary Active to Standby SDS Server. Provisioning clients must also redirect connections to the Secondary VIP on switchover from the Primary SDS Site to the DR SDS Site.

Provisioning clients must run a timeout for the response to a request, in case a response is not sent. If no response is received, a client drops and re-establishes the connection before trying again.

Note: By dropping the connection, any transaction that is in progress on that connection is automatically rolled back. Consequently, the entire transaction must be started and resent again.

Provisioning clients are expected to re-send XML/SOAP requests for database manipulation requests that resulted in a temporary error or for which no responses were received.

The SDS GUI is used to configure connections to the provisioning clients. See the *SDS Online Help* for more information.

Security

The following forms of security are provided for securing connections between the XML/SOAP Interfaces and provisioning clients in an unsecure/untrusted network:

- Client Server IP Address White List
- Secure Connections using SSLv3 (SOAP Interface only)

Client Server IP Address White List

The XML/SOAP Interfaces maintain a list of server IP addresses that clients can use to establish a TCP/IP connection. Each IP address on the list has read-only or read/write permissions. The SDS GUI is used to administer the list. See the *SDS Online Help* for more information.

Any connect request coming from an IP address that is not on the list is denied, and the connection is immediately closed. If an IP address is removed from the list, then any active connection established from that IP address is immediately closed.

Secure Connection Using SSLv3

The SOAP Server supports secure connections between provisioning clients and the SOAP Server using the Secure Sockets Layer version 3 (SSLv3) protocol.

SSL is an industry standard protocol for clients needing to establish secure (TCP-based) SSL-enabled network connections.

SSL capabilities address several fundamental concerns about communication over TCP/IP networks:

- SSL server authentication allows a client application to confirm the identity of the server application. The client application through SSL uses standard public-key cryptography to verify that the server's certificate and public key are valid and have been signed by a trusted certificate authority (CA) that is known to the client application.
- SSL client authentication allows a server application to confirm the identity of the client application. The server application through SSL uses standard public-key cryptography to verify that the client's certificate and public key are valid and have been signed by a trusted CA that is known to the server application.
- An encrypted SSL connection requires all information being sent between the client and server application to be encrypted. The sending application is responsible for encrypting the data and the receiving application is responsible for decrypting the data. In addition to encrypting the data, SSL provides message integrity, which provides a means to determine if the data has been tampered with since it was sent by the partner application.

Depending on whether the SOAP Server is configured to operate in a secure or unsecure mode, provisioning clients can connect using unsecure or secure connections to the SOAP Server TCP/SSL listening port. The SDS GUI is used to configure this functionality. See the *SDS Online Help* for more information.

Note: An SSL-enabled connection is slower than an unsecure TCP/IP connection due to providing adequate security.

SSL Certificates and Public/Private Key Pairs

SSL-enabled connections require SSL certificates. Certificates rely on asymmetric encryption (or public-key encryption) algorithms that have two encryption keys (a public key and a private key). A certificate owner can show the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted with this public key can be decrypted only using the corresponding, matching private key, which is held by the owner of the certificate.

Tekelec issues Privacy Enhanced Mail (PEM)-encoded SSL X.509v3 certificates and encryption keys to the SOAP Server and provisioning clients needing to establish an SSL-enabled connection with the SOAP Server. These files can be found on the SDS server under `/usr/TKLC/sds/ssl`. These files should be copied to the server running the provisioning client.

Table 3: SSL X.509 Certificate and Key PEM-encoded Files

Certificate and Key PEM-encoded Files	Description
tklcCaCert.pem	TEKELEC self-signed trusted root Certification Authority (CA) X.509v3 certificate.
serverCert.pem	The SOAP Servers X.509v3 certificate and 2,048-bit RSA public key digitally signed by TEKELEC

Certificate and Key PEM-encoded Files	Description
	Certification Authority (CA) using SHA-1 message digest algorithm.
serverKey.nopass.pem	The SOAP Servers corresponding, matching 2,048-bit RSA private key without passphrase digitally signed by TEKELEC Certification Authority (CA) using SHA-1 message digest algorithm.
clientCert.pem	Provisioning client's X.509v3 certificate and 2,048-bit RSA public key digitally signed by TEKELEC Certification Authority (CA) using SHA-1 message digest algorithm.
clientKey.nopass.pem	Provisioning client's corresponding, matching 2,048-bit RSA private key without passphrase digitally signed by TEKELEC Certification Authority (CA) using SHA-1 message digest algorithm.

Provisioning clients are required to send an SSL authenticating X.509v3 certificate when requested by the SOAP Server during the secure connection handshake protocol for mutual (two-way) authentication. If the provisioning client does not submit a certificate that is issued/signed by TEKELEC Certification Authority (CA), it will not be able to establish a secure connection with the SOAP Server.

Supported SSLv3 Cipher Suites

A cipher suite is a set/combination of lower-level algorithms that an SSL-enabled connection uses to do authentication, key exchange, and stream encryption. The following table lists the set of cipher suites that are supported by the SOAP Server to secure an SSL-enabled connection with provisioning clients. The cipher suites are listed and selected for use in the order of key strength, from highest to lowest. This ensures that during the handshake protocol of an SSL-enabled connection, cipher suite negotiation selects the most secure suite possible from the list of cipher suites the client wishes to support, and if necessary, back off to the next most secure, and so on down the list. Note: Cipher suites containing anonymous DH ciphers, low bit-size ciphers (currently those using 64 or 56 bit encryption algorithms but excluding export cipher suites), export-crippled ciphers (including 40 and 56 bits algorithms), or the MD5 hash algorithm are not supported due to their algorithms having known security vulnerabilities.

Table 4: SSLv3 Supported Cipher Suites

Cipher Suite	Key Exchange	Signing/Authentication	Encryption (Bits)	MAC (Hash) Algorithms
AES256-SHA	RSA	RSA	AES (256)	SHA-1
DES-CBC3-SHA	RSA	RSA	3DES (168)	SHA-1
AES128-SHA	RSA	RSA	AES (128)	SHA-1
KRB5-RC4-SHA	KRB5	KRB5	RC4 (128)	SHA-1
RC4-SHA	RSA	RSA	RC4 (128)	SHA-1

Cipher Suite	Key Exchange	Signing/Authentication	Encryption (Bits)	MAC (Hash) Algorithms
KRB5-DES-CBC3-SHA	KRB5	KRB5	3DES (168)	SHA-1

Multiple Session Connectivity

Multiple provisioning systems may be connected via the XML/SOAP Interfaces simultaneously. All systems can issue commands that do read or write. If more than one system requests to start a transaction, or issues an update/delete request, contention for write access will be handled as follows:

- The first system to submit a write request will be granted access, if it is authorized for write access.
- If a second system submits a write request while the first transaction is still open, it will either be immediately rejected with WRITE_UNAVAIL error code, or will be queued for a specified time out period to wait on the first system's transaction to complete.
- The time out period can be specified by the user in the start transaction/update/delete request. Valid value are from 0 to 3600 seconds. If the value is not included or is set to 0, the second request will be immediately rejected with WRITE_UNAVAIL error code.
- If the time out value is set to any non-zero value, the second start transaction or update/delete request will be held for that time period before being rejected. If the first user releases the transaction before the second user's time out period has expired, the second user will then be granted write access.
- If a third user submits a start transaction or update/delete request after the second user with a specified time out period, the third user's request will be queued behind the second user's request. Once the first user releases the transaction, the second user is granted access. After the second user releases the transaction, the third user is granted access and so forth. If any user's time out period expires, that request will be immediately rejected with WRITE_UNAVAIL error code.
- If the third user sets a time out period longer than the second user, and the second user's time out period expires before the first user releases the transaction, the second user's request will be dropped from the queue and the third user will move up in the queue. Thus, if the first user then releases the transaction before the third user's time out has expired; the third user will be granted access.

Request Queue Management

If multiple clients simultaneously issue requests, then each request is queued and processed in the order received on a per connection basis. The client does not have to wait for a response from one request before issuing another.

Incoming requests are not prioritized. Multiple requests from a single client are handled on a first-in, first-out basis. Generally, requests are answered in the order received. Invalid requests are responded to immediately, despite any other valid requests in the queue.

Syncronous/Asyncronous Mode

As described in *Request Queue Management*, a client that sends multiple requests before waiting for the response from a previous request is not guaranteed to receive the responses in the order they were sent.

If a client wishes to send a request before waiting for the response to the previous one (asyncronous mode), then the client must populate the `id` attribute in the request with a transaction ID value that will be passed back in the response. The `id` attribute needs to be unique enough to the client to correlate a response to a request that was sent. The XML Data Server will return the id passed in the response.

If a client wishes to send a single request and wait for the response before sending another one (syncronous mode), then the client does not need to populate the `id` attribute in the request, because the response will always be for the request last sent. The `id` attribute can be populated if desired, and it will be passed back in the response just as in asyncronous mode.

Message Processing (Transactions)

All subscription-related requests are performed within the context of a database transaction. The XML/SOAP Interfaces use a transaction-based API.

The SDS GUI is used to configure the transaction options. See the *SDS Online Help* for more information.

Transaction Modes

The XML Interface supports the following database transaction modes:

- Normal Transaction Mode
- Block Transaction Mode
- Single Transaction Mode (default)

The SOAP Interface supports the following database transaction modes:

- Normal Transaction Mode
- Single Transaction Mode (default)

The provisioning client controls which transaction mode will be used by the commands it sends.

Normal Database Transaction Mode

The normal database transaction mode requires an explicit `<startTransaction/>` request paired with `<commit/>` or `<rollback/>` request to complete the transaction.

A normal sequence of events might be:

- `<startTransaction/>`
- `<updateSubscriber … />`
- `<.../>`
- `<updateSubscriber … />`

- <commit/>

Or:

- <startTransaction/>
- <updateSubscriber … />
- <.../>
- <updateSubscriber … />
- <rollback/>

All requests within a transaction must be sent on the same TCP/IP connection, for both XML and SOAP interfaces. If the TCP/IP connection is disconnected when a transaction is in progress, the transaction is automatically rolled back.

In normal database transaction mode, many updates can be sent and committed to the database at once when the transaction is completed. This results in a much faster rate of updates per second.

Transaction integrity is ensured by allowing updates to be aborted or rolled back if there is an unexpected failure before the transaction is completed. Updates are not committed to the database until the <commit/> request is issued. If an unexpected failure occurs, or if the transaction is explicitly aborted by the <rollback/> request, the database is maintained in the state it was in prior to the beginning of the transaction.

Data across all requests performed inside a transaction is consistent. A transaction can only be opened by one client connection at a time, preventing multiple clients from updating the database at the same time.

Note: A block transaction (<tx> … </tx>) cannot be sent during a normal database transaction (i.e. after a <startTransaction/> request has been sent and before a <commit/> or <rollback/> request is sent. If a block transaction request is sent during this period, then the request is rejected with a INV_REQ_IN_NORMAL_TX error. This error does not affect or abort the open transaction.

Block Transaction Mode

The block transaction mode requires explicit <tx> tags around all of the requests in a transaction.

The block transaction is sent as one XML request, and all requests contained within the block are executed in sequence within a database transaction. If any request fails, then the entire transaction is automatically rolled back. If all requests are successful, then the transaction is automatically committed.

If a block transaction fails, then the request within the block that encountered an error will have the appropriate error code set. All requests after the failed request will have the error code set to NOT_PROCESSED. Any requests before the failed request will indicate success, and the number of affected rows.

All transactions must also satisfy limits indicated by the Max Transaction Size, Maximum Transaction Lifetime, and Transaction Durability Timeout system variables, which are defined in *XML/SOAP Interface System Variables*. If any of those limits are exceeded, the transaction is aborted and automatically rolled back.

Note: A block transaction cannot be sent in the context of a normal database transaction (i.e. after a <startTransaction/> request has been sent and before a <commit/> or <rollback/> request is sent). Normal database transaction requests, such as <startTransaction/>, <commit/> or <rollback/>, cannot be sent within a block transaction. If any normal requests are sent, then the block transaction fails with an INV_REQ_IN_BLOCK_TX error.

When incrementing measurements related to block transactions, the whole block is treated as a single provisioning command. If a block contains four requests (such as <updateSubscriber>), then the subsequent measurements are incremented by one.

Single Database Transaction Mode

Single database transaction mode implicitly begins and ends a transaction for each individual update request.

In single database transaction mode, database manipulation and query requests are sent without being enclosed by <startTransaction/> and <commit/> requests.

When sending Single Database Transaction Mode update or delete requests, each command is implicitly done within a transaction by the SDS, such as when sending <startTransaction/>, <request>, and <commit/> requests. For read requests, no transaction is used by the SDS.

ACID-Compliant Transactions

The SOAP Interfaces support Atomicity, Consistency, Isolation and Durability (ACID)-compliant database transactions which guarantee transactions are processed reliably.

Atomicity

Database manipulation requests are atomic. If one database manipulation request in a transaction fails, all of the pending changes can be rolled back by the client, leaving the database as it was before the transaction was initiated. However, the client also has the option to close the transaction, committing only the changes within that transaction which were executed successfully. If any database errors are encountered while committing the transaction, all updates are rolled back and the database is restored to its previous state.

Consistency

Data across all requests performed inside a transaction is consistent.

Isolation

All database changes made within a transaction by one client are not viewable by any other clients until the changes are committed by closing the transaction. In other words, all database changes made within a transaction cannot be seen by operations outside of the transaction.

Durability

Once a transaction has been committed and become durable, it will persist and not be undone. Durability is achieved by completing the transaction with the persistent database system before acknowledging commitment. Provisioning clients only receive SUCCESS responses for transactions that have been successfully committed and have become durable.

The system will recover committed transaction updates in spite of system software or hardware failures. If a failure (i.e., loss of power) occurs in the middle of a transaction, the database will return to a consistent state when it is restarted.

Data durability signifies the replication of the provisioned data to different parts of the system before a response is provided for a provisioning transaction. The following additive configurable levels of durability are supported:

1. Durability to the disk on the active provisioning server (i.e., just 1)
2. Durability to the local standby server memory (i.e., 1 + 2)
3. Durability to the active server memory at the Disaster Recovery site (i.e., 1 + 2 + 3)

Data Import

SDS provides automatic file-based bulk import of provisioning data. Files from a remote directory can be imported and the values within the files used to populate the database. The files can contain data in CSV or XML format. The type of data that can be imported for each format type is defined in *Table 2: Data Provisioning Interfaces*.

Import options are configured using the SDS GUI. See the *SDS Online Help* for more information.

Imports are not scheduled through the GUI. The imports are initiated by the presence of a file placed in the Remote Import Directory.

Import files that are placed in the specified location on the remote server are detected within five minutes and automatically downloaded using SSH File Transfer Protocol (SFTP) to the file management storage area on the active server. For a file to be imported it must:

- Be named correctly. CSV import files must match the file names shown in *Table 7: CSV Import Formats*. XML import files must have **`*.xml`** file extensions.
- Have been placed in the remote directory after the time when the import last ran.
- Not have been previously imported. A file that has already been imported into the local directory will not be imported again, even if the status is failed. To import a previously failed file, correct the file as necessary, rename the file, and place the renamed file in the remote directory.

Once fully downloaded, each file is automatically imported into the Provisioning Database in the order of their time stamps from the remote server.

The import file is an ASCII text file that contains a series of database manipulation requests. Each request must be formatted on a single line.

An import log file is created for each file that is imported, and a copy is automatically uploaded to the same location the import file was downloaded from on the remote server. The log file has the same name as its corresponding import file with **`.log`** appended. Import log files on the local system are viewable for up to 7 days or until manually removed.

The import log file contains:

- Date and time (in UTC) the import operation started and completed including percentage of the import file (lines) complete
- All requests that resulted in failure along with associated error code (value and string representation), and line of the import file containing the failure.
- Total number of requests successfully committed and failed.

The format of XML or CSV import logs:

```
mm/dd/yy hh:mm:ss Started (0 of linesToImport) 0% complete
```

```
reqMsg
[error errorValue errorString : line lineOfFailure]  [description]

. . .

reqMsg
[error errorValue errorString : line lineOfFailure]  [description]

mm/dd/yy hh:mm:ss <Completed|Interrupted> (linesImported of linesToImport)
percentCplt% complete

Successful: successfulCmds  Failures: failedCmds  Total: totalCmds
```

Table 5: Import Log File Parameters describes the import log file parameters.

Table 5: Import Log File Parameters

Parameter	Description	Values
mm/dd/yy	Date, in UTC, that the entry was logged.	• mm = 01-12 (month) • dd = 01-31 (day of month) • yy = 00-99 (last two digits of the year)
hh:mm:ss	Time, in UTC, the entry was logged.	• hh = 00-23 (hours) • mm = 00-59 (minutes) • ss = 00-59 (seconds)
linesImported	Number of lines of the import file that have been processed	
linesToImport	Total number of lines of the import file to be processed	
percentCplt	Percentage of import file (lines) processed	
reqMsg	Request Message that resulted in error	
errorValue	Message Response Error Value	
errorString	Message Response Error String	
lineOfFailure	Line number of the failed Request Message	
description	Description of any Request Message failure.	
successfulCmds	Total number of Request Messages successfully committed	
failedCmds	Total number of Request Messages that resulted in failure	
totalCmds	Total number of Request Messages that were processed	

Example of a successfully completed import log file:

```
02/06/13 13:28:01 Started (0 of 200) 0% complete

<removeSubscriber ent="subscriberRouting"
ns="dsr"><imsi>310910421000102</imsi></removeSubscribe r>
[error 2001 INV_REQUEST_NAME : line 5]

<updateSubscriber ent="subscriberRouting"
ns="dsr"><imsi>310910421000102</imsi><ltehss>LTE_HSS_9
</ltehss></updateSubscriber>
[error 2006 DEST_NOT_FOUND : line 17]

<deleteSubscriber ent="subscriberRouting" ns="dsr"><imsi>310910421000199</imsi>
 </deleteSubscriber>
[error 2007 IMSI_NOT_FOUND : line 36]

<startTransaction/
[error 1028 BAD_IMPORT_CMD : line 77]

02/06/13 13:28:03 Completed (200 of 200) 100% complete

Successful: successfulCmds  Failures: failedCmds  Total: totalCmds
```

Example of an interrupted import log file:

```
02/06/13 13:28:01 Started (0 of 200) 0% complete

02/06/13 13:28:03 Connection terminated

02/06/13 13:28:03 Interrupted (100 of 200) 50% complete

Successful: 100  Failures: 0  Total: 100
```

The status of all imported files can be viewed from the SDS GUI. See the *SDS Online Help* for more information

Provisioning Data Import (XML)

Data can be imported from an XML import file to add, update, or delete existing data in the provisioning database.

An XML import file is an ASCII text file that contains a series of database manipulation requests in XML format as specified in *XML Message Definitions*. An import file may contain as many requests as the storage media used to hold the import file allows. *Table 6: Supported Database Requests for XML Import Files* shows the database manipulation requests that are supported in an XML import file.

Table 6: Supported Database Requests for XML Import Files

XDS Operation	Description	Section
updateSubscriber	Update Subscriber Routing Data (type IMSI/MSISDN)	*Update Subscriber*
deleteSubscriber	Delete Subscriber Routing Data (type IMSI/MSISDN)	*Delete Subscriber*

XDS Operation	Description	Section
updateSubscriberNAI	Update Subscriber Routing Data (type NAI)	*Update Subscriber NAI*
deleteSubscriberNAI	Delete Subscriber Routing Data (type NAI)	*Delete Subscriber NAI*

Unsupported requests are skipped, and each occurrence is recorded as BAD_IMPORT_CMD in the import log file. Errors encountered while processing the import file are recorded in the import log. Unknown/invalid requests are skipped with each occurrence recorded as INV_REQUEST_NAME in the import log file.

Blank and comment lines are skipped. The format of a XML comment line is:

```
<!-- comment -->
```

XML requests are processed in the order that they are read from the import file and must be ordered to satisfy any data dependencies.

Import file names on the remote server must be suffixed with `.xml` to be automatically downloaded and imported into the provisioning database.

```
<filename>.xml
```

Update requests may be unavailable to other clients for the duration of an import operation, if the import mode is set to `blocking` (see the Export Mode configuration variable in *XML/SOAP Interface System Variables*). Read requests are always available.

Provisioning Data Import (CSV)

A CSV import file consists of an ASCII text file that contains a series of database manipulation requests in CSV format. Each request must be on a separate line.

An import file can contain as many requests as the storage media used to hold the import file allows. The CSV import process ignores all blank lines and lines that begin with a # character, which are treated as comments.

Table 7: CSV Import Formats shows the supported CSV import formats, examples of each update and delete command, and the import file names.

If the import line has fewer values separated by commas than the number of fields listed in the Format column, the missing fields are treated as unspecified and contain no value. The file names must have the format shown in the table, where **X** is at least one alpha-numeric character.

Table 7: CSV Import Formats

Import Type	Format/Example	File Name
Destination	<U|D>,<NAME>,<TYPE>,<FQDN>,<REALM>	import_X_destination.csv
	U,imsgroup1,imshss,operator.com,imswest D,aaagroup2	

Import Type	Format/Example	File Name
IMSI	<U|D>,<IMSI>,<IMSHSSname>,<LTEHSSname>, <PCRFname>,<OCSname>,<OFCSname>,<AAAname>, <UserDef1Name>, <UserDef2Name>	import_X_imsi.csv
	U,7857857802,,dest2,,,,,,dest8 D,7857857803	
MSISDN	<U|D>,<MSISDN>,<IMSHSSname>,<LTEHSSname>, <PCRFname>,<OCSname>,<OFCSname>,<AAAname>, <UserDef1Name>, <UserDef2Name>	import_X_msisdn.csv
	U,17857853013,imsgroup13,dest9,,,,,,dest8 D,17857853014	
NAI User	<U|D>,<USER>,<HOST>,<IMSHSSname>,<LTEHSSname>, <PCRFname>,<OCSname>, <OFCSname>, <AAAname>,<UserDef1Name>, <UserDef2Name>	import_X_naiuser.csv
	U,dptestUser03,dptestHost0,,,dest3 D,dptestUser07,dptestHost0	
Wildcard NAI User	<U|D>,<WCUSER>,<HOST>,<IMSHSSname>, <LTEHSSname>,<PCRFname>,<OCSname>, <OFCSname>,<AAAname>,<UserDef1Name>, <UserDef2Name>	import_X_wcnaiuser.csv
	U,dptestUser2,dptestHost0,,,,,,dest20,dest28 D,dptestUser3,dptestHost0	
NAI Host	<U|D>, <HOST>	import_X_naihost.csv
	U,dptestHost0 D,dptestHost1	
IMSI Prefix	<U|D>,<IMSIprefix>,<IMSHSSname>, <LTEHSSname>,<PCRFname>,<OCSname>, <OFCSname>,<AAAname>, <UserDef1Name>, <UserDef2Name>	import_X_imsiprefix.csv
	U,78578520,imsgroup22 D,78578540	
MSISDN Prefix	<U|D>,<MSISDNprefix>,<IMSHSSname>, <LTEHSSname>,<PCRFname>,<OCSname>, <OFCSname>,<AAAname>,<UserDef1Name>, <UserDef2Name>	import_X_msisdnprefix.csv
	U,17857852012,imsgroup24 D,178578540	

Import Type	Format/Example	File Name
IMSI Blacklist	<U|D>,<IMSI>	import_X_imsiblacklist.csv
	U,7857851011 D,7857852012	
MSISDN Blacklist	<U|D>,<MSISDN>	import_X_msisdnblacklist.csv
	U,17857851011 D,17857852012	
Subscriber	<U|D>,<AccountId>,<IMSI1>,<IMSI2>,<IMSI3>,<IMSI4>,<IMSI5>,<IMSI6>,<MSISDN1>,<MSISDN2>,<MSISDN3>,<MSISDN4>,<MSISDN5>,<MSISDN6>,<IMSHSSname>,<LTEHSSname>,<PCRFname>,<OCSname>,<OFCSname>,<AAAname>,<UserDef1Name>,<UserDef2Name>	import_X_subscriber.csv
	U,777777777777777777711111, 7774605500, 7774605501,,,,, 777771111122222, 777771111133333,,,,,imsgroup1,ltehss2,dest3,dest4,,,dest7,dest8 D,77777777777777777777722222,,,,,,,,,,,	

Table 8: CSV Import Fields defines the Import Type fields.

Table 8: CSV Import Fields

Field	Description
<U|D>	Update or delete record. (U=Update, D=Delete)
<NAME>	A unique string of 1-32 characters to identify the Destination.
<IMSI>	A unique string of 10-15 decimal digits.
<IMSIprefix>	A unique string of 1-15 decimal digits.
<TYPE>	Destination type. Values: imshss, ltehss, pcrf, ocs, ofcs, aaa, userdef1 or userdef2)
<FQDN>	A 1-255 character string for the Diameter FQDN for the destination.
<REALM>	A 1-255 character string for the Diameter Realm for the destination. Optional.
<MSISDN>	A unique string of 8-15 decimal digits.
<MSISDNprefix>	A unique string of 1-15 decimal digits.
<IMSHSSname>	Name of an IMS HSS destination.
<LTEHSSname>	Name of an LTE HSS destination.

Field	Description
<PCRFname>	Name of a PCRF destination.
<OCSname>	Name of an OCS destination.
<OFCSname>	Name of an OFCS destination.
<AAAname>	Name of an AAA destination.
<UserDef1Name>	Name of a UserDef1 destination.
<UserDef2Name>	Name of a UserDef2 destination.
<USER>	A string of 1-64 characters for the NAI User Name.
<WCUSER>	A string of 1-64 characters for the Wildcarded NAI User Name.
<HOST>	A unique string of 1-64 characters for the NAI Host Name.
<AccountId>	A unique string of 1-26 decimal digits, if present.

An import can be specified to run in one of the following modes:

- **Blocking** – An import runs while updates are blocked on all other PDBI connections. This allows for a logically complete import file created in the fastest time possible at the cost of delaying any new provisioning updates until the import is completed and the transaction is closed.
- **Non-Blocking (Real-time)** – An import runs while updates are continued to be received and committed to the database.

Import file names on the remote server must have a suffix of *.csv* to be automatically downloaded and imported into the provisioning database.

CSV Data Import for Subscribers

Each line in the subscriber import file is for one subscriber, which is defined as a group of related routing entities.

The update commands in the subscriber import file contain all of the routing entities and Account ID values for one subscriber. During the import process, all specified routing entities and Account ID values are added to a new subscriber, or an existing subscriber is updated to contain only the specified values.

For example, if an existing subscriber has an IMSI or MSISDN value that was not specified in the CSV import file, that IMSI or MSISDN routing entity is removed from the subscriber and deleted.

The delete commands in the subscriber import file must indicate at least one Account Id, MSISDN, or IMSI value for each subscriber. The delete commands delete the whole subscriber, including all routing entities related to that subscriber.

By default, the MSISDN and IMSI routing entity CSV files are for stand-alone routing entities. The import command adds a new stand-alone routing entity, updates an existing routing entity, or deletes a routing entity.

If an update command is for an MSISDN or IMSI value that is part of a subscriber, then the updated destination values are automatically applied to all other routing entities for the subscriber.

If a delete command is for an MSISDN or IMSI value that is part of a subscriber, the delete affects only a single routing entity. The delete command cannot delete the last routing entity for a subscriber. The user must delete the whole subscriber.

Data Export

The export feature allows a full text export of the database. Exported records can be used to perform data manipulation of subscriber data. Exports can be scheduled as one-time or recurring. Exported data can be offloaded to a remote server. The exported text file can be downloaded from the file transfer area.

Note: Export is a time consuming operation recommended to be scheduled during off-peak hours.

Export options and scheduling are configured using the GUI on the Active SDS Server on the Primary Provisioning Site. The GUI is also used to view the status of all in-progress or completed exports. See the *SDS Online Help* for more information.

The type of data that can be exported is defined in *Table 2: Data Provisioning Interfaces*. All export formats allow all available data to be exported for the given format.

The XML and CSV exports use the same format as the imports. See *Provisioning Data Import (XML)* and *Provisioning Data Import (CSV)* for more information.

The HLRR Export creates `ent_sub` HLRR PDBI commands. If data is exported in the HLRR format, only the MSISDN and/or IMSI values that have E.164 addresses are exported. The HLRR format produces commands that are in HLR Router's PDBI format.

If the *All* option is selected for export in the XML or CSV format, then each MSISDN and IMSI value is exported once. If the MSISDN value is assigned to a subscriber, then the MSISDN value is exported with the subscriber data. If the MSISDN value is not assigned to a subscriber, then the MSISDN value is exported with the MSISDN data. The IMSI value is also exported with subscriber or IMSI data.

The export file is an ASCII text file with 1 line per entry. The first line of the export file contains a comment that indicates the export mode, the data base level when export was started, and the time the export was started. Before each type of data is exported, a comment line indicates the type of data that follows. The last line of the export file contains a comment that indicates when the export finished. If the export was run in non-blocking mode, then the database level at the end of the export is listed before the time value within the comment.

Export file formats vary, depending on the export format type as shown in the following sections.

Export XML file format in non-blocking mode

```
<!-- mode, level, yyyymmddhhmmss -->
<!-- type -->
reqMsg

. . .

reqMsg

. . .

<!-- type -->
```

```
reqMsg

. . .

reqMsg
<!-- level, yyyymmddhhmmss -->
```

Export CSV file format in non-blocking mode

```
# mode, db level=level, start time=yyyymmddhhmmss
# type
reqMsg

. . .

reqMsg

. . .

# type
reqMsg

. . .

reqMsg
# db level=level, finish time=yyyymmddhhmmss
```

Export HLRR file format in non-blocking mode

```
# mode, db level=level, start time=yyyymmddhhmmss
reqMsg

. . .

reqMsg
# db level=level, finish time=yyyymmddhhmmss
```

Export Log File parameters

Table 9: Export Log File Parameters

Parameter	Description	Values
mode	Export Mode	• blocking - Updates are blocked during export. • realtime - Updates are allowed during export.
level (Optional)	Durable database level (at start or end of export). If exporting in blocking mode, the level is not displayed on the last line of file.	0-4294967295

Parameter	Description	Values
yyyymmddhhmmss	Date and time (in UTC) export started or completed .	• yyyy - 1970-2099 (year) • mm - 01-12 (month) • dd - 01-31 (day of month) • hh - 00-23 (hours) • mm - 00-59 (minutes) • ss - 00-59 (seconds)
type (Optional)	Comment with type of data being exported. Export in HLRR format does not have this field because HLRR format only exports one type of command.	
regMsg	Exported data in an update request message format.	

Relaying data to the HLR Router

SDS provides two ways to send data to the HLR router:

- PDB Relay
- Bulk Load Between SDS and HLR Router

These methods allow the MSISDN and IMSI routing entities to be provisioned once on the SDS server instead of provisioning the data on both SDS and the HLR router.

PDB Relay

The MSISDN and IMSI routing entities can be provisioned with destinations that have an E.164 network entity value. These provisioning commands are automatically sent from the Active SDS server on the Primary Provisioning Site to HLRR.

The PDBA client called `pdbrelay` connects to a remote PDBA running on the HLR Router system and relays the desired provisioning received from the customer provisioning system. Only commands that could affect HLR Router subscribers are relayed.

Bulk Load

Data can be transferred between the SDS and the HLR Router by exporting SDS data and then importing the file on the Active Network OAM&P HLR Router server.

Note: The data transfer is performed from the SDS GUI. See the *SDS Online Help* for more information.

To transfer the data:

1. Disable PDB Relay Enabled and set the Export Mode configuration option value to Blocking on the SDS GUI.
2. Schedule an export on the SDS GUI.
3. After the export, check the Relay Exception Log for any new pdbexport exceptions on the SDS GUI.
4. Transfer the export file to the HLR Router.
5. Store the file on the Remote Import server and directory displayed on the HLR Router GUI. Refer to the *Online Help* for the current version of the HLR Router for more information.
6. Rename the file on the HLR Router server by changing the `.hlrr` extension to `.pdbi`.
7. The HLR Router automatically imports the file. Verify successful import.
8. Enable PDB Relay Enabled and set the Export Mode configuration option to Blocking or Non-Blocking on the SDS GUI.

Measurements

XML Data Server and SOAP Server specific measurements are collected and made available to the user via the SDS GUI. See the *SDS Online Help* for more information. The XML Data Server, SOAP Server, and bulk import/export tools all update the same measurements.

Important: The format of this information will conform to SDM practices, so may vary from the format described here.

Table 10: SDS Measurements

ID	Group	Tag	Coll Interval	Description
4100	PROV	ProvConnectsAttempted	5 min	The total number of client-initiated connect attempts to establish a connection with the server.
4101	PROV	ProvConnectsAccepted	5 min	The total number of client-initiated connect attempts that have been accepted.
4102	PROV	ProvConnectsDenied	5 min	The total number of client initiated connect attempts that have been denied due to clients not running on an authorized server, maximum number of allowed connections already established, or the provisioning interface is disabled.
4103	PROV	ProvConnectsFailed	5 min	The total number of client initiated connect attempts that failed due to errors during initialization.
4105	PROV	ProvConnectionIdleTimeouts	5 min	Total number of connections that have timed out and terminated due to idleness. Timeout period is specified by *XML Interface Idle Timeout* as described in *XML/SOAP Interface System Variables*.

ID	Group	Tag	Coll Interval	Description
4110	PROV	ProvMsgsReceived	5 min	The total number of provisioning messages that have been received.
4111	PROV	ProvMsgsSuccessful	5 min	The total number of provisioning messages that have been successfully processed.
4112	PROV	ProvMsgsFailed	5 min	The total number of provisioning messages that have failed to be processed due to errors. See *SDS Response Message Error Codes* for a list and description of possible errors.
4113	PROV	ProvMsgsSent	5 min	The total number of provisioning messages that have been sent.
4114	PROV	ProvMsgsDiscarded	5 min	The total number of provisioning messages that have been discarded due to the connection being shutdown, server being shutdown, server's role switching from active to standby, or transaction not becoming durable within the allowed amount of time.
4120	PROV	ProvMsgsImported	5 min	The total number of provisioning messages that have been received from an import operation.
4140	PROV	ProvTxnCommitted	5 min	The total number of transactions that have been successfully committed to the database (memory and on disk) on the active server of the primary SDS site.
4141	PROV	ProvTxnWriteMutexTimeouts	5 min	The total number of transactions that have failed to be processed due to timing out while waiting to acquire the transaction mutex.
4142	PROV	ProvTxnFailed	5 min	The total number of transactions that have failed to be started, committed, or aborted due to errors. See *SDS Response Message Error Codes* for a list and description of possible errors.
4143	PROV	ProvTxnAborted	5 min	The total number of transactions that have been successfully aborted.
4144	PROV	ProvTxnTotal	5 min	The total number of transactions that have been attempted. It is the sum of ProvTxnCommitted, ProvTxnTimeouts, ProvTxnAborted, and ProvTxnFailed counters.

ID	Group	Tag	Coll Interval	Description
4145	PROV	ProvTxnDurabilityTimeouts	5 min	The total number of committed, non-durable transaction that have failed to become durable within the amount of time specified by T*ransaction Durability Timeout*, as described in *XML/SOAP Interface System Variables*.
4155	PROV	RemoteAuditStarted	5 min	The number of started remote audit requests.
4156	PROV	RemoteAuditCompleted	5 min	The number of successfully completed remote audit requests.
4157	PROV	ProvRelayMsgsSent	5 min	The total number of relayed PROVISIONING messages sent to the remote system.
4158	PROV	ProvRelayMsgsSuccessful	5 min	The total number of relayed PROVISIONING messages that have been successfully processed on the remote system.
4159	PROV	ProvRelayMsgsFailed	5 min	The total number of relayed PROVISIONING messages that have failed to be processed due to errors on the remote system.
4160	PROV	ProvImportsSuccessful	5 min	The total number of files imported successfully.
4161	PROV	ProvImportsFailed	5 min	The total number of files that had failed to be imported due to errors.
4162	PROV	ProvExportsSuccessful	5 min	The total number of successful CSV/XML export requests.
4163	PROV	ProvExportsFailed	5 min	The total number of CSV/XML export requests that have failed due to errors.
4174	PROV	ProvDnSplitCreated	5 min	The number of MSISDN records successfully created by an Active Split.
4175	PROV	ProvDnSplitRemoved	5 min	The number of MSISDN records successfully removed by a Completing Split.
4176	PROV	ProvNpaSplitStarted	5 min	The number of NPA split records successfully starting a PDP.
4177	PROV	ProvNpaSplitCompleted	5 min	The number of NPA split records successfully completing a PDP.
4179	PROV	ProvRemoteAuditMsgsSent	5 min	The number of IMSI and MSISDN records audited.

ID	Group	Tag	Coll Interval	Description
4189	PROV	ProvRelayTimeLag	5 min	The time in seconds between timestamps of last record PdbRelay processed and latest entry in the Command Log.

Key Performance Indicators

Table 11: Provisioning Interface KPI Measurements shows the provisioning-specific Key Performance Indicators (KPIs) that are available to the user on the SDS GUI. *Table 12: Process-based KPIs* shows the process-based KPIs.

For all Provisioning Interface KPIs, the Scope has a value of 'A'.

Important: The format of this information will conform to SDM practices, so may vary from the format described here.

Table 11: Provisioning Interface KPI Measurements

ID	Name	Avg. Interval	Description
4104	ProvConnections	60 sec	The number of provisioning client connections currently established. A single connection includes a client having successfully established a TCP/IP connection, sent a provisioning connect message, and having received a successful response.
4110	ProvMsgsReceived	60 sec	The number of provisioning messages that have been received per second.
4111	ProvMsgsSuccessful	60 sec	The number of provisioning messages that have been successfully processed per second.
4112	ProvMsgsFailed	60 sec	The number of provisioning messages per second that have failed to be processed due to errors. See *SDS Response Message Error Codes* for a list and description of possible errors.
4113	ProvMsgsSent	60 sec	The number of provisioning messages sent per second.
4114	ProvMsgsDiscarded	60 sec	The number of provisioning messages discarded per second. Provisioning messages are discarded due to the connection being shutdown, server being shutdown, server's role switching from active to standby, or transaction not becoming durable within the allowed amount of time.
4120	ProvMsgsImported	60 sec	The number of provisioning messages imported per second.

ID	Name	Avg. Interval	Description
4140	ProvTxnCommitted	60 sec	The number of provisioning transactions per second that have been successfully committed to the database (memory and on disk) on the active server of the primary SDS cluster.
4142	ProvTxnFailed	60 sec	The number of provisioning transactions per second that have failed to be started, committed, or aborted due to errors. See *SDS Response Message Error Codes* for a list and description of possible errors.
4143	ProvTxnAborted	60 sec	The number of provisioning transactions aborted per second.
4150	ProvTxnActive	60 sec	The number of provisioning transactions that are currently active (normal transaction mode only).
4151	ProvTxnNonDurable	60 sec	The number of transactions that have been committed, but are not yet durable. Responses for the associated requests are not sent until the transaction has become durable.
4157	ProvRelayMsgsSent	60 sec	The number of relayed provisioning messages sent per second.
4158	ProvRelayMsgs Successful	60 sec	The number of relayed provisioning messages that have been successfully processed per second.
4159	ProvRelayMsgs Failed	60 sec	The number of relayed provisioning messages per second that have failed to be processed due to errors.
4179	ProvRemoteAudit MsgsSent	60 sec	The number of IMSI and MSISDN records audited per second.
4189	ProvRelayTimeLag	60 sec	Time in seconds between timestamps of last record PdbRelay processed and latest entry in the Command Log.

For all process-based KPIs, the Scope has a value of 'A'.

Table 12: Process-based KPIs

ID	Name	Avg. Interval	Description
4165	provimport.Cpu	60 sec	CPU usage of provimport process
4166	provimport.MemHeap	60 sec	Heap memory usage of provimport process
4167	provimport.MemBasTotal	60 sec	Memory usage of the provimport process
4168	provimport.MemPerTotal	60 sec	Percent memory usage of provimport process
4170	provexport.Cpu	60 sec	CPU usage of provexport process
4171	provexport.MemHeap	60 sec	Heap memory usage of provexport process
4172	provexport.MemBasTotal	60 sec	Memory usage of the provexport process

ID	Name	Avg. Interval	Description
4173	provexport.MemPerTotal	60 sec	Percent memory usage of provexport process
4180	pdbrelay.Cpu	60 sec	CPU usage of pdbrelay process
4181	pdbrelay.MemHeap	60 sec	Heap memory usage of pdbrelay process
4182	pdbrelay.MemBasTotal	60 sec	Memory usage of the pdbrelay process
4183	pdbrelay.MemPerTotal	60 sec	Percent memory usage of pdbrelay process
4184	pdbaudit.Cpu	60 sec	CPU usage of pdbaudit process
4185	pdbaudit.MemHeap	60 sec	Heap memory usage of pdbaudit process
4186	pdbaudit.MemBasTotal	60 sec	Memory usage of the pdbaudit process
4187	pdbaudit.MemPerTotal	60 sec	Percent memory usage of pdbaudit process
4190	pdba.Cpu	60 sec	CPU usage of pdba process
4191	pdba.MemHeap	60 sec	Heap memory usage of pdba process
4192	pdba.MemBasTotal	60 sec	Memory usage of the pdba process
4193	pdba.MemPerTotal	60 sec	Percent memory usage of pdba process
4194	xds.Cpu	60 sec	CPU usage of xds process
4195	xds.MemHeap	60 sec	Heap memory usage of xds process
4196	xds.MemBasTotal	60 sec	Memory usage of the xds process
4197	xds.MemPerTotal	60 sec	Percent memory usage of xds process
4200	dpserver.Cpu	60 sec	CPU usage of dpserver process on DP
4201	dpserver.MemHeap	60 sec	Heap memory usage of dp server process on DP
4202	dpserver.MemBasTotal	60 sec	Memory usage of the dpserver process on DP
4203	dpserver.MemPerTotal	60 sec	Percent memory usage of dpserver process on DP
4310	era.Cpu	60 sec	CPU usage of era process
4311	era.MemHeap	60 sec	Heap memory usage of era process
4312	era.MemBasTotal	60 sec	Memory usage of the era process
4313	era.MemPerTotal	60 sec	Percent memory usage of era process

Alarms

XML Data Server and SOAP Server specific alarms are available to the user via the SDS GUI and Network Operation Center (NOC) console(s) if SNMP is configured by the SDS GUI. See the *SDS Online Help* for more information.

Table 13: Alarms

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
14100	PROV	Interface Disabled	PROV Interface manually disabled	Critical	N/A	Normal	5	0	Provisioning interface is manually disabled.
			PROV Interface manually enabled	Clear					Provisioning interface is manually enabled.
14101	PROV	No Remote Connections	No remote provisioning clients are connected	Major	N/A	Normal	5	0	Provisioning interface is enabled and no remote provisioning clients are connected.
			One or more remote provisioning clients are connected	Clear					Provisioning interface is enabled and one or more remote provisioning clients are connected.
14102	PROV	Connection Failed	Initialization Failed (CID Connection ID, IP IP Address)	Major	Connection ID: IP Address	Normal	5	300	Provisioning connection establishment failed due to an error specified in addl info.
			Initialization Successful (CID Connection	Clear					Alarm automatically cleared after 5 minutes.

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
			ID, IP IP Address)						
14103	PROV	Both Port Identical	Provisioning ports are the same	Major	N/A	Normal	5	0	XML & SOAP Provisioning interfaces are disabled since same port is configured for both interfaces.
			One provisioning port is changed to a different value	Clear					XML & SOAP Provisioning interfaces are enabled.
14140	PROV	Import Throttled	Import operation throttled (CID Connection ID)	Major	provimport	Normal	5	5	Provisioning import throttled to prevent overrunning idb service processes.
			Import operation throttled (CID Connection ID) cleared	Clear					Alarm automatically cleared in 5 seconds after throttling subsides.
14150	PROV	Import Initialization Failed	Initialization error, see trace log for details	Major	provimport	Normal	5	0	Provisioning import initialization failed due to an error specified in addl info.
			Initialization error cleared	Clear					Provisioning import initialization completed successfully.

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
14151	PROV	Import Generation Failed	Failed to import file, see trace log for details	Major	provimport	Normal	5	0	Provisioning import operation failed due to an error specified in addl info.
			Generation error cleared	Clear					Provisioning import operation completed successfully.
14152	PROV	Import Transfer Failed	Failed to transfer file from remote host, see trace log for details	Major	provimport	Normal	5	0	Provisioning import operation failed due to a file transfer error specified in addl info.
			Transfer error cleared	Clear					Provisioning import operation completed successfully.
14153	PROV	Export Initialization Failed	Initialization error, see trace log for details	Major	provimport	Normal	5	0	Provisioning export initialization failed due to an error specified in addl info.
			Initialization error cleared	Clear					Provisioning export initialization completed successfully.
14154	PROV	Export Generation Failed	Scheduled export failed, see trace log for details	Major	provimport	Normal	5	0	Provisioning export operation failed due to an error

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
									specified in addl info.
			Generation error cleared	Clear					Provisioning export operation completed successfully.
14155	PROV	Export Transfer Failed	Failed to transfer file to remote host, see trace log for details	Major	provimport	Normal	5	0	Provisioning export operation failed due to a file transfer error specified in addl info.
			Transfer error cleared	Clear					Provisioning export operation completed successfully.
14188	PROV	Pdbrelay not connected	Bulkload of remote system required: Cannot find last Prov Relay Timestamp in Cmd Log or No relay remote system IP address defined in configuration options or Timeout while	Major	pdbrelay	Normal	0	0	Pdbrelay feature is enabled, but the connection to the remote HLRR system is not established. To remedy, 1) perform Bulk Load Procedure at the HLRR, 2) configure the HLRR address in the SDS GUI, or 3) verify network connectivity

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
			connecting to server						with the HLRR.
			Remote HLRR is connected	Clear					Pdbrelay feature is enabled, and the connection to the remote HLRR system is established.
14189	PROV	PdbRelay Time Lag	ProvRelay TimeLag above critical threshold	Critical	pdbrelay	Normal	300	0	Pdbrelay feature is enabled and the time between timestamps of the last record processed and the latest entry in the Command Log exceeds 28.5 minutes.
			ProvRelay TimeLag above major threshold	Major					Pdbrelay feature is enabled and the time between timestamps of the last record processed and the latest entry in the Command Log exceeds 15 minutes.
			ProvRelay TimeLag	Minor					Pdbrelay feature is

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
			above minor threshold						enabled and the time between timestamps of the last record processed and the latest entry in the Command Log exceeds 4.5 minutes.
			ProvRelay TimeLag within acceptable time	Clear					Pdbrelay feature is not enabled or the time between timestamps of the last record processed and the latest entry in the Command Log is less than 3 minutes
14301	ERA	ERA_Responder Failed	Event responder failed	Major	N/A	Normal	300	0	Internal error occurred - contact Tekelec.
			Event responder error cleared	Clear					Internal error cleared

Events

XML Data Server and SOAP Server specific events are available to the user via the SDS GUI and Network Operation Center (NOC) console(s) if SNMP is configured by the SDS GUI. See the *SDS Online Help* for more information.

Important: The format of this information will conform to SDM practices, so may vary from the format described here.

The following parameters apply to all events listed in *Table 14: Events*:

- Severity - Info
- Instance - N/A
- HA Score - Normal
- Auto Clear Seconds - 0
- Assert/Clear Conditions - N/A

Table 14: Events

ID	Name/Descr Text	Addl Info	Throttle Secs
14120	Connection Established	Provisioning client connection established.	5
14121	Connection Terminated	Provisioning client connection terminated.	5
14122	Connection Denied	Provisioning client connection denied.	5
14160	Import Operation Completed	See XML Import screen for details.	5
14161	Export Operation Completed	See XML Export screen for details.	5
14170	Remote Audit started and in progress	Remote Audit started and is in progress.	30
14171	Remote Audit aborted	Remote Audit aborted.	30
14172	Remote Audit failed to complete	Remote Audit failed to complete.	30
14173	Remote Audit completed	Remote Audit completed.	30
14174	NPA Split pending request deleted	NPA Split pending request deleted.	0
14175	NPA Split activation failed	NPA Split activation failed.	0

ID	Name/Descr Text	Addl Info	Throttle Secs
14176	NPA Split started and is Active	NPA Split started and is Active.	0
14177	NPA Split completion failed	NPA Split completion failed.	0
14178	NPA Split completed	NPA Split completed.	0
14179	MSISDN deleted from Blacklist	Previously Blacklisted MSISDN is now a Routing Entity.	0
14180	IMSI deleted from Blacklist	Previously Blacklisted IMSI is now a Routing Entity.	0

Chapter 4

SOAP Message Definitions

Topics:

This chapter describes the SOAP message syntax and parameters.

Message Conventions

Message specification syntax follows several conventions to convey what parameters are required or optional and how they and their values must be specified.

Table 15: Message Conventions

Symbol	Description
`monospace with background`	All code examples.
`monospace`	Names of commands when provided outside of a code example.
italics	Variable names when provided outside of a code example or value list.
spaces	Spaces (ie, zero or more space characters, " ") may be inserted anywhere except within a single name or number. At least one space is required to separate adjacent names or numbers.
…	Ellipses represent a variable number of repeated entries. For example: `dn DN1 , dn DN2, …, dn DN7, dn DN8`

< >	Angle brackets are used to enclose parameter values that are choices or names.
	In the following example, the numbers represent specific value choices.
	`parameter1 <1|2|3>`
	In the following example, `ServerName` represents the actual value.
	`parameter2 <ServerName>`
	In the following example, the numbers represent a choice in the range from 0 to 3600.
	`parameter3 <0..3600>`

[ ]	Square brackets are used to enclose an optional parameter and its value. `[, parameter1 < 1|2|3 >]`

Symbol	Description
	A parameter and its value that are not enclosed in square brackets are mandatory.
|	The pipe symbol is used in a parameter value list to indicate a choice between available values. `Parameter1 <1|2|3>`
,	A literal comma is used in the message to separate each parameter that is specified.

SOAP Request Messages

A SOAP request message is sent to the SDS SOAP provisioning client as a series of ASCII characters. The SDS SOAP provisioning client sends back a SOAP response message.

Every SOAP message sent to SDS must be sent in a SOAP envelope. Each SOAP envelope has a `<soapenv:Body>` XML tag. The SDS provisioning or query request is embedded within the `<soapenv:Body>` tag. The tags and values within the `<soapenv:Body>` tag vary for each SDS request.

SOAP Request Message Format

This example shows the format for all SOAP requests. The bolded text varies for each provisioning request.

```
POST / HTTP/1.1
Host: ipAddress:port
Accept-Encoding: identity
Content-Length: lengthInBytes
SOAPAction: ""
Content-Type: text/xml; charset=”UTF-8”

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
        <requestName>
        [
            <requestParameters>
                ...
            </requestParameters>
            ...
            <requestParameters>
                ...
            </requestParameters>
        ]
        </requestName>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Message Request Parameters

Parameter	Description	Value
ipAddress	IP address of the SDS Provisioning Server that received the SOAP request.	
port	Port of the SDS Provisioning Server that received the SOAP request.	
lengthInBytes	Number of bytes in the SOAP request.	0-4294967295
requestName	The name of the SDS provisioning request.	A string with 1 to 64 characters.
requestParameters	The parameters (tag/value pairs) needed for each request. These parameters vary for each SDS Provisioning or query request.	

SOAP Response Messages

A SOAP response message is sent by the SOAP Server in response to a SOAP request. Each response contains a series of ASCII characters.

A rowset, contained between the `<rset>` tags, is present if data is to be returned (i.e. for `<readSubscriberRequest>` and `<readSubscriberNaiRequest>`).

A generic response type can be generated if an SOAP request cannot be parsed, the request is not valid, etc. The `responsename` for this generic response is `errorResponse`.

```
<res error="error" affected="affected" [description="description"]/>
```

Response Format (`<readSubscriberResponse>` and `<readSubscriberNaiResponse>` requests)

The bolded text differs for each response message.

```
HTTP/1.1 200 OK
Server: gSOAP/2.7
Content-Type: text/xml; charset=utf-8; action=""
Content-Length: lengthInBytes
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:ns2="http://www.tekelec.com/sds/"
xmlns:ns4="http://www.tekelec.com/sds/dsr/"
xmlns:ns3="http://www.tekelec.com/sds/dsr/soap/"
xmlns:ns5="http://www.tekelec.com/sds/soap"
    <SOAP-ENV:Header></SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <respName>
            <result affected="affected" error="error"
                [description="description"]>
            </result>
        [
            <resultSet>
                <rowName> [ [rowAttributeName]="rowAttributeValue"] …
                            [rowAttributeName]="rowAttributeValue"] ]>
                    <rowValueName>rowValue</rowValueName>
                    ...
                    <rowValueName>rowValue</rowValueName>
                </rowName>
                ...
                <rowName> [ [rowAttributeName]="rowAttributeValue"] …
                            [rowAttributeName]="rowAttributeValue"] ]>
                    <rowValueName>rowValue</rowValueName>
                    ...
                    <rowValueName>rowValue</rowValueName>
                </rowName>
            </resultSet>
        ]
        </respName>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response Format (All Other Requests)

The bolded text varies for each response message.

```
HTTP/1.1 200 OK
Server: gSOAP/2.7
Content-Type: text/xml; charset=utf-8; action=""
Content-Length: lengthInBytes
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://www.tekelec.com/sds/"
xmlns:ns4="http://www.tekelec.com/sds/dsr/"
xmlns:ns3="http://www.tekelec.com/sds/dsr/soap/"
xmlns:ns5="http://www.tekelec.com/sds/soap"
    <SOAP-ENV:Header></SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <ns2:sdsResult affected="affected" error="error"
            [description="description"]>
        </ns2:sdsResult>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Response Message Parameters

Table 16: Response Message Parameters (SOAP)

Parameter	Description	Value
lengthInBytes	Number of bytes in the SOAP request.	0-4294967295
error	Error code that indicates whether or not operation was successfully executed.	0 for success, non-zero for failure.
affected	The number of routing entities or subscribers (when group=”y”) created/updated/deleted/read. This number does not contain number of subscriber records created/update/deleted because “subscriber” data is not used for routing. It is possible to have affected=0 and error=0.	0-10
description (Optional)	A textual description associated with the response. This field may contain more information as to why a request failed or describe the changes if a request succeeds.	A string with 1 to 1024 characters.
respName (Optional)	The name of the response.	This field is only used for read responses and errors.
rowName	The name of the row type returned.	The value is dependent on the result set returned.
rowValue	The value of the row type returned.	The value is dependent on the result set returned.
rowAttributeName	The name of the row attribute name returned.	The value is dependent on the result set returned.
rowAttributeValue	The value of the row attribute name returned.	The value is dependent on result set returned.

Successful SOAP Subscriber Commands

If the SOAP command successfully updates or deletes a subscriber, then the response description text will indicate the deleted/created/changed IMSI and/or MSISDN values and optionally a list of the subscriber's destination values.

Note: Destination values are listed only if there were created or modified IMSI and/or MSISDN routing entities.

Description Text Format

```
[description="[deleted ({imsi nnnn|dn nnnn}[, imsi nnnn|,dn nnnn]…)]]
[, created ({imsi nnnn|dn nnnn}[, imsi nnnn|,dn nnnn]…)]]
[, changed ({imsi nnnn|dn nnnn}[, imsi nnnn|,dn nnnn]…)]]
[, imshss nnnn][, ltehss nnnn][, pcrf nnnn][, ocs nnnn][, ofcs nnnn]
[, aaa nnnn][, userdef1 nnnn][, userdef2 nnnn]"]
```

Example Description Text from an updateSubscriberRequest Command

```
description="deleted (imsi 444444444444440, dn 19195550000), created
(imsi 444444444444441, dn 19195550001, dn 19195550002), imshss imshss2,
ltehss ltehss1"
```

List of Request Operations

Table 17: Supported SOAP Requests lists the supported SOAP requests.

Table 17: Supported SOAP Requests

Operation Name	Description	
startTransactionRequest	Start Database Transaction	*Start Transaction*
commitRequest	Commit Database Transaction	*Commit Transaction*
rollbackRequest	Abort Database Transaction	*Rollback Transaction*
updateSubscriberRequest	Create/Update IMSI/MSISDN Routing	*Update Subscriber*
deleteSubscriberRequest	Delete IMSI/MSISDN Routing	*Delete Subscriber*
readSubscriberRequest	Get IMSI/MSISDN Routing	*Read Subscriber*
updateSubscriberNaiRequest	Create/Update NAI Routing	*Update Subscriber NAI*
deleteSubscriberNaiRequest	Delete NAI Routing	*Delete Subscriber NAI*
readSubscriberNaiRequest	Get NAI Routing	*Read Subscriber NAI*

Start Transaction

Request

The `<startTransactionRequest>` message is sent to begin a database transaction. Database manipulation and query requests (update, delete, and read) can be sent within the context of the transaction.

If a <startTransactionRequest> is sent, and the connection is lost or the user logs off without sending a <commitRequest> or <rollbackRequest>>, all pending requests are rolled back.

A provisioning session can have one transaction open at a time. If a <startTransactionRequest> is sent, another <startTransactionRequest> will fail with an ACTIVE_TXN error.

A timeout can occur between the <startTransactionRequest> and the <commitRequest>. If the <commitRequest> is not sent out within the configured Maximum Transaction Lifetime (see the *SDS Online Help* for more information) after the <startTransactionRequest>, the SOAP provisioning requests are rolled back (changes not applied to database).

A transaction can only be opened by one client at a time. If a transaction is already opened by another client, the <startTransactionRequest> is rejected immediately with WRITE_UNAVAIL or is queued up for the time specified by the timeout parameter. If the timeout parameter is specified with a non-zero value and that period of time elapses before the transaction is opened, the <startTransactionRequest> is rejected with WRITE_UNAVAIL.

Data manipulation requests are evaluated for validity and applied to a local database view which is a virtual representation of the main database plus local modifications made within this active transaction.

Local database view changes are not committed to the main database until the transaction is ended with a <commitRequest>.

The request can be aborted and rolled back with a <rollbackRequest> request any time before the transaction is ended with a <commitRequest>.

Request Format

The request must be inserted between the <soapenv:Body> and </soapenv:Body> XML tags of a SOAP request message, as shown in *SOAP Request Messages*.

```
<startTransactionRequest>timeout</startTransactionRequest>
```

Request Parameters

Table 18: <startTransactionRequest> Parameters (SOAP)

Parameter	Description	Value
timeout	The amount of time (in seconds) to wait to open a transaction if another connection already has one open. Clients waiting to open a transaction will be processed in the order that their requests were received.	0 (return immediately if not available) to 3600 seconds. The default is 0.

Response

The start transaction response is returned as a generic <ns2:sdsResult> response. This response returns the result of starting a database transaction. If the response error code indicates success, then

the database transaction was successfully started. If any failure response is returned, then the database transaction was not started.

Response Format

The response is displayed between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP response message, as shown in *SOAP Response Messages*.

```
<ns2:sdsResult affected="affected" error="error" [description="description"]>
</ns2:sdsResult>
```

Response Parameters

The parameters for all of the response commands are shown in *SOAP Response Messages*.

Start Transaction Response Error Codes

Table 19: <startTransactionResponse> Error Codes (SOAP) shows common error codes for the `<startTransactionResponse>` message. See *SDS Response Message Error Codes* for a full list of error codes.

Table 19: `<startTransactionResponse>` Error Codes (SOAP)

Error Code	Description
SUCCESS	Transaction was successfully started.
NO_WRITE_PERMISSION	The client making the connection does not have write access permissions.
WRITE_UNAVAILABLE	Another client already has a transaction open. This is only returned to clients who have write access permissions.
ACTIVE_TXN	A read or write transaction is already open on this connection or an open transaction was aborted prior to terminating the connection.

Examples

These examples show the full SDS provisioning request and response contents.

Start a Transaction Within 2 Minutes (success)

This example successfully starts a transaction within 2 minutes.

Request:

```
POST / HTTP/1.1
Host: localhost:9090
Accept-Encoding: identity
Content-Length: 211
SOAPAction: ""
Content-type: text/xml; charset="UTF-8"
```

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <startTransactionRequest>120</startTransactionRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Result:

```
POST / HTTP/1.1 200 OK
Server: gSOAP/2.7
Content-Type: text/xml; charset=utf-8; action=""
Content-Length: 592
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://www.tekelec.com/sds/"
xmlns:ns4="http://www.tekelec.com/sds/dsr/"
xmlns:ns3="http://www.tekelec.com/sds/dsr/soap/"
xmlns:ns5="http://www.tekelec.com/sds/soap">
   <SOAP-ENV:Header></SOAP-ENV:Header>
   <SOAP-ENV:Body>
       <ns2:sdsResult affected="0" error="0">
       </ns2:sdsResult>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Start a Transaction Immediately (fail)

This example attempts to immediately start a transaction but fails due to another client having a transaction open.

Request:

```
POST / HTTP/1.1
Host: localhost:9090
Accept-Encoding: identity
Content-Length: 209
SOAPAction: ""
Content-type: text/xml; charset="UTF-8"

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <startTransactionRequest>0</startTransactionRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Response:

```
POST / HTTP/1.1 200 OK
Server: gSOAP/2.7
Content-Type: text/xml; charset=utf-8; action=""
Content-Length: 595
```

```
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://www.tekelec.com/sds/"
xmlns:ns4="http://www.tekelec.com/sds/dsr/"
xmlns:ns3="http://www.tekelec.com/sds/dsr/soap/"
xmlns:ns5="http://www.tekelec.com/sds/soap">
   <SOAP-ENV:Header></SOAP-ENV:Header>
   <SOAP-ENV:Body>
       <ns2:sdsResult affected="0" error="1005">
       </ns2:sdsResult>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Commit Transaction

Request

The `<commitRequest>` message is sent to commit a database transaction.

If the open transaction has one or more successful updates, then committing the transaction causes all the database changes to be committed.

Important: All previous updates, even if they received a successful error code, are not committed to the database until the `<commitRequest>` is received.

Request Format

The request must be inserted between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP request message, as shown in *SOAP Request Messages*.

```
<commitRequest> </commitRequest>
```

Response

The commit response is returned as a generic `<ns2:sdsResult>` response. This response returns the result of the request to commit a database transaction.

If the response error code indicates success, then the database transaction was successfully committed in the database. If any failure response is returned, then the database commit failed. The commit operation causes the transaction to end regardless of whether any updates were actually made to the database.

Note: The affected row count in the SOAP response is always 0. It does not indicate how many rows were modified within the transaction.

Response Format

The response is displayed between the <soapenv:Body> and </soapenv:Body> XML tags of a SOAP response message, as shown in *SOAP Response Messages*.

```
<ns2:sdsResult affected="affected" error="error" [description="description"]>
</ns2:sdsResult>
```

Parameters

The parameters for all of the response commands are shown in *SOAP Response Messages*.

Error Codes

Table 20: <commitResponse> Error Codes (SOAP) shows common error codes for the <commitResponse> message. See *SDS Response Message Error Codes* for a full list of error codes.

Table 20: <commitResponse> Error Codes (SOAP)

Error Code	Description
SUCCESS	Transaction was successfully committed.
NO_ACTIVE_TXN	A read or write transaction is not currently open for this connection.

Examples

These examples show the SDS provisioning request and response contents that are stored within the <soapenv:Body> or <SOAP-ENV:Body> tags. See Start Transaction *Examples* for an example that contains the entire SOAP request/response text.

Commit a Transaction (success)

This example successfully commits a transaction.

Request:

```
<commitRequest> </commitRequest>
```

Response:

```
<ns2:sdsResult affected="15" error="0">
</ns2:sdsResult>
```

Commit a Transaction that is not Open (fail)

This example attempts to commit a transaction but fails because a transaction was not open.

Request:

```
<commitRequest> </commitRequest>
```

Response:

```
<ns2:sdsResult affected="0" error="1009">
</ns2:sdsResult>
```

Rollback Transaction

Request

The `<rollbackRequest>` message is sent to abort a database transaction. Any updates are rolled back before closing the transaction.

Request Format

The request must be inserted between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP request message, as shown in *SOAP Request Messages*.

```
<rollbackRequest> </rollbackRequest>
```

Response

The rollback response is returned as a generic `<ns2:sdsResult>` response. This response returns the results of rolling back (aborting) a database transaction. The rollback request causes the transaction to end regardless of whether any updates were actually made to the database.

Note: The affected row count in the SOAP response is always 0. The affected row count does not indicate how many rows were modified within the transaction.

If the response error code indicates success, then the database transaction was successfully aborted. If any failure response is returned, then the database rollback failed.

Response Format

The response is displayed between the `<soapenv:Body>` and `</soapenv:Body>` XML tags, as shown in *SOAP Response Messages*.

```
<ns2:sdsResult affected="affected" error="error" [description="description"]>
</ns2:sdsResult>
```

Response Parameters

The parameters for all of the response commands are shown in *SOAP Response Messages*.

Response Error Codes

Table 21: <rollback> Response Error Codes (SOAP) lists common error codes for the rollback response. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 21: `<rollback>` Response Error Codes (SOAP)

Error Code	Description
SUCCESS	Transaction was successfully rolled back.
NO_ACTIVE_TXN	A read or write transaction is already open on this connection or an open transaction was aborted prior to terminating the connection.

Examples

These examples show the SDS provisioning request and response contents that are stored within the `<soapenv:Body>` or `<SOAP-ENV:Body>` tags. See the Start Transaction *Examples* for examples that contain the entire SOAP request/response text.

Rollback a Transaction (success)

This example successfully rolls back a transaction.

Request:

```
<rollbackRequest> </rollbackRequest>
```

Response:

```
<ns2:sdsResult affected="15" error="0">
</ns2:sdsResult>
```

Rollback a Transaction that is not Open (fail)

This example attempts to rollback a transaction but fails because a transaction was not open.

Request:

```
<rollbackRequest> </rollbackRequest>
```

Response:

```
<ns2:sdsResult affected="0" error="1009">
</ns2:sdsResult>
```

Update Subscriber

Subscriber and Routing Data

A routing entity contains the IMSI or MSISDN value along with up to eight destination names that refer to destination data which contains FQDN and realm values that are used for routing messages.

A subscriber is a group of related IMSI and/or MSISDN routing entities and an optional Account ID value. All routing entities within a subscriber have the same destination values.

A stand-alone routing entity is a routing entity that is not assigned to any subscriber.

Each IMSI or MSISDN routing entity is either a stand-alone routing entity or is assigned to a single subscriber.

Request

The `<updateSubscriber>` request provisions IMSI and MSISDN routing data and can provision subscriber data. See *Subscriber and Routing Data* for a description of subscriber and routing data.

This request provisions stand-alone IMSI and MSISDN routing entities and/or subscriber data. Each routing entity contains up to eight destination names. Each destination contains FQDN and realm values, which are used for routing messages.

When the `group="y"` attribute is specified, the request establishes or removes relationships between IMSI, MSISDN and Account ID values. When adding new IMSI or MSISDN values to a subscriber, the request can also create a new IMSI or MSISDN routing entity. When an IMSI or MSISDN value is removed from a subscriber, the request deletes IMSI or MSISDN routing entities. Once a subscriber is created, all subsequent subscriber requests can use any of the subscriber's IMSI, MSISDN or Account ID values to update, delete or read the subscriber.

The request can also be used to update destination names in existing routing entities or create new routing entities, regardless of whether the `group="y"` attribute is specified. These destination changes are applied to all specified IMSI and MSISDN routing entities. If all of the specified IMSI, MSISDN and Account ID values are assigned to one subscriber, the destination changes are also applied to all of the subscriber's routing entities.

If the `group="y"` attribute is specified, then the changes are also applied to any specified new or existing standalone routing entities, which are assigned to the subscriber.

The request can also be used to remove a destination value from existing IMSI and/or MSISDN routing entities by specifying `"none"` as the destination name.

Semantic Rules (all requests)

- Each IMSI and MSISDN routing entity can be assigned to a maximum of 1 subscriber.
- All specified destination names must already exist in the database.
- Each destination name type can only be specified once.
- Any existing destination(s) for a routing entity will not be changed/removed if not specified in the request.
- Specifying a destination name of `"none"` removes the association of that destination from the specified routing entity(s).

Semantic Rules (requests that do not specify the group attribute or specify `group="n"`)

- The `accountId`, `deleteAccountId`, `deleteImsi`, and `deleteMsisdn` parameters cannot be specified.
- All specified existing IMSI and MSISDN values must be for stand-alone routing entities or must all be assigned to one subscriber. There cannot be a mixture of stand-alone routing entities and routing entities that are part of a subscriber.
- At least one routing entity (IMSI or MSISDN) value must be specified within the `addressList`.

- A maximum of 10 routing entities (IMSI, MSISDN, or combinations) can be specified within the `addressList`.
- At least one destination must be specified within the `destionationList`.
- All specified routing entities will be provisioned with the same destination value(s).

Semantic Rules (requests that specify `group="y"`)

- The accountId, `deleteAccountId`, `deleteImsi`, and `deleteMsisdn` parameters can be specified.
- All specified, existing `accountID`, `imsi`, or `msisdn` values must be assigned to the same subscriber or they can exist in a stand-alone routing entity. After the command successfully completes, all specified values are assigned to the same subscriber.
- All specified addresses within the `deleteAccountId`, `deleteImsi`, and `deleteMsisdn` tags that exist in the database must be assigned to the same subscriber. All specified addresses within the addressList (Account ID, IMSI, or MSISDN values) must also be assigned to the same subscriber or not assigned to any subscriber.
- At least one `imsi`, `msisdn`, or `accountId` value must be specified within the `addressList`.
- The destinationList tag is mandatory, but no values are required within it. This allows the user to add an Account ID or existing MSISDN and/or IMSI values to a subscriber.
- The addressList can have a maximum of one `accountId`, six `imsi`, six `msisdn`, one `deleteAccountId`, six `deleteImsi`, and/or six `deleteMsisdn` values specified. If any of these limits are exceeded, the request fails.
- All `accountId`, `imsi`, and `msisdn` values specified within the `addressList` that are not currently associated with a subscriber will be assigned to the same subscriber. They are added to an existing subscriber or new subscriber.
- If a new subscriber is being created with all new routing entities, all specified routing entities will be provisioned with the specified destination values
- If a new subscriber is being created with at least one existing stand-alone routing entity, all destination values from existing stand-alone routing entities must be the same prior to applying any specified destination changes. All new routing entities will inherit their destination values from an existing stand-alone routing entity with the applied destination changes.
- If existing stand-alone routing entities are being added to an existing subscriber, the destination values in each stand-alone routing entity must match the destination values for the subscriber (extracted from any of the subscriber routing entities) prior to applying any specified destination changes.
- If new routing entities are being added to an existing subscriber, the new routing entities will inherit the destination values for the subscriber (extracted from any of the subscriber routing entities).
- If a new routing entity is being created, at least one of its destination values cannot be equal to `"none"`.
- The updated subscriber must have at least one IMSI or MSISDN routing entity.
- The updated subscriber can have 0 or 1 `accountID` values, 0-6 `imsi` values, and 0-6 `msisdn` values, as long as there is at least 1 IMSI or MSISDN value. If the result of the update (deleting and then adding Account ID, IMSI and/or MSISDN values) would cause too many Account ID, IMSI or MSISDN values, the request will fail.
- The subscriber Account ID value can be updated only if it is currently `null` or deleted within the request (as specified by the `deleteAccountId` parameter).
- If any of the `deleteAccountId`, `deleteImsi`, or `deleteMsisdn` values do not exist in the database, they will be ignored. If nothing else changes for the subscriber, the NO_UPDATES is returned.

- If any of the `deleteAccountId`, `deleteImsi`, or `deleteMsisdn` values exist in the database, they must be assigned to the subscriber being updated or the command will fail.
- If any of the `deleteIMSI` or `deleteMsisdn` values exist, the routing entity will be deleted unless it is the last IMSI or MSISDN routing entity for the subscriber, in which case the command will fail.

Request Format

The request must be inserted between the `<soapenv:Body>` and `</soapenv:Body>` XML tags as shown in *SOAP Request Messages*.

```
<updateSubscriberRequest [timeout="timeout"] [group="group"]>

    <addressList>
    [   <deleteAccountId>deleteAccountId</deleteAccountId> ]
    [
        <deleteImsi>deleteImsi</deleteImsi>
        …
        <deleteImsi>deleteImsi</deleteImsi>
    ]
    [
        <deleteMsisdn>deleteMsisdn</deleteMsisdn>
        …
        <deleteMsisdn>deleteMsisdn</deleteMsisdn>
    ]
    [   <accountId>accountId</accountId> ]
    [
    <imsi>imsi</imsi>
        …
    <imsi>imsi</imsi>
    ]
    [
    <msisdn>msisdn</msisdn>
        …
    <msisdn>msisdn</msisdn>
    ]
    </addressList>
    <destinationList>
    [   <imshss>imshss</imshss>       ]
    [   <ltehss>ltehss</ltehss>       ]
    [   <pcrf>pcrf</pcrf>             ]
    [   <ocs>ocs</ocs>                ]
    [   <ofcs>ofcs</ofcs>             ]
    [   <aaa>aaa</aaa>                ]
    [   <userdef1>userdef1</userdef1> ]
    [   <userdef2>userdef2</userdef2> ]
    </destinationList>
</updateSubscriberRequest>
```

Request Parameters

Table 22: `<updateSubscriberRequest>` Parameters (SOAP)

Parameter	Description	Values
timeout (Optional)	The amount of time (in seconds) to wait before being able to perform a write if another connection is performing a write,	0 (return immediately if not available) to 3600 seconds (default is 0).

Parameter	Description	Values
	or has a transaction open. Clients waiting to write will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	
group (Optional)	Indicates if relationships between a group of related IMSI and/or MSISDN routing entities and Account ID value should be created/updated.	• y - Create new or update existing subscriber relationships and update destinations. • n - Only update destinations, not relationships between routing entities. (default)
addressList	XML tag that contains a list of addresses to be created or updated.	Must have at least 1 of the following tags and values: • 0-1 - deleteAccountId • 0-6 - deleteImsi • 0-6 - deleteMsisdn • 0-1 - accountId • 0-6 - imsi • 0-6 - msisdn
deleteAccountId (Optional)	A user-defined Account ID value to delete.	1 to 26 numeric digits.
deleteImsi (Optional)	An IMSI (specified in E.212 format) value to delete.	10 to 15 numeric digits.
deleteMsisdn (Optional)	An MSISDN (specified in E.164 international public telecommunication numbering plan format) value to delete.	8 to 15 numeric digits.
accountId (Optional)	A user-defined Account ID value to add or update.	1 to 26 numeric digits.
imsi (Optional)	An IMSI (specified in E.212 format) value to add or update.	10 to 15 numeric digits.
msisdn (Optional)	An MSISDN (specified in E.164 international public telecommunication numbering plan format) value to add or update.	8 to 15 numeric digits.

Parameter	Description	Values
destinationList (Optional)	XML tag that contains a list of destination values to update or set in the routing entity(s).	Can be empty, or contain any of the following destination tags and values: imshss, ltehss, pcrf, ocs, ofcs, aaa, userdef1, and/or userdef2
imshss (Optional)	The name of the IMS HSS destination.	A string with 1 to 32 characters.
ltehss (Optional)	The name of the LTE HSS destination.	A string with 1 to 32 characters.
pcrf (Optional)	The name of the PCRF destination.	A string with 1 to 32 characters.
ocs (Optional)	The name of the OCS destination.	A string with 1 to 32 characters.
ofcs (Optional)	The name of the OFCS destination.	A string with 1 to 32 characters.
aaa (Optional)	The name of the AAA server destination.	A string with 1 to 32 characters.
userdef1 (Optional)	The name of the first user defined destination	A string with 1 to 32 characters.
userdef2 (Optional)	The name of the second user defined destination	A string with 1 to 32 characters.

Response

The update subscriber response is returned as a generic `<ns2:sdsResult>` response. This response returns the result of the request to provision subscriber and/or routing entities. A single result applies to all routing entities supplied. Either all subscriber and/or routing entities were successfully updated, or no updates were made.

If applying all of the provisioning changes results in no database records being modified because the database already contained the updated values, the NO_UPDATES error code is returned and the number of affected records is 0.

If a subscriber is successfully created or updated, the description field contains lists of deleted, created and changed IMSI and MSISDN values.

Response Format

The response is displayed between the `<soapenv:Body>` and `</soapenv:Body>` XML tags as shown in *SOAP Response Messages*.

```
<ns2:sdsResult affected="affected" error="error" [description="description"]>
</ns2:sdsResult>
```

Response Parameters

The parameters for all of the response commands are shown in *SOAP Response Messages*.

Response Error Codes

Table 23: Update Subscriber Response Error Codes (SOAP) lists common error codes for this command. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 23: Update Subscriber Response Error Codes (SOAP)

Error Code	Description
SUCCESS	The update request was successfully completed.
NO_UPDATES	The write transaction did not have any successful updates.
DEST_NOT_FOUND	Destination name does not exist.
DEST_TYPE_MISMATCH	Destination has a different destination type than the desired destination type.
MULTIPLE_SUBSCRIBERS	Specified parameters refer to multiple subscribers.
SUBSCRIBER_TOO_BIG	Resulting subscriber would exceed 6 IMSI or 6 MSISDN limit.
ACCTID_UPDATE_PROHIBITED	An attempt was made to change an accountId without specifying `<deleteAccountId>` tag.
ROUTE_TYPE_MISMATCH	Standalone and subscriber routes are not allowed in same command.
DEL_ROUTE_NOT_PERMITTED	Cannot delete last route from subscriber.
NO_ROUTES_SPECIFIED	At least one MSISDN or IMSI must be specified.
ROUTE_DEST_MISMATCH	Specified routes have different destinations.

Examples

Below are examples of how to use the `<updateSubscriber>` request and likely response. Some of these examples are based on previous requests; hence, the order of the requests could be important.

These examples show the SDS provisioning request and response contents that are stored within the `<soapenv:Body>` or `<SOAP-ENV:Body>` tags. See the Start Transaction *Examples* for examples of the entire SOAP request/response text.

Add Stand-Alone Routing Entities

This example creates new stand-alone IMSI and MSISDN routing entities and sets their destination values to the specified values.

The result of this request is:

- New IMSI and MSISDN routing entities are created.

- All of the destination values for each routing entity are set to specified values.

Request:

```
<updateSubscriberRequest>
  <addressList>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <imsi>111111111100003</imsi>
    <msisdn>8004605500</msisdn>
    <msisdn>8004605503</msisdn>
  </addressList>
  <destinationList>
    <ltehss>LTE_HSS_1</ltehss>
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="5" error="0"
</ns2:sdsResult>
```

Update Stand-Alone Routing Entities Destinations

This example updates existing stand-alone IMSI and MSISDN routing entities with new destination values.

Note: This request does not update all NAI values that were specified in the previous request.

The result of this request is that the IMSI and MSISDN routing entities are updated with specified values.

Request:

```
<updateSubscriberRequest>
  <addressList>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <imsi>111111111100003</imsi>
    <msisdn>8004605500</msisdn>
  </addressList>
  <destinationList>
    <ltehss>LTE_HSS_4</ltehss>
    <aaa>AAA_4</aaa>
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="4" error="0"
</ns2:sdsResult>
```

Create Subscriber Using Existing Routing Entities (Success)

This example creates a subscriber using existing routing entities that all have the same destination values.

After this request is completed, a new subscriber is created and all of the routing entities are assigned to that subscriber.

Request:

```
<updateSubscriberRequest group=”y”>
  <addressList>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
  </addressList>
  <destinationList>
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="1" error="0">
</ns2:sdsResult>
```

Create Subscriber Using Existing Routing Entities (Failure)

This example fails when creating a subscriber using existing routing entities because the existing routing entities have different destination values.

No changes are made to the database because the request failed.

Request:

```
<updateSubscriberRequest group=”y”>
  <addressList>
    <imsi>111111111100003</imsi>
    <msisdn>8004605503</msisdn>
  </addressList>
  <destinationList>
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="0" error="2029" description=
”all routes must have the same destination values”>
</ns2:sdsResult>
```

Add Account ID to Existing Subscriber

This example adds an Account ID to an existing subscriber. Any of the subscriber IMSI or MSISDN values can be used. For this example, the MSISDN value is used.

The result of this request is that the subscriber will have an Account ID value.

Request:

```
<updateSubscriberRequest group=”y”>
  <addressList>
    <accountId>800444000012345678901111112</accountId>
    <msisdn>8004605500</msisdn>
  </addressList>
  <destinationList>
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="1" error="0">
</ns2:sdsResult>
```

Modify Destinations for Existing Subscriber

This example modifies one of the destination values for an existing subscriber. Any of the subscriber's IMSI, MSISDN or Account ID values can be used. For this example, an IMSI value is used.

Note: It does not matter if group="y" is specified. The same changes are always applied to the whole subscriber.

The result of this request is that all of the subscriber's IMSI and MSISDN routing entities will have a new destination value.

Request:

```
<updateSubscriberRequest>
  <addressList>
    <imsi>111111111100002</imsi>
   </addressList>
   <destinationList>
     <ltehss>LTE_HSS_99</ltehss>
   </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="3" error="0"
</ns2:sdsResult>
```

Replace MSISDN value

This example replaces an MSISDN value for an existing subscriber. The new MSISDN routing entity inherits the destination values from an old IMSI or MSISDN routing entity. It does not matter which of the subscriber's routing entities is used. All entities have the same destination values.

The result of this request is:

- The old MSISDN routing entity is deleted from the database
- The new MSISDN routing entity is added to the database, its destination values are set to the subscriber destination values, and the new MSISDN value is assigned to the subscriber (relationships are established).

  Note: If the new MSISDN routing entity already exists in the database, and it has the same destination values as the subscriber, the only change is that the routing entity is assigned to the subscriber.

Request:

```
<updateSubscriberRequest group=”y”>
  <addressList>
    <deleteMsisdn>8004605500</deleteMsisdn>
    <msisdn>8884605500</msisdn>
  </addressList>
  <destinationList>
```

```
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="1" error="0">
</ns2:sdsResult>
```

Replace Account ID, Two IMSI values, and One MSISDN Value

This example replaces several identification values for an existing subscriber. The new IMSI and MSISDN routing entities inherit the destination values from the old IMSI and MSISDN routing entities. It does not matter which of the Subscriber's routing entities is used. All routing entities have the same destination values.

The result of this request is:

- The old IMSI and MSISDN routing entities are deleted from the database.
- The new IMSI and MSISDN routing entities are added to the database, their destination values are set to the subscriber's destination values, and the routing entities are assigned to the subscriber (relationships are established).

  Note: If the new IMSI and MSISDN routing entities already exist in the database and they have the same destination values as the subscriber, the only change is that the new IMSI and MSISDN values are assigned to the subscriber.

- The subscriber Account ID value is changed.

Request:

```
<updateSubscriberRequest>
  <addressList>
    <deleteImsi>111111111100001</deleteImsi>
    <deleteImsi>111111111100002</deleteImsi>
    <deleteMsisdn>8884605500</deleteMsisdn>
    <imsi>888888888800001</imsi>
    <imsi>888888888800002</imsi>
    <msisdn>8884605555</msisdn>
  </addressList>
  <destinationList>
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="1" error="0">
</ns2:sdsResult>
```

Create Subscriber Using New Routing Entities (Success)

This example creates a subscriber using new routing entities with specified destinations.

The result of this request is:

- A new subscriber is created with the specified Account ID, IMSI and MSISDN values.
- New IMSI and MSISDN routing entities are created with the specified destinations.

Request:

```
<updateSubscriberRequest group=”y”>
  <addressList>
    <accountId>111112222233333444445555556</accountId>
    <imsi>333333333300001</imsi>
    <imsi>333333333300002</imsi>
    <msisdn>9198675309</msisdn>
  </addressList>
  <destinationList>
    <ltehss>LTE_HSS_3</ltehss>
    <aaa>AAA_3</aaa>
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="1" error="0">
</ns2:sdsResult>
```

Create Subscriber Using New Routing Entities (Failure)

This example fails when creating a subscriber using new routing entities because no destinations were specified.

No changes are made to the database because the request failed.

Request:

```
<updateSubscriberRequest group=”y”>
  <addressList>
    <accountId>1111122222</accountId>
    <imsi>333333333300003</imsi>
    <imsi>333333333300004</imsi>
    <msisdn>9198675309</msisdn>
  </addressList>
  <destinationList>
  </destinationList>
</updateSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="0" error="2013" description=
”at least one destination must be specified”>
</ns2:sdsResult>
```

Delete Subscriber

Request

The `<deleteSubscriberRequest>` request removes IMSI and MSISDN routing data and can remove subscriber data. See *Subscriber and Routing Data* for a description of subscriber and routing data. Each routing entity contains up to eight destination names. Each destination contains FQDN and realm values.

If the `group="y"` attribute is specified, then the request deletes all data associated with the subscriber. The Account ID, all relationships, and all IMSI and MSISDN routing entities that were assigned to the subscriber are deleted.

If `group="y"` is not specified or if `group="n"` is specified, only IMSI and MSISDN routing entities are deleted. If the IMSI or MSISDN value is assigned to a subscriber and there is at least one more IMSI or MSISDN value assigned to the subscriber, then the IMSI or MSISDN value is removed from the subscriber.

Note: The last IMSI or MSISDN value cannot be removed from a subscriber. The user must delete the whole subscriber by specifying the `group="y"` attribute.

Semantic Rules (requests that do not specify the group attribute or specify `group="n"`)

- All specified `imsi` or `msisdn` values must be assigned to one subscriber or exist as a stand-alone routing entity.
- The `accountID` parameter cannot be specified.
- At least one routing entity (IMSI or MSISDN) must be specified.
- A maximum of 10 routing entities (IMSI, MSISDN, or combinations of the two) can be specified.
- The last IMSI or MSISDN for a subscriber cannot be deleted. Use `group="y"` to delete the whole subscriber.

Semantic Rules (requests that specify `group="y"`)

- All specified `accountId`, `imsi`, or `msisdn` values must be assigned to the same subscriber. The specified `imsi` or `msisdn` values cannot exist in a stand-alone routing entity.
- The `accountId` parameter can be specified.
- At least one `accountId`, `imsi`, or `msisdn` value must be specified.
- A maximum of six `imsi`, six `msisdn`, and one `accountId` values can be specified.

Request Format

The request must be inserted between the `<soapenv:Body>` and `</soapenv:Body>` XML tags, as shown in *SOAP Request Messages*.

```
<deleteSubscriberRequest [timeout="timeout"] [group="group"]>
    <addressList>
    [
        <accountId>accountId</accountId>
    ]
    [
    <imsi>imsi</imsi>
        …
    <imsi>imsi</imsi>
    ]
    [
    <msisdn>msisdn</msisdn>
        …
    <msisdn>msisdn</msisdn>
]
    </addressList>
</deleteSubscriberRequest>
```

Request Parameters

Table 24: `<deleteSubscriberRequest>` Parameters (SOAP)

Parameter	Description	Values
timeout (Optional)	The amount of time (in seconds) to wait before being able to perform a write if another connection is performing a write, or has a transaction open. Clients waiting to write will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	0 (return immediately if not available) to 3600 seconds (default is 0).
group (Optional)	Indicates if all of the subscriber's data should be deleted or just specified IMSI or MSISDN routing entities.	• y - Delete subscriber and all of its IMSI and MSISDN routing entities. • n - Only delete specified IMSI and MSISDN routing entities (default).
accountId (Optional)	A user-defined Account ID value to delete.	1 to 26 numeric digits.
imsi (Optional)	An IMSI (specified in E.212 format) value to delete.	10 to 15 numeric digits.
msisdn (Optional)	An MSISDN (specified in E.164 format) value to delete.	8 to 15 numeric digits.

Response

The delete Subscriber response is returned as a generic `<ns2:sdsResult>` response. This response returns the result of the request to delete subscriber and/or routing entities. A single result applies to all routing entities supplied. Either all subscriber and/or routing entities were successfully deleted, or no deletes were made.

If applying all of the delete changes results in no routing entities being deleted (because the database already did not contain the specified values), the NO_UPDATES error code is returned and the number of affected records is 0.

If a subscriber is successfully deleted, the `description` field contains lists of deleted IMSI and MSISDN values.

Response Format

The response is displayed between the <soapenv:Body> and </soapenv:Body> XML tags, as shown in *SOAP Response Messages*.

```
<ns2:sdsResult affected="affected" error="error" [description="description"]>
</ns2:sdsResult>
```

Response Error Codes

Table 25: <deleteSubscriberResponse> Error Codes (SOAP) displays common error codes for the <deleteSubscriber> response. See *SDS Response Message Error Codes* for a full list of error codes.

Table 25: <deleteSubscriberResponse> Error Codes (SOAP)

Error Code	Description
SUCCESS	The delete request was successfully completed.
NO_UPDATES	The records were already deleted from the database.
MULTIPLE_SUBSCRIBERS	Specified parameters refer to multiple subscribers.
ROUTE_TYPE_MISMATCH	Standalone and subscriber routes are not allowed in same command.
DEL_ROUTE_NOT_PERMITTED	The last route from a subscriber cannot be deleted.

Examples

These examples show the SDS provisioning request and response contents that are stored within the <soapenv:Body> or <SOAP-ENV:Body> tags. See Start Transaction *Examples* for an example of the whole SOAP request/response text.

Delete Stand-Alone Routing Entities

This example deletes stand-alone IMSI and MSISDN routing entities.

Request:

```
<deleteSubscriberRequest>
  <addressList>
    <imsi>111111111100021</imsi>
    <imsi>111111111100022</imsi>
    <msisdn>8004605520</msisdn>
  </addressList>
</deleteSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="3" error="0">
</ns2:sdsResult>
```

Delete Several Routing Entities

This example successfully deletes two stand-alone IMSI routing entities. Other IMSI values were not found and were not deleted.

Request:

```
<deleteSubscriberRequest>
  <addressList>
    <imsi>777777777777777</imsi>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <imsi>888888888888888</imsi>
  </addressList>
</deleteSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="2" error="0">
</ns2:sdsResult>
```

Delete Routing Entities Assigned to the Same Subscriber

This example deletes IMSI and MSISDN routing entities that are assigned to the same subscriber. The example assumes that the subscriber has at least one more routing entity other than the specified values.

Request

```
<deleteSubscriberRequest>
  <addressList>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
  </addressList>
</deleteSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="2" error="0”>
</ns2:sdsResult>
```

Delete Last Routing Entity for a Subscriber (success)

This example successfully deletes the subscriber and all IMSI and MSISDN routing entities assigned to the subscriber. Any of the subscriber's Account ID, MSISDN or IMSI values can be specified. In this example, all of the IMSI and MSISDN values are specified even though only 1 value is required.

Request:

```
<deleteSubscriberRequest timeout="10" group=”y”>
  <addressList>
     <imsi>111111111100001</imsi>
     <imsi>111111111100002</imsi>
     <msisdn>8004605500</msisdn>
  </addressList>
</deleteSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="1" error="0”>
</ns2:sdsResult>
```

Delete Last Routing Entity for a Subscriber (failure)

This example attempts to delete IMSI and MSISDN routing entities that are assigned to the same subscriber. The example fails because the subscriber does not have any more routing entities.

No changes are made to the database because the request failed.

Request:

```
<deleteSubscriberRequest timeout="10">
  <addressList>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
  </addressList>
</deleteSubscriberRequest>
```

Response:

```
<ns2:sdsResult description=”cannot delete the last route from subscriber"
affected="0" error="2027">
</ns2:sdsResult>
```

Delete a Subscriber (success)

This example successfully deletes the subscriber and all IMSI and MSISDN routing entities assigned to the subscriber. Any of the subscriber's Account ID, MSISDN or IMSI values can be specified. In this example, the Account ID is specified.

Request:

```
<deleteSubscriberRequest timeout="10" group=”y”>
  <addressList>
    <accountId>800444000012345678901111112</accountId>
  </addressList>
</deleteSubscriberRequest>
```

Response:

```
<ns2:sdsResult affected="1" error="0">
</ns2:sdsResult>
```

Read Subscriber

Request

The `<readSubscriberRequest>` request extracts IMSI and MSISDN routing data and subscriber data. See *Subscriber and Routing Data* for a description of subscriber and routing data. Each routing entity contains up to eight destination names.

If the `group="y"` attribute is specified, then the request extracts and displays all data associated with the subscriber. The returned response will have the Subscriber's Account ID, all IMSI and MSISDN values, and the eight destination values from any of the subscriber's routing entities is returned in the response. All of a subscriber's routing entities have the same destination values, so any routing entity can be used to extract the values.

If `group="y"` is not specified or if `group="n"` is specified, then only the specified IMSI and MSISDN routing entities are retrieved. The returned response will have each IMSI or MSISDN value along with its individual up to eight destination values.

Semantic Rules (requests that do not specify the group attribute or specify `group="n"`)

- The `accountId` parameter cannot be specified.
- At least one routing entity (IMSI or MSISDN) must be specified.
- A maximum of 10 routing entities (IMSI, MSISDN, or combinations of the two) can be specified.

Semantic Rules (requests that specify `group="y"`)

- All specified `accountId`, `imsi`, or `msisdn` values must be assigned to one subscriber. The specified IMSI and MSISDN values cannot exist in a stand-alone routing entity.
- The `accountId` parameter can be specified.
- A maximum of six `imsi`, six `msisdn`, and one `accountId` values can be specified.

Request Format

The request must be inserted between the `<soapenv:Body>` and `</soapenv:Body>` XML tags, as shown in *SOAP Request Messages*.

```
<readSubscriberRequest [group="group"]>
    <addressList>
    [
        <accountId>accountId</accountId>
    ]
    [
    <imsi>imsi</imsi>
        …
    <imsi>imsi</imsi>
    ]
    [
    <msisdn>msisdn</msisdn>
        …
    <msisdn>msisdn</msisdn>
]
```

```
    </addressList>
</readSubscriberRequest>
```

Request Parameters

Table 26: `<readSubscriberRequest>` Parameters (SOAP)

Parameter	Description	Values
group (Optional)	Indicates if all of the subscriber's data should be retrieved or just specified IMSI or MSISDN routing entities.	• y - Read subscriber and all of its IMSI and MSISDN routing entities. • n - Only read specified MSISDN and IMSI routing entities (default).
accountId (Optional)	A user-defined Account ID value to read.	1 to 26 numeric digits.
imsi (Optional)	An IMSI (specified in E.212 format) value to read.	10 to 15 numeric digits.
msisdn (Optional)	An MSISDN (specified in E.164 format) value to read.	8 to 15 numeric digits.

Response

The `<readSubscriberResponse>` message returns the result of the request to read subscriber routing entities. Only those subscribers or routing entities that are found are returned. The response message contains up to eight destinations (one for each destination type, such as `<ltehss>`) for each routing entity or subscriber. Only provisioned destination names are displayed. (i.e. destination names= *"none"* are not displayed).

Some variations in the response occur, depending on whether a subscriber or routing entities are being retrieved.

Routing entities are retrieved (`group="y"` is not specified or `group="n"` is specified):

- No `<subscriber>` or `<accountId>` tags are used.
- The destination values are listed within each IMSI or MSISDN routing entity value.

A subscriber is retrieved (`group="y"` was specified):

- The `<subscriber>` tag is used within the `<resultSet>` tag.
- The `<accountId>` tag is displayed if the subscriber has an Account ID value defined.
- The destination values are listed one time, after the last routing entity.

Response Format (group="y" is not specified)

The response is displayed between the <soapenv:Body> and </soapenv:Body> XML tags of a SOAP response message, as shown in *SOAP Response Messages*.

```
<readSubscriberResponse>
    <result affected="affected" error="error" [description="description"]>
    </result>
[
    <resultSet>
    [
        <imsi imsi="imsi">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </imsi>
              ...
        <imsi imsi="imsi">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </imsi>
    ]
    [
        <msisdn msisdn="msisdn">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </msisdn>
              ...
        <msisdn msisdn="msisdn">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </msisdn>
    ]
    </resultSet>
]
</readSubscriberResponse>
```

Response Format (group="y" is specified)

The response is displayed between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP response message, as shown in *SOAP Response Messages*.

```
<readSubscriberResponse>
    <result affected="affected" error="error" [description="description"]/>
[
    <resultSet>
        <subscriber>
        [   <accountId>accountId</accountId>]
        [
        [   <imsi>imsi</imsi>             ]
                ...
        [   <imsi>imsi</imsi>             ]
        [   <msisdn>msisdn</msisdn>       ]
                ...
        [   <msisdn>msisdn</msisdn>       ]
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2>
        </subscriber>
    </resultSet>
]
</readSubscriberResponse>
```

Response Parameters

Table 27: `<readSubscriberResponse>` Parameters (SOAP)

Parameter	Description	Values
group (Optional)	Indicates if all of the subscriber's data should be retrieved or just specified IMSI or MSISDN routing entities.	y - Read subscriber and all of its IMSI and MSISDN routing entities. n - Only read specified MSISDN and IMSI routing entities (default).
error	Error code that indicates whether or not operation was successfully executed.	0 for success, non-zero for failure.
affected	The number of routing entities or subscribers (for group="y") read.	0-12
description	A textual description associated with the response. This field may contain more information as to why a request failed.	A string with 1 to 1024 characters.

Parameter	Description	Values
resultSet	Contains 1 row for each extracted record. Each row contains a stand-alone routing entity (MSISDN or IMSI value and its destination values) or a subscriber (list of related MSISDN, IMSI and Account ID values and the destination values used by all routing entities assigned to the subscriber).	
subscriber (Optional)	Contains all of the IMSI and MSISDN values for a specific subscriber, an optional Account ID, and all destinations defined for the subscriber.	
accountId (Optional)	A user-defined Account ID value.	1 to 26 numeric digits.
imsi (Optional)	An IMSI (specified in E.212 format) value.	10 to 15 numeric digits.
msisdn (Optional)	An MSISDN (specified in E.164 international public telecommunication numbering plan format) value.	8 to 15 numeric digits.
imshss (Optional)	The name of the IMS HSS destination.	A string with 1 to 32 characters.
ltehss (Optional)	The name of the LTE HSS destination.	A string with 1 to 32 characters.
pcrf (Optional)	The name of the PCRF destination.	A string with 1 to 32 characters.
ocs (Optional)	The name of the OCS destination.	A string with 1 to 32 characters.
ofcs (Optional)	The name of the OFCS destination.	A string with 1 to 32 characters.
aaa (Optional)	The name of the AAA server destination.	A string with 1 to 32 characters.
userdef1 (Optional)	The name of the first user defined destination.	A string with 1 to 32 characters.
userdef2 (Optional)	The name of the second user defined destination.	A string with 1 to 32 characters.

Error Codes

Table 28: <readSubscriberResponse> Error Codes (SOAP) lists common errors for the `<readSubscriberResponse>` command. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 28: `<readSubscriberResponse>` Error Codes (SOAP)

Error Code	Description
SUCCESS	The read request was successfully completed.
IMSI_NOT_FOUND	IMSI does not exist.
MSISDN_NOT_FOUND	MSISDN does not exist.
SUBSCIRIBER_NOT_FOUND	The subscriber could not be found in the database.
MULTIPLE_SUBSCRIBERS	Specified parameters refer to multiple subscribers.
ROUTE_TYPE_MISMATCH	Standalone and subscriber routes are not allowed in same command.
NO_ROUTES_SPECIFIED	At least one MSISDN or IMSI must be specified.

Examples

These examples show the SDS provisioning request and response contents that are stored within the `<soapenv:Body>` or `<SOAP-ENV:Body>` tags. See Start Transaction *Examples* for an example of the whole SOAP request/response text.

The format of the response differs depending on whether the group="y" attribute is specified.

If group="y" is not specified, then each routing entity that was found is displayed with its destination values.

If group="y" is specified, then the result response includes an optional Account ID value, all MSISDN and IMSI values for that subscriber, and one set of destination values (all routing entities within a subscriber have the same destination values).

Read Routing Entities (not subscribers)

This example reads IMSI and MSISDN routing entities and displays their destination values. It does not matter if any of the routing entities are assigned to a subscriber because the same result will occur.

Request:

```
<readSubscriberRequest>
  <addressList>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
  </addressList>
</readSubscriberRequest>
```

Response:

```
<ns3:readSubscriberResponse>
  <result affected="3" error="0"></result>
  <resultSet>
    <imsi imsi="111111111100001">
      <ltehss>LTE_HSS_4</ltehss>
      <aaa>AAA_4</aaa>
    </imsi>
    <imsi imsi="111111111100002">
      <ltehss>LTE_HSS_4</ltehss>
      <aaa>AAA_4</aaa>
    </imsi>
    <msisdn msisdn="8004605500">
      <ltehss>LTE_HSS_4</ltehss>
      <aaa>AAA_4</aaa>
    </msisdn>
  </resultSet>
</ns3:readSubscriberResponse>
```

Read Routing Entities with Not Found MSISDN/IMSI Values

This example reads IMSI and MSISDN routing entities and displays their destination values. In this example, one MSISDN and one IMSI value do not exist, so the response returns the two values that do exist. The same result will occur if any of the routing entities are assigned to a subscriber.

Request:

```
<readSubscriberRequest>
  <addressList>
    <imsi>777777777777777</imsi>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
    <msisdn>88888888888888</msisdn>
  </addressList>
</readSubscriberRequest>
```

Response:

```
<ns3:readSubscriberResponse>
  <result affected="2" error="0"></result>
  <resultSet>
    <imsi imsi="111111111100002">
      <ltehss>LTE_HSS_4</ltehss>
      <aaa>AAA_4</aaa>
    </imsi>
    <msisdn msisdn="8004605500">
      <ltehss>LTE_HSS_4</ltehss>
      <aaa>AAA_4</aaa>
    </msisdn>
  </resultSet>
</ns3:readSubscriberResponse>
```

Read Subscriber (success)

This example reads a subscriber and displays all of the subscriber data. Any of the subscriber Account ID, MSISDN or IMSI values can be specified. In this example, the MSISDN value is specified.

Request:

```
<readSubscriberRequest group=”y”>
  <addressList>
    <msisdn>8004605500</msisdn>
  </addressList>
</readSubscriberRequest>
```

Response:

```
<ns3:readSubscriberResponse>
  <result affected="1" error="0"></result>
  <resultSet>
    <subscriber>
      <accountId>800444000012345678901111112</accountId>
      <imsi>111111111100001</imsi>
      <imsi>111111111100002"</imsi>
      <msisdn>8004605500</msisdn>
      <ltehss>LTE_HSS_4</ltehss>
      <aaa>AAA_4</aaa>
    </subscriber>
  </resultSet>
</ns3:readSubscriberResponse>
```

Read Subscriber Fails for Stand-alone Routing Entity

This example attempts to read a subscriber. The request fails because the specified MSISDN value is for a stand-alone routing entity.

Request

```
<readSubscriberRequest group=”y”>
  <addressList>
    <msisdn>9198675309</msisdn>
  </addressList>
</readSubscriber>
```

Response:

```
<ns3:readSubscriberResponse>
  <result description=”subscriber not found”
   affected="0" error="2022"></result>
</ns3:readSubscriberResponse>
```

Update Subscriber NAI

Request

The `<updateSubscriberNaiRequest>` provisions NAI routing entities. Each NAI value is defined as a combination of an NAI host and NAI user value. For example, "John.Smith@tekelec.com" would have "John.Smith" as the NAI user value and "tekelec.com" as the NAI host value.

Each routing entity contains up to eight destination names. Each destination contains FQDN and realm values, which are used for routing messages. The request can remove a destination value from existing NAI routing entities by specifying `"none"` as the destination name.

The request can add new routing entities or update destination names in existing routing entities. These destination changes are applied to all specified NAI routing entities.

Semantic Rules

- The host name must already exist in the database.
- Between 1 and 10 user names must be specified.
- At least one destination must be specified.
- All specified destination names must already exist in the database.
- Each destination name type can only be specified once.
- Specifying a destination name of `"none"` removes the association of that destination from the specified routing entity.
- All specified routing entities will be provisioned with the same destination value(s).

Request Format

The request must be inserted between the `<soapenv:Body>` and `</soapenv:Body>` XML tags, as shown in *SOAP Request Messages*.

```
<updateSubscriberNaiRequest [timeout="timeout"]>
  <naiList>
    <host>host</host>
    <user>user</user>
[
    <user>user</user>
          …
    <user>user</user>
]
  </naiList>
  <destinationList>
[   <imshss>imshss</imshss>       ]
[   <ltehss>ltehss</ltehss>       ]
[   <pcrf>pcrf</pcrf>             ]
[   <ocs>ocs</ocs>                ]
[   <ofcs>ofcs</ofcs>             ]
[   <aaa>aaa</aaa>                ]
[   <userdef1>userdef1</userdef1> ]
[   <userdef2>userdef2</userdef2> ]
  <destinationList>
</updateSubscriberNaiRequest>
```

Request Parameters

Table 29: `<updateSubscriberNaiRequest>` Parameters (SOAP)

Parameter	Description	Values
timeout (Optional)	The amount of time (in seconds) to wait before being able to perform a write if another connection is performing a write, or has a transaction open. Clients	0 (return immediately if not available) to 3600 seconds (default is 0).

Parameter	Description	Values
	waiting to write will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	
host	The host name, which is used with all user values.	A string with 1 to 64 characters.
user	The NAI user name to be associated with the host to form an NAI.	A string with 1 to 64 characters. Must have 1-10 user values.
imshss (Optional)	The name of the IMS HSS destination.	A string with 1 to 32 characters.
ltehss (Optional)	The name of the LTE HSS destination.	A string with 1 to 32 characters.
pcrf (Optional)	The name of the PCRF destination.	A string with 1 to 32 characters.
ocs (Optional)	The name of the OCS destination.	A string with 1 to 32 characters.
ofcs (Optional)	The name of the OFCS destination.	A string with 1 to 32 characters.
aaa (Optional)	The name of the AAA server destination.	A string with 1 to 32 characters.
userdef1 (Optional)	The name of the first user defined destination.	A string with 1 to 32 characters.
userdef2 (Optional)	The name of the second user defined destination.	A string with 1 to 32 characters.

Response

The update subscriber NAI response is returned as a generic `<ns2:sdsResult>` response. This response returns the result of the request to provision NAI subscriber routing entities. A single result applies to all routing entities supplied. Either all routing entities were successfully updated, or no updates were made.

Note: If applying all of the provisioning changes results in no database records being modified (because the database already contained the updated values), the NO_UPDATES error code is returned and the number of affected records is 0.

Response Format

The response is displayed between the <soapenv:Body> and </soapenv:Body> XML tags, as shown in *SOAP Response Messages*.

```
<ns2:sdsResult affected="affected" error="error" [description="description"]>
</ns2:sdsResult>
```

Response Parameters

The parameters for all of the SOAP response commands are shown in *SOAP Response Messages*.

Response Error Codes

Table 30: <updateSubscriberNaiResponse> Error Codes (SOAP) lists common error codes for the <updateSubscriberNaiResponse> command. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 30: <updateSubscriberNaiResponse> Error Codes (SOAP)

Error Code	Description
SUCCESS	The update request was successfully completed.
NO_UPDATES	All of the changes were already in the database.
NAI_HOST_NOT_FOUND	Host name does not exist.
DEST_NOT_FOUND	Destination name does not exist.
DEST_TYPE_MISMATCH	Destination has a different destination type than the desired destination type.

Examples

Some of the following examples are based upon previous requests. The order of the requests can be important.

These examples show the SDS provisioning request and response contents that are stored within the <soapenv:Body> or <SOAP-ENV:Body> tags. See Start Transaction *Examples* for an example of the whole SOAP request/response text.

Add New NAI Routing Entities

This example creates three new NAI routing entities and sets their destination values to the specified values. This example assumes that the host and destination values already exist.

The result of this request is:

- New NAI routing entities are created.
- All destination values for each routing entity are set to specified values.

Request:

```
<updateSubscriberNaiRequest timeout="10">
  <naiList>
```

```
    <host>tekelec.com</host>
    <user>John.Smith</user>
    <user>Jane.Doe</user>
    <user>Mike.Jones</user>
  </naiList>
  <destinationList>
    <imshss>IMS_HSS_1</imshss>
    <ltehss>LTE_HSS_1</ltehss>
    <aaa>AAA_Texas</aaa>
  </destinationList>
</updateSubscriberNaiRequest>
```

Response:

```
<ns2:sdsResult affected="3" error="0">
</ns2:sdsResult>
```

Update NAI Routing Entities Destinations (success)

This example updates existing NAI routing entities with new destination values.

Note: This request does not update all NAI values that were specified in the previous request.

The result of this request is that the specified NAI routing entities are updated with specified values.

Request:

```
<updateSubscriberNaiRequest timeout="10">
  <naiList>
    <host>tekelec.com</host>
    <user>Jane.Doe</user>
    <user>Mike.Jones</user>
  </naiList>
  <destinationList>
    <ltehss>LTE_HSS_4</ltehss>
    <pcrf>PCRF_Ohio</pcrf>
  </destinationList>
</updateSubscriberNaiRequest>
```

Response:

```
<ns2:sdsResult affected="2" error="0">
</ns2:sdsResult>
```

Update NAI Routing Entities Destinations (failure)

This example fails to update existing NAI routing entities with new destination values because the destination does not exist.

No changes are made to the database because the request failed.

Request:

```
<updateSubscriberNaiRequest timeout="10">
  <naiList>
    <host>tekelec.com</host>
    <user>Jane.Doe</user>
  </naiList>
  <destinationList>
    <ltehss>junk</ltehss>
```

```
  </destinationList>
</updateSubscriberNaiRequest>
```

Response:

```
<ns2:sdsResult description="destination not found"
 affected="0" error="2006">
</ns2:sdsResult>
```

Delete Subscriber NAI

Request

The <deleteSubscriberNaiRequest> message deletes NAI routing entities. Each NAI value is defined as a combination of a NAI host and NAI user value. For example, "John.Smith@tekelec.com" would have "John.Smith" as the NAI user value and "tekelec.com" as the NAI host value. The <deleteSubscriberNaiRequest> removes the NAI user value, but does not affect the NAI host value.

Semantic Rules

- The host name must already exist in the database.
- Between 1 and 10 user names must be specified.

Request Format

The request must be inserted between the <soapenv:Body> and </soapenv:Body> XML tags of a SOAP request message, as shown in *SOAP Request Messages*.

```
<deleteSubscriberNaiRequest [timeout="timeout"]>
    <naiList>
        <host>host</host>
        <user>user</user>
     [
        <user>user</user>
              …
        <user>user</user>
    ]
    </naiList>
</deleteSubscriberNaiRequest>
```

Request Parameters

Table 31: `<deleteSubscriberNaiRequest>` Parameters (SOAP)

Parameter	Description	Values
timeout (Optional)	The amount of time (in seconds) to wait before being able to perform a write if another connection is performing a write, or has a transaction open. Clients waiting to write will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	0 (return immediately if not available) to 3600 seconds (default is 0).
host	The host name, which is used with all user values.	A string with 1 to 64 characters.
user	The NAI user name to be associated with the host to form an NAI.	A string with 1 to 64 characters. Must have 1-10 user values.

Response

The delete subscriber NAI response is returned as a generic `<ns2:sdsResult>` response. This response returns the result of the request to delete NAI subscriber routing entities. A single result applies to all routing entities supplied. The response returns the number actually deleted. Any that do not exist are not included in the count. However, if any actual delete fails, then the whole command fails and no changes are made.

Note: If applying all of the delete requests results in no database records being deleted (because they already did not exist in the database), the NO_UPDATES error code is returned and the number of affected records is 0.

Response Output

The response is displayed between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP response message, as shown in *SOAP Response Messages*.

```
<ns2:sdsResult affected="affected" error="error" [description="description"]>
</ns2:sdsResult>
```

Response Parameters

The parameters for all of the SOAP response commands are shown in *SOAP Response Messages*.

Error Codes

Table 32: <deleteSubscriberNaiResponse> Error Codes (SOAP) lists the common error codes for the SOAP `<deleteSubscriberNaiResponse>` message. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 32: `<deleteSubscriberNaiResponse>` Error Codes (SOAP)

Error Code	Description
SUCCESS	The delete request was successfully completed.
NO_UPDATES	The records were already deleted from the database.
NAI_HOST_NOT_FOUND	The Host name does not exist.

Examples

These examples show the SDS provisioning request and response contents that are stored within the `<soapenv:Body>` or `<SOAP-ENV:Body>` tags. See Start Transaction *Examples* for an example of the whole SOAP request/response text.

Delete NAI Routing Entities

This example successfully deletes three NAI routing entities.

Request:

```
<deleteSubscriberNaiRequest timeout="10">
  <naiList>
    <host>tekelec.com</host>
    <user>John.Smith</user>
    <user>Jane.Doe</user>
    <user>Mike.Jones</user>
  </naiList>
</deleteSubscriberNaiRequest>
```

Response:

```
<ns2:sdsResult affected="3" error="0">
</ns2:sdsResult>
```

Delete Several NAI Routing Entities

This example successfully deletes two NAI routing entities. Other NAI values were not found and were not deleted.

Request:

```
<deleteSubscriberNaiRequest timeout="10">
  <naiList>
    <host>tekelec.com</host>
    <user>John.Smith</user>
    <user>Ann.Jones</user>
    <user>Jane.Doe</user>
```

```
    <user>Mike.Jackson</user>
  </naiList>
</deleteSubscriberNaiRequest>
```

Response:

```
<ns2:sdsResult affected="2" error="0">
</ns2:sdsResult>
```

Delete NAI Routing Entities (failure)

This example fails because no NAI subscribers are found.

Request:

```
<deleteSubscriberNaiRequest>
  <naiList>
    <host>junk.com</host>
    <user>John.Smith</user>
    <user>Jane.Doe</user>
  </naiList>
</deleteSubscriberNaiRequest>
```

Response:

```
<ns3:deleteSubscriberNaiResponse>
  <result description=”host not found” affected="0" error="2010">
  </result>
</ns3:deleteSubscriberNaiResponse>
```

Read Subscriber NAI

Request

The `<readSubscriberNaiRequest>` message extracts (reads) NAI routing entities and displays the 1-8 destination values for each routing entity.

Semantic Rules

- The host name must already exist in the database.
- Between 1 and 10 user names must be specified.

Request Format

The request must be inserted between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP request message, as shown in *SOAP Request Messages*.

```
<readSubscriberNaiRequest [timeout="timeout"]>
    <naiList>
        <host>host</host>
        <user>user</user>
```

```
     [
        <user>user</user>
              …
        <user>user</user>
     ]
    </naiList>
</readSubscriberNaiRequest>
```

Request Parameters

Table 33: `<readSubscriberNaiRequest>` Parameters (SOAP)

Parameters	Description	Values
timeout (Optional)	The amount of time (in seconds) to wait before being able to perform a write if another connection is performing a write, or has a transaction open. Clients waiting to write will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	0 (return immediately if not available) to 3600 seconds (default is 0).
host	The host name, which is used with all user values.	A string with 1 to 64 characters.
user	The NAI user name to be associated with the host to form an NAI.	A string with 1 to 64 characters. Must have 1-10 user values.

Response

The `<readSubscriberNaiResponse>` response returns the result of the request to read NAI subscriber routing entities. Only those NAI subscriber routing entities that are found are returned. The response message contains up to eight destinations (one for each destination type, such as `<ltehss>`) for each routing entity. Only provisioned destination names are displayed. (i.e. destination names= *"none"* are not displayed).

Response Format

The response is displayed between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP response message, as shown in *SOAP Response Messages*.

```
<ns3:readSubscriberNaiResponse>
    <result affected="affected" error="error" [description="description"]>
    </result>
 [
```

```
    <resultSet>
        <user="user" nai host="host">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </nai>
    [
             ...
        <user="user" nai host="host">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </nai>
    ]
    </resultSet>
]
</ns3:readSubscriberNaiResponse>
```

Response Parameters

Table 34: `<readSubscriberNaiResponse>` Parameters (SOAP)

Parameter	Description	Values
error	Error code that indicates whether or not operation was successfully executed.	0 for success, non-zero for failure.
affected	The number of routing entities read.	0-10
description (Optional)	A textual description associated with the response. This field may contain more information as to why a request failed or a description of the changes if the request succeeded.	A string with 1 to 1024 characters.
<resultSet> SOAP tag (optional)	Indicates rows of data are returned. If no records are being returned, this tag is not present.	A string with 1 to 1024 characters.
host	The host name, which is used with all user values.	A string with 1 to 64 characters.
user	The NAI user name to be associated with the host to form an NAI.	A string with 1 to 64 characters. Must have 1-10 user values.

Parameter	Description	Values
imshss (Optional)	The name of the IMS HSS destination.	A string with 1 to 32 characters.
ltehss (Optional)	The name of the LTE HSS destination.	A string with 1 to 32 characters.
pcrf (Optional)	The name of the PCRF destination.	A string with 1 to 32 characters.
ocs (Optional)	The name of the OCS destination.	A string with 1 to 32 characters.
ofcs (Optional)	The name of the OFCS destination.	A string with 1 to 32 characters.
aaa (Optional)	The name of the AAA server destination.	A string with 1 to 32 characters.
userdef1 (Optional)	The name of the first user defined destination.	A string with 1 to 32 characters.
userdef2 (Optionnal)	The name of the second user defined destination.	A string with 1 to 32 characters.

Error Codes

Table 35: <readSubscriberNaiResponse> Error Codes (SOAP) lists the common error codes for the `<readSubscriberNaiResponse>` command. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 35: `<readSubscriberNaiResponse>` Error Codes (SOAP)

Error Code	Description
SUCCESS	The read request was successfully completed.
NAI_HOST_NOT_FOUND	Host name does not exist.
NAI__NOT_FOUND	None of the specified NAIs exist.

Examples

These examples show the SDS provisioning request and response contents that are stored within the `<soapenv:Body>` or `<SOAP-ENV:Body>` tags. See Start Transaction *Examples* for an example of the whole SOAP request/response text.

Read NAI Routing Entities

This example successfully reads three NAI routing entities.

Request:

```
<readSubscriberNaiRequest>
  <naiList>
    <host>tekelec.com</host>
    <user>John.Smith</user>
    <user>Jane.Doe</user>
    <user>Mike.Jones</user>
  </naiList>
</readSubscriberNaiRequest>
```

Response:

```
<ns3:readSubscriberNaiResponse>
  <result affected="3" error="0">
  </result>
  <resultSet>
    <nai host=”tekelec.com” user=”John.Smith”>
      <imshss>IMS_HSS_1</imshss>
      <ltehss>LTE_HSS_1</ltehss>
      <aaa>AAA_Texas</aaa>
    </nai>
    <nai host=”tekelec.com” user=”Jane.Doe”>
      <imshss>IMS_HSS_1</imshss>
      <ltehss>LTE_HSS_4</ltehss>
      <pcrf>PCRF_Ohio</pcrf>
      <aaa>AAA_Texas</aaa>
    </nai>
    <nai host=”tekelec.com” user=”Mike.Jones”>
      <imshss>IMS_HSS_1</imshss>
      <ltehss>LTE_HSS_4</ltehss>
      <pcrf>PCRF_Ohio</pcrf>
      <aaa>AAA_Texas</aaa>
    </nai>
  </resultSet>
</ns3:readSubscriberNaiResponse>
```

Read NAI Routing Entities

This example successfully reads two NAI routing entities. Other NAI values are not found.

Request:

```
<readSubscriberNaiRequest>
  <naiList>
    <host>tekelec.com</host>
    <user>John.Smith</user>
    <user>Ann.Jones</user>
    <user>Jane.Doe</user>
    <user>Mike.Jackson</user>
  </naiList>
</readSubscriberNaiRequest>
```

Response:

```
<ns3:readSubscriberNaiResponse>
  <result affected="2" error="0">
  </result>
  <resultSet>
    <nai host=”tekelec.com” user=”John.Smith”>
      <imshss>IMS_HSS_1</imshss>
```

```
      <ltehss>LTE_HSS_1</ltehss>
      <aaa>AAA_Texas</aaa>
    </nai>
    <nai host=”tekelec.com” user=”Jane.Doe”>
      <imshss>IMS_HSS_1</imshss>
      <ltehss>LTE_HSS_4</ltehss>
      <pcrf>PCRF_OHIO</pcrf>
      <aaa>AAA_Texas</aaa>
    </nai>
  </resultSet>
</ns3:readSubscriberNaiResponse>
```

Read NAI Routing Entities (failure)

This example fails because no NAI subscribers are found.

Request:

```
<readSubscriberNaiRequest>
  <naiList>
    <host>tekelec.com</host>
    <user>Kevin.Smith</user>
    <user>John.Doe</user>
  </naiList>
</readSubscriberNaiRequest>
```

Response:

```
<ns3:readSubscriberNaiResponse>
  <result description=”nai not found” affected="0" error="2009">
  </result>
</ns3:readSubscriberNaiResponse>
```

Message Flow Example Sessions

The following sections contain example usages of the exchanging messages between the Customer Provisioning System (CPS) and the XDS process on the Active SDS Server on the Primary Provisioning Site. All scenarios assume that a TCP/IP connection has already been established between the client and SDS.

The examples only show the text that is between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP request message.

The first column in the tables is the direction that the message is going. The strings displayed in the Message column are the actual ASCII text that is between the `<soapenv:Body>` and `</soapenv:Body>` XML tags of a SOAP request that would flow over the connection.

The actual request and response messages are just a series of characters with no extra spaces or new line characters. New lines and extra spaces were added to the examples for readability purposes.

Single Command Transaction

This example shows three request/response pairs that are exchanged between the CPS and SDS. These requests are processed as single command transactions, which means that each request is immediately committed to the database. This example creates IMSI and MSISDN routing entities.

Table 36: Single Command Transaction Message Flow Example SOAP)

Message		Description
CPS—>SDS	<updateSubscriberRequest> <addressList> <imsi>310910421000106</imsi> <imsi>310910421000307</imsi> <imsi>310910421000309</imsi> <msisdn>15634210106</msisdn> <msisdn>15634210107</msisdn> </addressList> <destinationList> <ltehss>LTE_HSS_2</ltehss> <aaa>AAA_4</aaa> </destinationList> </updateSubscriberRequest>	Request to create 5 stand-alone routing entities - 3 IMSIs and 2 MSISDNs with an LTE HSS and AAA server destinations. **Note:** Request is made to include the original request in the response. Response to create stand-alone routing entities - success. Affected rows = 5 (as 5 new entries created for 3 IMSIs and 2 MSISDNs).
CPS<—SDS	<ns2:sdsResult affected="5" error="0"> </ns2:sdsResult>	
CPS—>SDS	<updateSubscriberRequest> <addressList> <imsi>310910421000106</imsi> <msisdn>15634210106</msisdn> </addressList> <destinationList> <ltehss>LTE_HSS_5</ltehss> </destinationList> </updateSubscriberRequest>	Request to update existing IMSI and MSISDN subscriber routing entities with a new LTE HSS value. Response to update subscriber routing entities - success. Affected rows = 2 (2 entries for an IMSI and MSISDN were updated with new LTE HSS value).
CPS<—SDS	<ns2:sdsResult affected="2" error="0"> </ns2:sdsResult>	
CPS—>SDS	<updateSubscriberRequest> <addressList> <imsi>310910421000102</imsi> </addressList> <destinationList> <ltehss>BAD_VALUE</ltehss> </destinationList> </updateSubscriberRequest>	Request to create a stand-alone routing entitiy with an invalid LTE HSS destination value. Request fails, as the destination does not exist.
CPS<—SDS	<ns2:sdsResult description=“destination not found” affected="0" error="2006"> </ns2:sdsResult>	

Multiple Commands Transaction Committed

This example issues several requests within one transaction which is then committed successfully.

Table 37: Multiple Commands Transaction Committed Message Flow Example (SOAP)

Message		Description
CPS—>SDS	`<startTransactionRequest>0</startTransactionRequest>`	Request to start a transaction immediately.
CPS<—SDS	`<ns2:sdsResult affected="0" error="0"> </ns2:sdsResult>`	Response to start transaction - success.
CPS—>SDS	`<updateSubscriberRequest <addressList> <imsi>310910421000444</imsi> <msisdn>15634210444</msisdn> </addressList> <destinationList> <ltehss>LTE_HSS_1</ltehss> </destinationList> </updateSubscriberRequest>`	Request to add new stand-alone IMSI and MSISDN - success.
CPS<—SDS	`<ns2:sdsResult affected="2" error="0"/> </ns2:sdsResult>`	
CPS—>SDS	`<updateSubscriberRequest> <addressList> <imsi>310910421000555</imsi> <msisdn>15634210555</msisdn> </addressList> <destinationList> <ltehss>LTE_HSS_2</ltehss> </destinationList> </updateSubscriberRequest>`	Request to update existing stand-alone IMSI and MSISDN - success.
CPS<—SDS	`<ns2:sdsResult affected="2" error="0"/> </ns2:sdsResult>`	
CPS—>SDS	`<updateSubscriberNaiRequest> <naiList> <host>operator.com</host> <user>roger.brown</user> </naiList> <destinationList> <ltehss>LTE_HSS_1</ltehss> </destinationList> </updateSubscriberNaiRequest>`	Request to update an NAI - success.
CPS<—SDS	`<ns2:sdsResult affected="1" error="0"> </ns2:sdsResult>`	

Message		Description
CPS—>SDS	`<commitRequest> </commitRequest>`	Request to commit the transaction. Response to commit transaction - success. All updates were successfully performed.
CPS<—SDS	`<ns2:sdsResult affected="0" error="0"> </ns2:sdsResult>`	

Multiple Commands Transaction Rolled Back

This example issues several requests within one transaction which is rolled back.

Table 38: Multiple Commands Transaction Rolled Back Message Flow Example (SOAP)

Message		Description
CPS—>SDS	`<startTransactionRequest>10</startTransactionRequest>`	Request to start a transaction within 10 seconds.
CPS<—SDS	`<ns2:sdsResult affected="0" error="0"> </ns2:sdsResult>`	Response to start transaction - success.
CPS—>SDS	`<updateSubscriberRequest> <addressList> <imsi>310910421000777</imsi> <msisdn>15634210777</msisdn> </addressList> <destinationList> <ltehss>LTE_HSS_7</ltehss> </destinationList> </updateSubscriberRequest>`	Request to update existing stand-alone IMSI and MSISDN - success.
CPS<—SDS	`<ns2:sdsResult affected="2" error="0"/> </ns2:sdsResult>`	
CPS—>SDS	`<updateSubscriberNaiRequest> <naiList> <host>operator.com</host> <user>david.leno</user> </naiList> <destinationList> <ltehss>LTE_HSS_1</ltehss> </destinationList> </updateSubscriberNaiRequest>`	Request to create an NAI - success.
CPS<—SDS	`<ns2:sdsResult affected="1" error="0"> </ns2:sdsResult>`	
CPS—>SDS	`<rollbackRequest> </rollback>`	Transaction is rolled back by the client. None of the previous IMSI, MSISDN or NAI entities will be created. Rollback is successful; no creations/updates are made. The client could have sent a commit instead of the rollback, which would have resulted in the 2 IMSIs, 2 MSISDNs, and 1 NAI being created.
CPS<—SDS	`<ns2:sdsResult affected="0" error="0"> </ns2:sdsResult>`	

Chapter 5

XML Message Definitions

Topics:

This chapter describes XML requests and responses syntax and parameters.

Message Conventions

Message specification syntax follows several conventions to convey what parameters are required or optional and how they and their values must be specified.

Table 39: Message Conventions

Symbol	Description
`monospace with background`	All code examples.
`monospace`	Names of commands when provided outside of a code example.
italics	Variable names when provided outside of a code example or value list.
spaces	Spaces (ie, zero or more space characters, " ") may be inserted anywhere except within a single name or number. At least one space is required to separate adjacent names or numbers.
…	Ellipses represent a variable number of repeated entries. For example: `dn DN1 , dn DN2, …, dn DN7, dn DN8`

< >	Angle brackets are used to enclose parameter values that are choices or names.
	In the following example, the numbers represent specific value choices.
	`parameter1 <1|2|3>`
	In the following example, `ServerName` represents the actual value.
	`parameter2 <ServerName>`
	In the following example, the numbers represent a choice in the range from 0 to 3600.
	`parameter3 <0..3600>`

[ ]	Square brackets are used to enclose an optional parameter and its value. `[, parameter1 < 1|2|3 >]`

Symbol	Description
	A parameter and its value that are not enclosed in square brackets are mandatory.
|	The pipe symbol is used in a parameter value list to indicate a choice between available values. `Parameter1 <1|2|3>`
,	A literal comma is used in the message to separate each parameter that is specified.

XML-based Interface

The XML Data Server uses an XML based protocol, in which a client communicates with the XML Data Server by issuing request message strings over an underlying TCP/IP network connection. A session consists of a series of XML commands, initiated by the client, and responses from the XML Data Server.

Every XML request/response consists of a 4-byte binary length value, followed by the indicated number of ASCII characters that form the XML request. There is no need to terminate the XML request with any terminating character(s).

The length value is a 4 byte integer in network byte order indicating the size in bytes of the XML part.

Note: "Network byte order" refers to the standard byte order defined in the IP protocol. It corresponds to big-endian (most significant first). It is a zero-padded 4 byte value.

The following data-stream Hex dump provides an example of an update subscriber request sent from an XML Data Server client to the XML Data Server.

```
00000000  00 00 00 8d 3c 75 70 64  61 74 65 53 75 62 73 63  ....<updateSubsc
00000010  72 69 62 65 72 20 65 6e  74 3d 22 73 75 62 73 63  riber ent="subsc
00000020  72 69 62 65 72 52 6f 75  74 69 6e 67 22 20 6e 73  riberRouting" ns
00000030  3d 22 64 73 72 22 3e 3c  69 6d 73 69 3e 33 31 30  ="dsr"><imsi>310
00000040  39 31 30 34 32 31 30 30  30 30 31 30 33 3c 2f 69  9104210000103</i
00000050  6d 73 69 3e 3c 6c 74 65  68 73 73 3e 4c 54 45 5f  msi><ltehss>LTE_
00000060  48 53 53 5f 32 3c 2f 6c  74 65 68 73 73 3e 3c 61  HSS_2</ltehss><a
00000070  61 61 3e 41 41 41 5f 34  3c 2f 61 61 61 3e 3c 2f  aa>AAA_4</aaa></
00000080  75 70 64 61 74 65 53 75  62 73 63 72 69 62 65 72  updateSubscriber
00000090  3e
```

Like the XML request message, an XML response message consists of a 4 byte binary length value, followed by the indicated number of ASCII characters that form the XML response. There is no terminator to the XML response.

The following data-stream Hex dump provides an example of an update subscriber response message string sent from an XML Data Server client to the XML Data Server client.

```
00000000  00 00 00 4a 3c 75 70 64  61 74 65 53 75 62 73 63  ....<updateSubsc
00000010  72 69 62 65 72 52 65 73  70 3e 3c 72 65 73 20 65  riberResp><res e
00000020  72 72 6f 72 3d 22 30 22  20 61 66 66 65 63 74 65  rror="0" affecte
00000030  64 3d 22 31 22 2f 3e 3c  2f 75 70 64 61 74 65 53  d="1"/></updateS
00000040  75 62 73 63 72 69 62 65  72 52 65 73 70 3e        ubscriberResp>
```

Transaction Id (ID)

Each message can have a Transaction Id called the id as an attribute. The id attribute is used by the XML Data Server client to correlate request and response messages. The id attribute is optional and if specified, is an integer between 1 and 4294967295, expressed as a decimal number in ASCII. If the id attribute is specified in a request, the same id attribute and value are returned by the XML Data Server in the corresponding response. A unique id value must be used in each request message to differentiate responses.

XML Response Messages

An XML response message is sent by the SDS XML provisioning client in response to an XML request.

Each response message consists of a 4-byte binary length value, followed by the XML response in ASCII characters. The length value contains the number of bytes in the XML response, excluding the 4-bytes for the length.

The original XML request is included in the response only if indicated in the initiating request.

A rowset, contained between the `<rset>` tags, is only present if data is to be returned, such as in the `<readSubscriber>` and `<readSubscriberNai>` requests.

A generic response type can be generated if the XML request cannot be parsed, the request is not valid, and in some other cases. The response name of a generic response type is `errorResp`. The `id` field, if supplied in the original request, may be included if was possible to extract it, but this cannot be guaranteed, depending on the error condition.

Response Message Format (`<readSubscriberResp>` and `<readSubscriberNaiResp>` messages)

```
lengthInBytes
<respName [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
[
    <rset>
        <rowName [ [rowAttributeName]="rowAttributeValue"] …
                   [rowAttributeName]="rowAttributeValue"] ]>
            <rowValueName>rowValue</rowValueName>
            ...
            <rowValueName>rowValue</rowValueName>
        </rowName>
        ...
        <rowName [ [rowAttributeName]="rowAttributeValue"] …
                   [rowAttributeName]="rowAttributeValue"] ]>
            <rowValueName>rowValue</rowValueName>
            ...
            <rowValueName>rowValue</rowValueName>
        </rowName>
    </rset>
]
</respName>
```

Response Message Format (all other requests)

```
lengthInBytes
<respName [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
</respName>
```

Response Message Parameters

Table 40: Response Message Parameters (XML)

Parameter	Description	Values
lengthInBytes	Number of bytes following to form XML request. This is a 4-byte binary value.	0-4294967295
respName	The name of the response based on the original XML request sent.	A string with 1 to 64 characters. The value is the request name appended with `Resp`. E.g. for the `<updateSubscriber>` request, the response name is `updateSubscriberResp`. If the request name is invalid, or the XML cannot be parsed, the response name is `errorResp`.
id (Optional)	Transaction ID value provided in the request and passed back in the response.	1-4294967295
originalXMLRequest (Optional)	The text of the original XML request that was sent. This parameter is present only if the `resonly=n` attribute is set in the original request.	A string with 1 to 4096 characters.
error	Whether or not operation was successfully executed by the XML Data Server.	0 - success; non zero - failure.
affected	The number of routing entities (or subscribers if `group="y"`) created/updated.	0-10
description (Optional)	A textual description associated with the response. This may contain more information as to why a request failed or describe the changes if it succeeds.	A string with 1 to 1024 characters.

Parameter	Description	Values
rowName	The name of the row type returned.	This value is dependant on the result set returned.
rowValue	The value of the row type returned.	This value is dependant on the result set returned.
rowAttributeName	The name of the row attribute name returned.	This value is dependant on the result set returned.
rowAttributeValue	The value of the row attribute name returned.	This value is dependant on the result set returned.

Update and Delete Subscriber Command

If the XML command successfully updates or deletes a subscriber, the response description text indicates the deleted/created/changed IMSI and/or MSISDN values and a list of the subscriber destination values.

Note: Destination values are listed only if there were created or modified IMSI and/or MSISDN routing entities.

Description text format

```
[description="[deleted ({imsi nnnn|dn nnnn}[, imsi nnnn|,dn nnnn]…)]
[, created ({imsi nnnn|dn nnnn}[, imsi nnnn|,dn nnnn]…)]
[, changed ({imsi nnnn|dn nnnn}[, imsi nnnn|,dn nnnn]…)]
[, imshss nnnn][, ltehss nnnn][, pcrf nnnn][, ocs nnnn]
[, ofcs nnnn][, aaa nnnn][, userdef1 nnnn][, userdef2 nnnn]"]
```

<updateSubscriber> description text example

```
description="deleted (imsi 444444444444440, dn 19195550000), created
(imsi 444444444444441, dn 19195550001, dn 19195550002), imshss imshss2, ltehss
ltehss1"
```

Supported Request Messages

Table 41: Supported XML Data Server Requests lists the requests supported by the XML Data Server. Unsupported operations/requests are rejected with an INV_REQUEST_NAME error code. XML Data Server clients are to construct requests as specified in the sections referenced in *Table 41: Supported XML Data Server Requests*.

Table 41: Supported XML Data Server Requests

Request	Description	Section
startTransaction	Start Database Transaction	*Start Transaction*
commit	Commit Database Transaction	*Commit Transaction*

Request	Description	Section
rollback	Abort Database Transaction	*Rollback Transaction*
tx	Block Transaction	*Block Transactions*
updateSubscriber	Create/Update IMSI/MSISDN Routing	*Update Subscriber*
deleteSubscriber	Delete IMSI/MSISDN Routing	*Delete Subscriber*
readSubscriber	Get IMSI/MSISDN Routing	*Read Subscriber*
updateSubscriberNai	Create/Update NAI Routing	*Update Subscriber NAI*
deleteSubscriberNai	Delete NAI Routing	*Delete Subscriber NAI*
readSubscriberNai	Get NAI Routing	*Read Subscriber NAI*

Start Transaction

Request

The <startTransaction> request begins a database transaction.

Data manipulation and query requests (update, delete, and read) can be sent within the context of a transaction. A client connection can only have one transaction open at a time.

Data manipulation requests are evaluated for validity and applied to a local database view, which is a virtual representation of the main database plus local modifications made within this active transaction. Local database view changes are not committed to the main database until the transaction is ended with a `<commit>` request.

If a `<startTransaction>` request is sent, and then the connection is lost or the user logs off without sending a `<commit>` or `<rollback>` request, all pending requests are rolled back.

A provisioning session can have one transaction open at a time. If a `<startTransaction>` request is sent, another `<startTransaction>` request will fail with an ACTIVE_TXN error.

A timeout occurs between the `<startTransaction>` and `<commit>` requests. If the `<commit>` request is not sent out within the configured "Maximum Transaction Lifetime" on the SDS GUI (see the *SDS Online Help* for more information) of the `<startTransaction>` request, the XML provisioning requests are rolled back (changes not applied to database).

A transaction can only be opened by one client at a time. If a transaction is already opened by another client, the `<startTransaction>` request is rejected immediately with WRITE_UNAVAIL or is queued up for the time specified by the timeout parameter. If the timeout parameter is specified with a non-zero value and that period of time elapses before the transaction is opened, the `<startTransaction>` request is rejected with WRITE_UNAVAIL.

Data manipulation requests are evaluated for validity and applied to a local database view which is a virtual representation of the main database plus local modifications made within this active transaction.

Local database view changes are not committed to the main database until the transaction is ended with a `<commit>` request.

The request can be aborted and rolled back with a `<rollback>` request any time before the transaction is ended with a `<commit>` request.

A block transaction (`<tx> ... </tx>`) is not allowed with a normal transaction, and will result in an INV_REQ_IN_NORMAL_TX error being returned for that request.

Request Format

```
<startTransaction [resonly="resonly"] [id="id"] [timeout="timeout"]/>
```

Parameters

Table 42: `<startTransaction>` Parameters (XML)

Parameter	Description	Values
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction ID value provided in request, and will be passed back in the response.	1-4294967295
timeout (Optional)	The amount of time (in seconds) to wait to open a transaction if another connection already has one open. Clients waiting to open a transaction will be processed in the order that the requests were received.	0 (return immediately if not available) to 3600 seconds (default is 0).

Response

The `<startTransactionResp>` response returns the result of starting a database transaction. If the response error code indicates success, then the database transaction was successfully started. If any failure response is returned, then the database transaction was not started.

Response Format

```
lengthInBytes
<startTransactionResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
</startTransactionResp>
```

Response Parameters

The parameters for all of the response commands are shown in *XML Response Messages*.

Error Codes

Table 43: <startTransactionResp> Error Codes (XML) shows the common error codes for the <startTransactionResponse>. See *SDS Response Message Error Codes* for a full list of error codes.

Table 43: <startTransactionResp> Error Codes (XML)

Error Code	Description
SUCCESS	Transaction was successfully started.
NO_WRITE_PERMISSION	The client making the connection does not have write access permissions.
WRITE_UNAVAILABLE	Another client already has a transaction open. This will only be returned to clients who do have write access permissions.
ACTIVE_TXN	A transaction is already open on this connection.

Examples

Start a Transaction Within 2 Minutes (success)

This example successfully starts a transaction within 2 minutes.

Request:

```
<startTransaction id="101" timeout="120"></startTransaction>
```

Response:

```
<startTransactionResp id="101">
  <res error="0" affected="0"/>
</startTransactionResp>
```

Start a Transaction Immediately (fail)

This example attempts to immediately start a transaction but fails due to another client having a transaction open.

Request:

```
<startTransaction resonly="n" id="102"></startTransaction>
```

Response:

```
<startTransactionResp id="102">
  <startTransaction resonly="n" id="102"></startTransaction>
  <res error="1005" affected="0"/>
</startTransactionResp>
```

Commit Transaction

Request

The <commit> request commits an active database transaction.

If the currently opened transaction has one or more successful updates, then committing the transaction will cause all the database changes to be committed. All previous updates, even though they received a successful error code, are not committed to the database until the <commit> request is received.

Request Format

```
<commit [resonly="resonly"] [id="id"]/>
```

Parameters

Table 44: <commit> Request Parameters (XML)

Parameter	Description	Values
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction ID value provided in the request and passed back in the response.	1-4294967295

Response

The <commitResp> response returns the results of committing a database transaction. If the response error code indicates success, then the update was successfully committed in the database. If any failure response is returned, then the database commit failed. The <commit> request causes the transaction to end regardless of whether any updates were actually made to the database.

Note: The affected row count in the XML response will always be 0. It does not indicate how many rows were modified within the transaction.

Response Format

```
lengthInBytes
<commitResp [id="id"]>
[
    originalXMLRequest
]
```

```
    <res error="error" affected="affected" [description="description"]/>
</commitResp>
```

Response Parameters

The parameters for all of the response commands are shown in *XML Response Messages*.

Error Codes

Table 45: <commitResp> Error Codes (XML) lists the common error codes for the `<commitResp>` response. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 45: `<commitResp>` Error Codes (XML)

Return Code	Description
SUCCESS	Database transaction was committed successfully.
NO_ACTIVE_TXN	A transaction is not currently open on this connection.

Examples

Commit a Transaction (success)

This example successfully commits a transaction.

Request:

```
<commit id="101"></commit>
```

Response:

```
<commitResp id="101">
  <res error="0" affected="0"/>
</commitResp>
```

Commit a Transaction that is not Open (fail)

This example attempts to commit a transaction but fails because a transaction was not open.

Request:

```
<commit resonly="n" id="102"></commit>
```

Response:

```
<commitResp id="102">
  <commit resonly="n" id="102"></commit>
  <res error="1009" affected="0"/>
</commitResp>
```

Rollback Transaction

Request

The `<rollback>` request aborts the currently active database transaction. Any updates are rolled back prior to closing the transaction.

Request Format

```
<rollback [resonly="resonly"] [id="id"]/>
```

Request Parameters

Table 46: `<rollback>` Parameters (XML)

Parameter	Description	Values
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result. Do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction ID value provided in request, and passed back in the response.	1-4294967295

Response

The `<rollbackResp>` response returns the results of aborting a database transaction.

Response Format

```
lengthInBytes
<rollbackResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
</rollbackResp>
```

Parameters

The parameters for all of the XML response commands are shown in *XML Response Messages*.

Error Codes

Table 47: <rollbackResp> Error Codes (XML) lists the common error codes for the `<rollbackResp>` command. See *SDS Response Message Error Codes* for a full list of error codes.

Table 47: `<rollbackResp>` Error Codes (XML)

Return Code	Description
SUCCESS	Database transaction was aborted successfully.
NO_ACTIVE_TXN	A transaction is not currently open on this connection.

Examples

Rollback a Transaction (success)

This example successfully rolls back a transaction.

Request:

```
<rollback resonly="n" id="101"></rollback>
```

Response:

```
<rollbackResp id="101">
  <rollback resonly="n" id="101"></rollback>
  <res error="0" affected="0"/>
</rollbackResp>
```

Rollback a Transaction that is not Open (fail)

This example attempts to rollback a transaction but fails because a transaction was not open.

Request:

```
<commit resonly="n" id="102"></rollback>
```

Response:

```
<rollbackResp id="102">
  <rollback resonly="n" id="102"></rollback>
  <res error="1009" affected="0"/>
</rollbackResp>
```

Block Transactions

A block transaction allows the user to group a number of requests within a transaction and send them as a single unit of data. Requests are executed when the whole unit has been sent.

The data unit consists of the block transaction tags, with a number of requests contained within the tags.

It is possible to select if the result to each request is included in the response, by use of the resonly attribute. The selection, even the default when not included, is applied to every request within the block transaction. If an individual request sets the resonly attribute, the attribute is overridden with the value from the block transaction.

The following requests are not permitted within a block transaction, and will result in a INV_REQ_IN_BLOCK_TX error being returned:

- `<startTransaction>`
- `<commit>`
- `<rollback>`

Request

Request Format

```
<tx [resonly="resonly"] [id="id"] [timeout="timeout"]>
[
    <requestName ...>
    …
    </requestName>

    …

    <requestName ...>
    …
    </requestName>
]
</tx>
```

Request Parameters

Table 48: `<tx>` Request Parameters

Parameter	Description	Values
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction ID value provided in the request and passed back in the response.	1-4294967295
timeout (Optional)	The amount of time (in seconds) to wait to open a transaction if another connection already has one open. Clients waiting to open a transaction will be	0 (return immediately if not available) to 3600 seconds. The default is 0.

Parameter	Description	Values
	processed in the order that their requests were received.	
requestName (Optional)	Contains 0-50 occurrences of the following XML requests: <updateSubscriber>, <deleteSubscriber>, <readSubscriber>, <updateSubscriberNai>, <deleteSubscriberNai>, <readSubscriberNai>	

Response

The <txResp> response returns the number or requests within the transaction, and a response message for each request.

If an error occurred performing one request, then all requests within the transaction, up to and including the failed request will automatically be rolled back. If all requests are successful, then all requests within the transaction are automatically committed.

Response Format

```
lengthInBytes
<txResp nbreq="nbreq" [id="id"]>
[
    <requestResp ...>
        …
        <res error=...>
    </requestResp>

    …

    <requestResp ...>
        …
        <res error=...>
    </requestResp>
]
</txResp>
```

Response Parameters

Table 49: <txResp> Parameters

Parameter	Description	Values
lengthinbytes	Number of bytes following to form XML request. This is a 4 byte binary value.	0-4294967295
nbreq	Number of requests within the transaction. The response will	0-50

Parameter	Description	Values
	contain responses and optionally the requests themselves for each request.	
id (Optional)	Transaction ID value provided in request and passed back in the response.	1-4294967295
requestResp	Contains 0-50 occurrences of the following XML request responses: `<updateSubscriberResp>`, `<deleteSubscriberResp>`, `<readSubscriberResp>`, `<updateSubscriberNaiResp>`, `<deleteSubscriberNaiResp>`, `<readSubscriberNaiResp>`	

Response Error Codes

Table 50: <txResp> Error Codes lists the common error codes for the Block Transaction response. See *SDS Response Message Error Codes* for a full list of error codes.

Table 50: `<txResp>` Error Codes

Error Code	Description
SUCCESS	Database transaction was committed successfully.
ACTIVE_TXN	A transaction is already open on this connection.
TXN_TOO_BIG	Transaction too big (more than the configured maximum number of requests).
DB_EXCEPTION	An unexpected exception was thrown during the database commit. The entire transaction was rolled back to ensure predictable behavior. Contact Tekelec.
NOT_PROCESSED	Not processed. The request was within a block transaction, and was not processed due to an error with another request within the same block transaction.
INV_REQ_IN_BLOCK_TX	An invalid request has been sent in a block transaction (e.g. `<startTransaction>`, `<commit>`, or `<rollback>`).

Examples

Start a Block Transaction Within 2 Minutes (success)

This example successfully starts a block transaction within two minutes and successfully runs requests.

Request:

```
<startTransaction id=”101” timeout=”120”>
  <updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”201”>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
    <ltehss>LTE_HSS_1</ltehss>
  </updateSubscriber>
  <updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”202”>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
    <imshss>none</imshss>
    <ltehss>LTE_HSS_4</ltehss>
    <aaa>AAA_4</aaa>
  </updateSubscriber>
  <deleteSubscriber ent=”subscriberRouting” ns=”dsr” id=”203”>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
  </deleteSubscriber>
</tx>
```

Response:

```
<txResp nbreq=”3” id=”101”>
  <updateSubscriberResp id=”201”>
    <res error=”0” affected=”3”/>
  </updateSubscriberResp>
  <updateSubscriberResp id=”202”>
    <res error=”0” affected=”2”/>
  </updateSubscriberResp>
  <deleteSubscriberResp id=”203”>
    <res error=”0” affected=”2”/>
  </deleteSubscriberResp>
</txResp>
```

Block Transaction Failed (and Rolled Back)

This example attempts to run requests within a block transaction, but the second request fails. All requests are rolled back.

Request:

```
<tx id=”102”>
  <updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”201”>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
    <ltehss>LTE_HSS_1</ltehss>
  </updateSubscriber>
  <updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”202”>
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
```

```
    <imshss>none</imshss>
    <ltehss>BAD_VALUE</ltehss>
    <aaa>AAA_4</aaa>
  </updateSubscriber>
  <deleteSubscriber ent="subscriberRouting" ns="dsr" id="203">
    <imsi>111111111100002</imsi>
    <msisdn>8004605500</msisdn>
  </deleteSubscriber>
</tx>
```

Response:

```
<txResp nbreq="3" id="102">
  <updateSubscriberResp id="201">
    <res error="0" affected="3"/>
  </updateSubscriberResp>
  <updateSubscriberResp id="202">
    <res error="2006" affected="0"/>
  </updateSubscriberResp>
  <deleteSubscriberResp id="203">
    <res error="1" affected="0"/>
  </deleteSubscriberResp>
</txResp>
```

Update Subscriber

Subscriber and Routing Data

A routing entity contains the IMSI or MSISDN value along with up to eight destination names that refer to destination data which contains FQDN and realm values that are used for routing messages.

A subscriber is a group of related IMSI and/or MSISDN routing entities and an optional Account ID value. All routing entities within a subscriber have the same destination values.

A stand-alone routing entity is a routing entity that is not assigned to any subscriber.

Each IMSI or MSISDN routing entity is either a stand-alone routing entity or is assigned to a single subscriber.

Request

The `<updateSubscriber>` request provisions IMSI and MSISDN routing data and can provision subscriber data. The request updates existing routing entities, and creates any new routing entities that do not exist.

The request allows for the provisioning of IMSI, MSISDN, or combinations of IMSI and MSISDNs to be associated with eight different destinations.

A destination name can be specified as *"none"*, which removes the association of that destination from the specified routing entity(s).

Semantic Rules (all requests)

- At least one routing entity (IMSI or MSISDN) must be specified.
- No more than 10 routing entities (IMSI, MSISDN, or combinations of the two) can be specified.
- A destination name must already exist in the database.
- Each destination name type can only be specified once.
- All specified routing entities will be provisioned with the same destination value(s).
- Any existing destination(s) for a routing entity will not be changed/removed if not specified in the request.
- Specifying a destination name of `"none"` removes the association of that destination from the specified routing entity(s).

Semantic Rules (requests that do not specify the group attribute or specify `group="n"`)

- The `accountId`, `deleteAccountId`, `deleteImsi`, and `deleteMsisdn` parameters cannot be specified.
- All specified IMSI and MSISDN values must be for stand-alone routing entities or they all must be assigned to one subscriber. There cannot be a mixture of stand-alone routing entities and routing entities that are part of a subscriber.
- At least one routing entity (IMSI or MSISDN) must be specified.
- A maximum of 10 routing entities (IMSI, MSISDN, or combinations of the two) can be specified.
- At least one destination must be specified.
- All specified routing entities will be provisioned with the same destination value(s).

Semantic (requests that specify attribute `group="y"`)

- The `accountId`, `deleteAccountId`, `deleteImsi`, and `deleteMsisdn` parameters can be specified.
- All specified Account ID, IMSI, or MSISDN values must be assigned to one subscriber or must exist in stand-alone routing entities. After the command successfully completes, all specified values will be assigned to one subscriber.
- All specified `deleteAccountId`, `deleteImsi`, or `deleteMsisdn` values that exist in the database must be assigned to the same subscriber. All Account ID, IMSI or MSISDN values must be assigned to the same subscriber or not assigned to any subscriber.
- At least one IMSI, MSISDN, or Account ID value must be specified.
- The `deleteAccountId`, `deleteImsi`, `deleteMsisdn` values and all destination tags and values are optional. This allows a user to just add an Account ID or MSISDN and/or IMSI values to a subscriber.
- A maximum of 1 `accountId`, 1 `deleteAccountId`, 6 `imsi`, 6 `deleteImsi`, 6 `msisdn`, and/or 6 `deleteMsisdn` values can be specified. If any of these limits are exceeded, the request fails.
- All specified `accountId`, `imsi`, and `msisdn` values that are not currently associated with a subscriber will be assigned to the same subscriber. They are added to an existing subscriber or new subscriber.
- If a new subscriber is being created with all new routing entities, all specified routing entities will be provisioned with the specified destination values.
- If a new subscriber is being created with at least one existing stand-alone routing entity, all destination values from existing stand-alone routing entities must be the same prior to applying any specified destination changes. All new routing entities will inherit their destinations values from an existing stand-alone routing entity with the applied destination changes.

- If existing stand-alone routing entities are being added to an existing subscriber, the destination values in each stand-alone routing entity must match the destination values for the subscriber (extracted from any of the subscriber's routing entities) prior to applying any specified destination changes.
- If new routing entities are being added to an existing subscriber, the new routing entities will inherit the destination values for the subscriber (extracted from any of the subscriber's routing entities).
- If a new routing entity is being created, at least 1 of its destination values must not be equal to `none`.
- The updated subscriber must have at least 1 IMSI or MSISDN routing entity.
- The updated subscriber can have 0 or 1 Account ID values, 0-6 IMSI values and 0-6 MSISDN values, as long as there is at least one IMSI or MSISDN value. If the result of the update (deleting and then adding new Account ID, IMSI, and/or MSISDN values) would cause too many Account ID, IMSI or MSISDN values, the request will fail.
- The subscriber's Account ID value can be updated only if it is currently null or deleted within the request (as specified by the `deleteAccountId` parameter).
- If any of the values specified in the `deleteAccountId`, `deleteImsi`, or `deleteMsisdn` parameters do not exist in the database, they will be ignored. If nothing else changes for the subscriber, the NO_UPDATES is returned.
- If any of the values specified in the `deleteAccountId`, `deleteImsi`, or `deleteMsisdn` parameters exist in the database, they must be assigned to the subscriber being updated or the command will fail.
- If any of the values specified in the `deleteImsi` or `deleteMsisdn` parameters exist, the routing entity will be deleted unless it is the last IMSI or MSISDN routing entity for the subscriber, in which case the command will fail.

Request Format

```
<updateSubscriber ent="subscriberRouting" ns="dsr" [resonly="resonly"]
                  [id="id"] [timeout="timeout"] [group="group"]>
[   <deleteAccountId>deleteAccountId</deleteAccountId> ]
[
    <deleteImsi>deleteImsi</deleteImsi>
    …
    <deleteImsi>deleteImsi</deleteImsi>
]
[
    <deleteMsisdn>deleteMsisdn</deleteMsisdn>
    …
    <deleteMsisdn>deleteMsisdn</deleteMsisdn>
]
[   <accountId>accountId</accountId> ]
[
    <imsi>imsi</imsi>
    …
    <imsi>imsi</imsi>
]
[
    <msisdn>msisdn</msisdn>
    …
    <msisdn>msisdn</msisdn>
]
[   <imshss>imshss</imshss>       ]
[   <ltehss>ltehss</ltehss>       ]
[   <pcrf>pcrf</pcrf>             ]
[   <ocs>ocs</ocs>                ]
```

```
[   <ofcs>ofcs</ofcs>             ]
[   <aaa>aaa</aaa>                ]
[   <userdef1>userdef1</userdef1> ]
[   <userdef2>userdef2</userdef2> ]
</updateSubscriber>
```

Request Parameters

Table 51: `<updateSubscriber>` Request Parameters (XML)

Parameter	Description	Values
ent	The entity name within the global schema.	subscriberRouting
ns	The namespace within the global schema.	dsr
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction id value provided in request and passed back in the response.	1-4294967295
timeout (Optional)	The amount of time (in seconds) to wait to before being able to perform a write if another connection is performing a write, or has a transaction open. Clients waiting to write are processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	0 (return immediately if not available) to 3600 seconds (default is 0).
group (Optional)	Indicates if relationships between a group of related IMSI and/or MSISDN routing entities and Account ID value should be created/updated.	• y - Create new or update existing subscriber relationships and update destinations. • n - Only update destinations, not relationships between routing entities (default).
deleteAccountId (Optional)	A user-defined Account ID value to delete.	1 to 26 decimal digits.

Parameter	Description	Values
deleteImsi (Optional)	An IMSI (specified in E.212 format) value to delete.	10 to 15 numeric digits.
deleteMsisdn (Optional)	An MSISDN (specified in E.164 international public telecommunication numbering plan format) value to delete.	8 to 15 decimal digits.
accountId (Optional)	A user-defined Account ID value to add or update.	1 to 26 numeric digits.
imsi (Optional)	An IMSI (specified in E.212 format) to add or update.	10 to 15 numeric digits.
msisdn (Optional)	An MSISDN (specified in E.164 international public telecommunication numbering plan format) to add or update.	8 to 15 numeric digits.
imshss (Optional)	The name of the IMS HSS destination.	A string with 1 to 32 characters.
ltehss (Optional)	The name of the LTE HSS.	A string with 1 to 32 characters.
pcrf (Optional)	The name of the PCRF destination.	A string with 1 to 32 characters.
ocs (Optional)	The name of the OCS destination.	A string with 1 to 32 characters.
ofcs (Optional)	The name of the OFCS destination.	A string with 1 to 32 characters.
aaa (Optional)	The name of the AAA server destination.	A string with 1 to 32 characters.
userdef1 (Optional)	The name of the first user defined destination.	A string with 1 to 32 characters.
userdef2 (Optional)	The name of the second user defined destination.	A string with 1 to 32 characters.

Response

The `<updateSubscriberResp>` response returns the result of the request to provision subscriber routing entities. There is a single result that applies to all routing entities supplied. Either all routing entities were successfully updated, or no updates were made to any routing entity.

Note: If an IMSI/MSISDN is updated with destination values that already exist, this may result in NO_UPDATES being returned, which is not treated as an error. When a routing entity is not updated, the count of affected rows in the command is not incremented for that IMSI/MSISDN.

If applying all of the provisioning changes results in no database records being modified (because the database already contained the updated values), the NO_UPDATES error code is returned and the number of affected records is 0.

If a subscriber is successfully created or updated, the `description` field contains lists of deleted, created and changed IMSI and MSISDN values.

Response Format

```
lengthInBytes
<updateSubscriberResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
</updateSubscriberResp>
```

Parameters

The parameters for all of the response commands are shown in *XML Response Messages*.

Error Codes

Table 52: <updateSubscriberResp> Error Codes (XML) lists the common error codes for the `<updateSubscriberResp>` response. See *SDS Response Message Error Codes* for a full list of error codes.

Table 52: `<updateSubscriberResp>` Error Codes (XML)

Error Code	Description
SUCCESS	The update request was successfully completed.
NO_UPDATES	All of the changes were already in the database.
DEST_NOT_FOUND	Destination name does not exist.
TOO_MANY_ADDR	Too many address values supplied.
NO_DEST_VAL	No destination name supplied.
MISSING_PARAMETER	A mandatory parameter is missing.
DEST_TYPE_MISMATCH	Destination has a different destination type than the desired destination type.
MULTIPLE_SUBSCRIBERS	Specified parameters refer to multiple subscribers.
SUBSCRIBER_TOO_BIG	Resulting subscriber would exceed 6 IMSI or 6 MSISDN limit.
ACCTID_UPDATE_PROHIBITED	An attempt was made to change an Account ID without specifying the `<deleteAccountId>` tag.
ROUTE_TYPE_MISMATCH	Standalone and subscriber routes are not allowed in same command.

Error Code	Description
DEL_ROUTE_NOT_PERMITTED	Cannot delete last route from subscriber.
NO_ROUTES_SPECIFIED	At least one MSISDN or IMSI must be specified.
ROUTE_DEST_MISMATCH	Specified routes have different destinations.

Examples

Below are examples of how to use the `<updateSubscriber>` request and likely response. Some of these examples are based upon previous requests; hence, the order of the requests could be important.

Add Stand-Alone Routing Entities

This example creates new stand-alone IMSI and MSISDN routing entities and sets their destination values to the specified values.

The result of this request is:

- New IMSI and MSISDN routing entities are created.
- All of the destination values for each routing entity are set to specified values.

Request:

```
<updateSubscriber ent=”subscriberRouting” ns=”dsr” resonly=”n” id=”101”>
  <imsi>111111111100001</imsi>
  <imsi>111111111100002</imsi>
  <imsi>111111111100003</imsi>
  <msisdn>8004605500</msisdn>
  <msisdn>8004605503</msisdn>
  <ltehss>LTE_HSS_1</ltehss>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id=”101”>
  <updateSubscriber ent=”subscriberRouting” ns=”dsr” resonly=”n” id=”101”>
    <imsi>111111111100001</imsi>
    <imsi>111111111100002</imsi>
    <imsi>111111111100003</imsi>
    <msisdn>8004605500</msisdn>
    <msisdn>8004605503</msisdn>
    <ltehss>LTE_HSS_1</ltehss>
  </updateSubscriber>
    <res error=”0” affected=”5”>
</updateSubscriberResp>
```

Update Stand-Alone Routing Entities Destinations

This example updates existing stand-alone IMSI and MSISDN routing entities with new destination values.

Note: This request does not update all NAI values that were specified in the previous request.

The result of this request is that the IMSI and MSISDN routing entities are updated with specified values.

Request:

```
<updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”102”>
  <imsi>111111111100001</imsi>
  <imsi>111111111100002</imsi>
  <imsi>111111111100003</imsi>
  <msisdn>8004605500</msisdn>
  <ltehss>LTE_HSS_4</ltehss>
  <aaa>AAA_4</aaa>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id=”102”>
  <res error=”0” affected=”4”/>
</updateSubscriberResp>
```

Create Subscriber Using Existing Routing Entities (Success)

This example creates a subscriber using existing routing entities that all have the same destination values.

After this request is completed, a new subscriber is created and all of the routing entities are assigned to that subscriber.

Request:

```
<updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”103” group=”y”>
  <imsi>111111111100001</imsi>
  <imsi>111111111100002</imsi>
  <msisdn>8004605500</msisdn>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id=”103”>
  <res error=”0” affected=”1”/>
</updateSubscriberResp>
```

Create Subscriber Using Existing Routing Entities (Failure)

This example fails when creating a subscriber using existing routing entities because the existing routing entities have different destination values.

No changes are made to the database because the request failed.

Request

```
<updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”104” group=”y”>
  <imsi>111111111100003</imsi>
  <msisdn>8004605503</msisdn>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id=”104”>
  <res error=”2029” affected=”0” description=
  ”all routes must have the same destination values”/>
</updateSubscriberResp>
```

Add Account ID to Existing Subscriber

This example adds an Account ID to an existing subscriber. Any of the subscriber's IMSI or MSISDN values can be used. For this example, the MSISDN value is used.

The result of this request is that the subscriber will have an Account ID value.

Request:

```
<updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”105” group=”y”>
  <accountId>8004440000123456789011112</accountId>
  <msisdn>8004605500</msisdn>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id=”105”>
  <res error=”0” affected=”1”>
</updateSubscriberResp>
```

Modify Destinations for Existing Subscriber

This example modifies a destination value for an existing subscriber. Any of the subscriber's IMSI, MSISDN or Account ID values can be used. For this example, an IMSI value is used.

Note: It does not matter if group="y" is specified. The same changes are always applied to the whole subscriber.

The result of this request is that all of the subscriber's IMSI and MSISDN routing entities will have a new destination value

Request:

```
<updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”106”>
  <imsi>111111111100002</imsi>
  <ltehss>LTE_HSS_99</ltehss>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id=”106”>
  <res error=”0” affected=”3”/>
</updateSubscriberResp>
```

Replace Subscriber's MSISDN value

This example replaces an MSISDN value for an existing subscriber. The new MSISDN routing entity inherits the destination values from an old IMSI or MSISDN routing entity. (It doesn't matter which of the Subscriber's routing entities is used because they all have the same destination values.)

The result of this request is:

- The old MSISDN routing entity is deleted from the database.
- The new MSISDN routing entity is added to the database, its destination values are set to the subscriber's destination values, and the new MSISDN value is assigned to the subscriber (relationships are established).

Note: If the new MSISDN routing entity already exists in the database, and it has the same destination values as the subscriber, the only change is that the routing entity is assigned to the subscriber.

Request:

```
<updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”107” group=”y”>
  <deleteMsisdn>8004605500</deleteMsisdn>
  <msisdn>8884605500</msisdn>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id=”107”>
  <res error=”0” affected=”1”/>
</updateSubscriberResp>
```

Replace Subscriber's Account ID, 2 IMSIs, and 1 MSISDN Values

This example replaces several identification (Account ID, IMSI and MSISDN) values for an existing subscriber. The new IMSI and MSISDN routing entities inherit the destination values from the old IMSI and MSISDN routing entities. It does not matter which of the Subscriber's routing entities is used because they all have the same destination values.

The result of this request is:

- The old IMSI and MSISDN routing entities are deleted from the database.
- The new IMSI and MSISDN routing entities are added to the database, their destination values are set to the subscriber's destination values, and the routing entities are assigned to the subscriber (relationships are established).

  Note: If the new IMSI and MSISDN routing entities already exist in the database and they have the same destination values as the subscriber, the only change is that the new IMSI and MSISDN values are assigned to the subscriber.

- The subscriber's Account ID value is changed.

Request:

```
<updateSubscriber ent=”subscriberRouting” ns=”dsr” id=”108” group=”y”>
  <deleteAccountId>800444000012345678901111112</deleteAccountId>
  <deleteImsi>111111111100001</deleteImsi>
  <deleteImsi>111111111100002</deleteImsi>>
  <deleteMsisdn>8884605500</deleteMsisdn>
  <imsi>888888888800001</imsi>
  <imsi>888888888800002</imsi>
  <msisdn>8884605555</msisdn>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id=”108”>
  <res error=”0” affected=”1”/>
</updateSubscriberResp>
```

Create Subscriber Using New Routing Entities (Success)

This example creates a subscriber using new routing entities with specified destinations.

The result of this request is:

- A new subscriber is created with the specified Account ID, IMSI and MSISDN values.
- New IMSI and MSISDN routing entities are created with the specified destinations.

Request:

```
<updateSubscriber ent="subscriberRouting" ns="dsr" id="109" group="y">
  <accountId>111112222233333444445555556</accountId>
  <imsi>333333333300001</imsi>
  <imsi>333333333300002</imsi>
  <msisdn>9198675309</msisdn>
  <ltehss>LTE_HSS_3</ltehss>
  <aaa>AAA_3</aaa>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id="109">
  <res error="0" affected="1"/>
</updateSubscriberResp>
```

Create Subscriber Using New Routing Entities (Failure)

This example fails when creating a subscriber using new routing entities because no destinations were specified.

No changes are made to the database because the request failed.

Request:

```
<updateSubscriber ent="subscriberRouting" ns="dsr" id="110" group="y">
  <accountId>1111122222</accountId>
  <imsi>333333333300003</imsi>
  <imsi>333333333300004</imsi>
  <msisdn>9198675309</msisdn>
</updateSubscriber>
```

Response:

```
<updateSubscriberResp id="110">
  <res error="2013" affected="0" description=
  "at least one destination must be specified"/>
</updateSubscriberResp>
```

Delete Subscriber

Request

The `<deleteSubscriber>` request can be used to delete IMSI and MSISDN routing data and subscriber data. See *Subscriber and Routing Data* for a description of subscriber and routing data.

The request allows for the removal of IMSI, MSISDN, or combinations of IMSI and MSISDN routing entities. Each routing entity contains up to eight destination names.

When the `group="y"` attribute is specified, the request deletes all data associated with the subscriber, including the Account ID, all relationships, and all IMSI and MSISDN routing entities that were assigned to the subscriber.

When `group="y"` is not specified or when `group="n"` is specified, only IMSI and MSISDN routing entities are deleted. If the IMSI or MSISDN value is assigned to a subscriber and there is at least one more IMSI or MSISDN value assigned to the subscriber, the IMSI or MSISDN value is removed from the subscriber.

The last IMSI or MSISDN value cannot be removed from a subscriber - the user must delete the whole subscriber by specifying the `group="y"` attribute.

Semantic Rules (requests that do not specify the group attribute or specify `group="n"`)

- All specified IMSI or MSISDN values must be assigned to one subscriber or must exist in stand-alone routing entities.
- The `accountID` parameter cannot be specified.
- At least one routing entity (IMSI or MSISDN) must be specified.
- A maximum of 10 routing entities (IMSI, MSISDN, or combinations of the two) can be specified.
- The last IMSI or MSISDN for a subscriber cannot be deleted. Use `group="y"` to delete whole subscriber).

Semantic Rules (requests that specify `group="y"`)

- All specified `accountId`, `imsi`, and `msisdn` values must be assigned to one subscriber. The specified `imsi` or `msisdn` values cannot exist in a stand-alone routing entity.
- The accountID parameter can be specified.
- At least 1 `imsi`, `msisdn`, or `accountId` value must be specified.
- A maximum of 6 `imsi`, 6 `msisdn`, and 1 `accountId` value can be specified.

Request Format

```
<deleteSubscriber ent="subscriberRouting" ns="dsr" [resonly="resonly"]
                  [id="id"] [timeout="timeout"] [group="group"]>
[   <accountId>accountId</accountiId> ]
[
    <imsi>imsi</imsi>
    …
    <imsi>imsi</imsi>
]
[
    <msisdn>msisdn</msisdn>
    …
    <msisdn>msisdn</msisdn>
]
</deleteSubscriber>
```

Request Parameters

Table 53: `<deleteSubscriber>` Request Parameters (XML)

Parameter	Definition	Values
ent	The entity name within the global schema.	subscriberRouting
ns	The namespace within the global schema.	dsr
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction ID value provided in request and passed back in the response.	1-4294967295
timeout (Optional)	The amount of time (in seconds) to wait to before being able to perform a write if another connection is performing a write, or has a transaction open. Clients waiting to write are processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	0 (return immediately if not available) to 3600 seconds (default is 0).
group (Optional)	Indicates if all of the subscriber's data should be deleted or just specified IMSI or MSISDN routing entities.	• y - Delete subscriber and all of its IMSI and MSISDN routing entities. • n - Only delete specified MSISDN and IMSI routing entities (default).
accountId	A user-defined Account ID value to delete.	1 to 26 numeric digits.
imsi (Optional)	An IMSI (specified in E.212 format).	10 to 15 numeric digits.
msisdn (Optional)	An MSISDN (specified in E.164 international public telecommunication numbering plan format).	8 to 15 numeric digits.

Response

The <deleteSubscriberResp> response returns the result of the request to delete subscriber routing entities. There is a single result that applies to all routing entities supplied. Either all subscriber and/or routing entities were successfully deleted, or no deletes are made.

If applying all of the delete changes results in no routing entities being deleted (because the database already did not contain the specified values), the NO_UPDATES error code is returned and the number of affected records is 0. If a subscriber is successfully deleted, the description field contains lists of deleted IMSI and MSISDN values.

When a routing entity does not exist, this means that the affected rows count is not incremented for that IMSI/MSISDN.

Response Format

```
lengthInBytes
<deleteSubscriberResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
</deleteSubscriberResp>
```

Parameters

The parameters for all of the XML response commands are shown in *XML Response Messages*.

Error Codes

Table 54: <deleteSubscriberResp> Error Codes (XML) lists the common error codes for the <deleteSubscriberResp> command. See *SDS Response Message Error Codes* for a full list of error codes.

Table 54: <deleteSubscriberResp> Error Codes (XML)

Error Code	Description
SUCCESS	The delete request was successfully completed.
NO_UPDATES	The records were already deleted from the database.
TOO_MANY_ADDR	Too many address values supplied.
MISSING_PARAMETER	A mandatory parameter is missing.
MULTIPLE_SUBSCRIBERS	Specified parameters refer to multiple subscribers.
ROUTE_TYPE_MISMATCH	Standalone and subscriber routes are not allowed in same command.
DEL_ROUTE_NOT_PERMITTED	Cannot delete last route from subscriber.

Examples

Delete Stand-Alone Routing Entities

This example deletes stand-alone IMSI and MSISDN routing entities.

Request:

```
<deleteSubscriber ent=”subscriberRouting” ns=”dsr” id=”101”>
  <imsi>111111111100021</imsi>
  <imsi>111111111100022</imsi>
  <msisdn>8004605520</msisdn>
</deleteSubscriber>
```

Response:

```
deleteSubscriberResp id=”101”>
  <res error=”0” affected=”3”/>
</deleteSubscriberResp>
```

Delete Several Routing Entities

This example successfully deletes two stand-alone IMSI routing entities. Other IMSI values were not found and were not deleted.

Request:

```
<deleteSubscriber ent=”subscriberRouting” ns=”dsr” id=”102”>
  <imsi>777777777777777</imsi>
  <imsi>111111111100001</imsi>
  <imsi>111111111100002</imsi>
  <imsi>888888888888888</imsi>
</deleteSubscriber>
```

Response:

```
<deleteSubscriberResp id=”102”>
  <res error=”0” affected=”2”/>
</deleteSubscriberResp>
```

Delete Routing Entities Assigned to the Same Subscriber

This example deletes IMSI and MSISDN routing entities that are assigned to the same subscriber. The example assumes that the subscriber has at least one more routing entity other than the specified values.

Request

```
<deleteSubscriber ent=”subscriberRouting” ns=”dsr” id=”103”>
  <imsi>111111111100002</imsi>
  <msisdn>8004605500</msisdn>
</deleteSubscriber>
```

Response:

```
<<deleteSubscriberResp id="103">
  <res error="0" affected="2"/>
</deleteSubscriberResp>
```

Delete Last Routing Entity for a Subscriber (success)

This example successfully deletes the subscriber and all IMSI and MSISDN routing entities assigned to the subscriber. Any of the subscriber's Account ID, MSISDN or IMSI values can be specified. In this example, all of the IMSI and MSISDN values are specified even though only one value is required.

Request:

```
<deleteSubscriber ent="subscriberRouting" ns="dsr" id="105"
                  timeout="10" group="y">
  <imsi>111111111100001</imsi>
  <imsi>111111111100002</imsi>
  <msisdn>8004605500</msisdn>
</deleteSubscriber>
```

Response:

```
<deleteSubscriberResp id="105">
  <res error="0" affected="1"/>
</deleteSubscriberResp>
```

Delete Last Routing Entity for a Subscriber (failure)

This example attempts to delete IMSI and MSISDN routing entities that are assigned to the same subscriber. The example fails because the subscriber does not have any more routing entities.

No changes are made to the database because the request failed.

Request:

```
<deleteSubscriber ent="subscriberRouting" ns="dsr" id="104" timeout="10">
  <imsi>111111111100001</imsi>
  <imsi>111111111100002</imsi>
  <msisdn>8004605500</msisdn>
</deleteSubscriber>
```

Response:

```
deleteSubscriberResp id="104">
  <res error="2028" affected="0" description=
  "cannot delete the last route from subscriber"/>
</deleteSubscriberResp>
```

Delete a Subscriber (success)

This example successfully deletes the subscriber and all IMSI and MSISDN routing entities assigned to the subscriber. Any of the subscriber's Account ID, MSISDN or IMSI values can be specified. In this example, the Account ID is specified.

Request:

```
<deleteSubscriber ent=”subscriberRouting” ns=”dsr” id=”106” group=”y”>
  <accountId>800444000012345678901111112</accountId>
</deleteSubscriber>
```

Response:

```
<deleteSubscriberResp id=”106”>
  <res error=”0” affected=”1”/>
</deleteSubscriberResp>
```

Read Subscriber

Request

The `<readSubscriber>` request extracts IMSI and MSISDN routing and subscriber data. See *Subscriber and Routing Data* for a description of subscriber and routing data. Each routing entity contains up to eight destination names.

When the `group="y"` attribute is specified, the request extracts and displays all data associated with the subscriber. The returned response will have the Subscriber Account ID, all IMSI and MSISDN values, and the eight destination values from any of the subscriber routing entities is returned in the response. All routing entities for a subscriber have the same destination values; hence, any routing entity can be used to extract the values.

When `group="y"` is not specified or when `group="n"` is specified, only the specified IMSI and MSISDN routing entities are retrieved. The returned response will have each IMSI or MSISDN value along with individual up to eight destination values.

Semantic Rules (requests that do not specify the group attribute or specify `group="n"`)

- All specified `imsi` or `msisdn` values must be assigned to one subscriber or must exist in stand-alone routing entities.
- The `accountId` parameter cannot be specified.
- At least one routing entity (IMSI or MSISDN) must be specified.
- A maximum of 10 routing entities (IMSI, MSISDN, or combinations of the two) can be specified.

Semantic Rules (requests that specify `group="y"`)

- All specified `accountId`, `imsi`, or `msisdn` values must be assigned to one subscriber. The specified `imsi` or `msisdn` values cannot exist in a stand-alone routing entity.
- The `accountId` parameter can be specified.
- A maximum of 6 `imsi`, 6 `msisdn`, and 1 `accountId` values can be specified.

Request Format

```
<readSubscriber ent="subscriberRouting" ns="dsr" [resonly="resonly"] [id="id"]
                [timeout="timeout"] [group="group"]>
```

```
[   <accountId>accountId</accountId> ]
[
    <imsi>imsi</imsi>

    <imsi>imsi</imsi>
]
[
    <msisdn>msisdn</msisdn>
    …
    <msisdn>msisdn</msisdn>
]
</readSubscriber>
```

Request Parameters

Table 55: `<readSubscriber>` Request Parameters (XML)

Parameter	Description	Values
ent	The entity name within the global schema.	subscriberRouting
ns	The namespace within the global schema.	dsr
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction ID value provided in request and passed back in the response.	1-4294967295
timeout (Optional)	The amount of time (in seconds) to wait before being able to perform a read if another connection is performing a write, or has a transaction open. Clients waiting to read will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	0 (return immediately if not available) to 3600 seconds. The default is 0.
group (Optional)	Indicates if all subscriber data should be retrieved or just specified IMSI or MSISDN routing entities.	• y - Read subscriber and all of its IMSI and MSISDN routing entities.

Parameter	Description	Values
		• n - Only read specified MSISDN and IMSI routing entities (default).
accountId (Optional)	A user-defined Account ID value to read.	1 to 26 numeric digits.
imsi (Optional)	An IMSI (specified in E.212 format).	10 to 15 numeric digits.
msisdn (Optional)	An MSISDN (specified in E.164 international public telecommunication numbering plan format).	8 to 15 numeric digits.

Response

The `<readSubscriberResp>` response returns the result of the request to read subscriber routing entities. Only those subscribers or routing entities that are found are returned. The response message contains up to eight destinations (one for each destination type, such as `<ltehss>`) for each routing entity or subscriber. Only provisioned destination names are displayed. (i.e. destination names=*"none"* are not displayed).

Variations can occur in the response, depending on whether a subscriber is being retrieved or routing entities are being retrieved.

If routing entities are retrieved (`group="y"` was not specified or `group="n"` was specified):

- There will not be any `<subscriber>` or `<accountId>` tags.
- The destination values are listed within each IMSI or MSISDN routing entity value.

If a subscriber is retrieved (`group="y"` is specified):

- The `<subscriber>` tag is used within the `<rset>` tag.
- The `<accountId>` tag is displayed if the subscriber has an Account ID value defined.
- The destination values are listed once, after the last routing entity.

Response Format (`group="y"` is not specified)

```
lengthInBytes
<readSubscriberResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
[
    <rset>
    [
        <imsi imsi="imsi">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
```

```
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </imsi>
        ...
        <imsi imsi="imsi">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </imsi>
    ]
    [
        <msisdn msisdn="msisdn">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </msisdn>
        ...
        <msisdn msisdn="msisdn">
        [   <imshss>imshss</imshss>       ]
        [   <ltehss>ltehss</ltehss>       ]
        [   <pcrf>pcrf</pcrf>             ]
        [   <ocs>ocs</ocs>                ]
        [   <ofcs>ofcs</ofcs>             ]
        [   <aaa>aaa</aaa>                ]
        [   <userdef1>userdef1</userdef1> ]
        [   <userdef2>userdef2</userdef2> ]
        </msisdn>
    ]
    </rset>
]
</readSubscriberResp>
```

Response Format (`group="y"`) is specified

```
lengthInBytes
<readSubscriberResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
[
    <rset>
        <subscriber>
        [   <accountiId>accountId</accountId>]
        [   <imsi>imsi</imsi>              ]
                ...
        [   <imsi>imsi</imsi>              ]
        [   <msisdn>msisdn</msisdn>        ]
                ...
        [   <msisdn>msisdn</msisdn>        ]
        [   <imshss>imshss</imshss>        ]
```

```
        [   <ltehss>ltehss</ltehss>         ]
        [   <pcrf>pcrf</pcrf>               ]
        [   <ocs>ocs</ocs>                  ]
        [   <ofcs>ofcs</ofcs>               ]
        [   <aaa>aaa</aaa>                  ]
        [   <userdef1>userdef1</userdef1>   ]
        [   <userdef2>userdef2</userdef2>   ]
       </subscriber>
    </rset>
]
</readSubscriberResp>
```

Response Parameters

Table 56: `<readSubscriberResp>` Parameters (XML)

Parameter	Description	Response
lengthinbytes	Number of bytes following to form XML request. This is a 4 byte binary value.	0-4294967295
id (Optional)	Transaction id value provided in request and passed back in the response.	1-4294967295
originalXMLRequest (Optional)	The text of the original `<readSubscriber>` XML request that was sent. Note: this is only present if the `resonly="n"` attribute is set in the original request.	A string with 1 to 4096 characters.
error	Error code. Whether or not operation was successfully executed by the XDS.	0 - success, non zero - failure.
affected	If `group="y"`, then the number of subscribers read (0 or 1). Otherwise, the number of routing entities read (0 - 10).	0-10
description (Optional)	A textual description associated with the response. This may contain more information as to why a request failed.	A string with 1 to 1024 characters.
rset	Contains 1 row for each extracted record. Each row contains a stand-alone routing entity (MSISDN or IMSI value with its destination values) or a subscriber (list of related MSISDN, IMSI and Account ID values with the destination	

Parameter	Description	Response
	values that are used by all routing entities assigned to the subscriber.)	
subscriber (Optional)	Contains all IMSI and MSISDN values for a specific subscriber with an optional Account ID and all destinations defined for the subscriber.	
accountId (Optional)	A user-defined Account ID value.	1 to 26 numeric digits
imsi (Optional)	An IMSI (specified in E.212 format).	10 to 15 numeric digits
msisdn (Optional)	An MSISDN (specified in E.164 international public telecommunication numbering plan format).	8 to 15 numeric digits.
imshss (Optional)	The name of the IMS HSS destination.	A string with 1 to 32 characters.
ltehss (Optional)	The name of the LTE HSS destination.	A string with 1 to 32 characters.
pcrf (Optional)	The name of the PCRF destination.	A string with 1 to 32 characters.
ocs (Optional)	The name of the OCS destination.	A string with 1 to 32 characters.
ofcs (Optional)	The name of the OFCS destination.	A string with 1 to 32 characters.
aaa (Optional)	The name of the AAA server destination.	A string with 1 to 32 characters.
userdef1 (Optional)	The name of the first user defined destination.	A string with 1 to 32 characters.
userdef2 (Optional)	The name of the second user defined destination.	A string with 1 to 32 characters.

Response Error Codes

Table 57: <readSubscriberResp> Error Codes (XML) lists the common error codes for the `<readSubscriberResp>` command. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 57: `<readSubscriberResp>` Error Codes (XML)

Error Code	Description
SUCCESS	The read request was successfully completed.
TOO_MANY_ADDR	Too many address values supplied.
MISSING_PARAMETER	A mandatory parameter is missing.
IMSI_NOT_FOUND	The specified IMSI does not exist.
MSISDN_NOT_FOUND	The specified MSISDN does not exist.
SUBSCRIBER_NOT_FOUND	The subscriber does not exist.
MULTIPLE_SUBSCRIBERS	Specified parameters refer to multiple subscribers.
ROUTE_TYPE_MISMATCH	Standalone and subscriber routes are not allowed in same command.

Examples

The format of the response differs depending on whether the `group="y"` attribute is specified.

If `group="y"` is NOT specified, then each routing entity that was found is displayed with its destination values.

If `group="y"` is specified, then the result response includes an optional Account ID value (if it exists), all MSISDN and IMSI values for that subscriber, and one set of destination values (all routing entities within a subscriber have the same destination values).

Read Routing Entities (not subscribers)

This example reads IMSI and MSISDN routing entities and displays their destination values. It does not matter if any of the routing entities are assigned to a subscriber because the same result will occur.

Request:

```
<readSubscriber ent=”subscriberRouting” ns=”dsr” id=”101”>
  <imsi>111111111100001</imsi>
  <imsi>111111111100002</imsi>
  <msisdn>8004605500</msisdn>
</readSubscriber>
```

Response:

```
<readSubscriberResp>
  <res error=”0” affected=”3”/>
  <rset>
    <imsi imsi="111111111100001">
      <ltehss>LTE_HSS_4</ltehss>
      <aaa>AAA_4</aaa>
    </imsi>
    <imsi imsi="111111111100002">
     <ltehss>LTE_HSS_4</ltehss>
     <aaa>AAA_4</aaa>
    </imsi>
```

```
    <msisdn msisdn="8004605500">
     <ltehss>LTE_HSS_4</ltehss>
     <aaa>AAA_4</aaa>
    </msisdn>
  </rset>
</readSubscriberResp>
```

Read Routing Entities with Not Found MSISDN/IMSI Values

This example reads IMSI and MSISDN routing entities and displays their destination values. In this example, one MSISDN and one IMSI value do not exist, so the response returns the two values that do exist. The same result will occur if any of the routing entities are assigned to a subscriber.

Request:

```
<readSubscriber ent=”subscriberRouting” ns=”dsr” id=”102”>
  <imsi>777777777777777</imsi>
  <imsi>111111111100002</imsi>
  <msisdn>8004605500</msisdn>
  <msisdn>88888888888888</msisdn>
</readSubscriber>
```

Response:

```
<readSubscriberResp>
  <res error=”0” affected=”2”/>
  <rset>
    <imsi imsi="111111111100002">
     <ltehss>LTE_HSS_4</ltehss>
     <aaa>AAA_4</aaa>
    </imsi>
    <msisdn msisdn="8004605500">
     <ltehss>LTE_HSS_4</ltehss>
     <aaa>AAA_4</aaa>
    </msisdn>
  </rset>
</readSubscriberResp>
```

Read Subscriber (success)

This example reads a subscriber and displays all of the subscriber data. Any of the subscriber Account ID, MSISDN or IMSI values can be specified. In this example, the MSISDN value is specified.

Request:

```
<readSubscriber ent=”subscriberRouting” ns=”dsr” id=”103” group=”y”>
    <msisdn>8004605500</msisdn>
</readSubscriber>
```

Response:

```
<readSubscriberResp>
    <res error=”0” affected=”1”/>
    <rset>
      <subscriber>
        <accountId>800444000012345678901111112</accountId>
        <imsi>"111111111100001></imsi>
        <imsi>111111111100002</imsi>
        <msisdn>8004605500</msisdn>
```

```
        <ltehss>LTE_HSS_4</ltehss>
        <aaa>AAA_4</aaa>
      </subscriber>
    </rset>
</readSubscriberResp>
```

Read Subscriber Fails for Stand-alone Routing Entity

This example attempts to read a subscriber. The request fails because the specified MSISDN value is for a stand-alone routing entity.

Request

```
<readSubscriber ent=”subscriberRouting” ns=”dsr” id=”1041” group=”y”>
  <msisdn>8004605503</msisdn>
</readSubscriber>
```

Response:

```
<readSubscriberResp>
  <res error=”2022” affected=”0” description=”subscriber not found”/>
</readSubscriberResp>
```

Update Subscriber NAI

Request

The `<updateSubscriberNai>` request provisions NAI routing entities. Each NAI value is defined as a combination of an NAI host and NAI user value. For example, "John.Smith@tekelec.com" would have "John.Smith" as the NAI user value and "tekelec.com" as the NAI host value.

Each routing entity contains up to eight destination names. Each destination contains FQDN and realm values, which are used for routing messages. The request can remove a destination value from existing NAI routing entities by specifying `"none"` as the destination name.

The request can add new routing entities or update destination names in existing routing entities. These destination changes are applied to all specified NAI routing entities.

Semantic Rules

- Between 1 and 10 user names must be specified.
- At least one destination must be specified.
- The host name must already exist in the database.
- A destination name must already exist in the database.
- Each destination name type may only be specified once.
- All specified routing entities will be provisioned with the same destination value(s).
- Any existing destination(s) for a routing entity will not be changed/removed if not specified in the request.
- Specifying a destination name of `“none”` will remove the association of that destination from the specified routing entity(s).

Request Format

```
<updateSubscriberNai ent="subscriberRouting" ns="dsr" [resonly="resonly"]
                     [id="id"] [timeout="timeout"]>
    <host>host</host>
    <user>user</user>
[
    <user>user</user>
    …
    <user>user</user>
]
[   <imshss>imshss</imshss>       ]
[   <ltehss>ltehss</ltehss>       ]
[   <pcrf>pcrf</pcrf>             ]
[   <ocs>ocs</ocs>                ]
[   <ofcs>ofcs</ofcs>             ]
[   <aaa>aaa</aaa>                ]
[   <userdef1>userdef1</userdef1> ]
[   <userdef2>userdef2</userdef2> ]
</updateSubscriberNai>
```

Request Parameters

Table 58: `<updateSubscriberNai>` Request Parameters (XML)

Parameter	Description	Values
ent	The entity name within the global schema.	subscriberRouting
ns	The namespace within the global schema.	dsr
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction id value provided in request, and will be passed back in the response	1-4294967295
timeout (Optional)	The amount of time (in seconds) to wait to before being able to perform a write if another connection is performing a write, or has a transaction open. Clients waiting to write will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	0 (return immediately if not available) to 3600 seconds. The default is 0.

Parameter	Description	Values
host	A host name.	A string with 1 to 64 characters.
user	A user name to be associated with the host to form an NAI.	A string with 1 to 64 characters.
imshss (Optional)	The name of the IMS HSS destination.	A string with 1 to 32 characters.
ltehss (Optional)	The name of the LTE HSS destination.	A string with 1 to 32 characters.
pcrf (Optional)	The name of the PCRF destination.	A string with 1 to 32 characters.
ocs (Optional)	The name of the OCS destination.	A string with 1 to 32 characters.
ofcs (Optional)	The name of the OFCS destination.	A string with 1 to 32 characters.
aaa (Optional)	The name of the AAA server destination.	A string with 1 to 32 characters.
userdef1 (Optional)	The name of the first user defined destination.	A string with 1 to 32 characters.
userdef2 (Optional)	The name of the second user defined destination.	A string with 1 to 32 characters.

Response

The `<updateSubscriberNaiResp>` response returns the result of the request to provision subscriber routing entities. There is a single result that applies to all routing entities supplied. Either all routing entities were successfully updated, or no updates were made to any routing entity.

Note: If applying all of the provisioning changes results in no database records being modified because the database already contained the updated values, then the NO_UPDATES error code is returned, and the number of affected records is 0.

Response Format

```
lengthInBytes
<updateSubscriberNaiResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
</updateSubscriberNaiResp>
```

Response Parameters

The parameters for all of the XML response commands are shown in *XML Response Messages*.

Response Error Codes

Table 59: `<updateSubscriberNaiResp>` Error Codes (XML)

Error Code	Description
SUCCESS	The update request was successfully completed.
NO_UPDATES	All of the changes were already in the database.
NAI_HOST_NOT_FOUND	Host name does not exist.
TOO_MANY_NAI	Too many NAI values supplied.
NO_DEST_VAL	No destination name supplied.
MISSING_PARAMETER	A mandatory parameter is missing.
DEST_NOT_FOUND	Destination name does not exist.
DEST_TYPE_MISMATCH	Destination has a different destination type than the desired destination type.

Examples

Some of the following examples are based upon previous requests. The order of the requests can be important.

Add New NAI Routing Entities

This example creates three new NAI routing entities and sets their destination values to the specified values. This example assumes that the host and destination values already exist.

The result of this request is:

- New NAI routing entities are created.
- All destination values for each routing entity are set to specified values.

Request:

```
<updateSubscriberNai ent=”subscriberRouting” ns=”dsr” id=”101”>
  <host>tekelec.com</host>
  <user>John.Smith</user>
  <user>Jane.Doe</user>
  <user>Mike.Jones</user>
  <imshss>IMS_HSS_1</imshss>
  <ltehss>LTE_HSS_1</ltehss>
  <aaa>AAA_Texas</aaa>
</updateSubscriberNai>
```

Response:

```
<updateSubscriberNaiResp id=”101”>
  <res error=”0” affected=”3”/>
</updateSubscriberNaiResp>
```

Update NAI Routing Entities Destinations (success)

This example updates existing NAI routing entities with new destination values.

Note: This request does not update all NAI values that were specified in the previous request.

The result of this request is that the specified NAI routing entities are updated with specified values.

Request:

```
<updateSubscriberNai ent=”subscriberRouting” ns=”dsr” id=”102”>
  <host>tekelec.com</host>
  <user>Jane.Doe</user>
  <user>Mike.Jones</user>
  <ltehss>LTE_HSS_4</ltehss>
  <pcrf>PCRF_Ohio</pcrf>
</updateSubscriberNai>
```

Response:

```
<updateSubscriberNaiResp id=”102”>
  <res error=”0” affected=”2”/>
</updateSubscriberNaiResp>
```

Update NAI Routing Entities Destinations (failure)

This example fails to update existing NAI routing entities with new destination values because the destination does not exist.

No changes are made to the database because the request failed.

Request:

```
<updateSubscriberNai ent=”subscriberRouting” ns=”dsr” id=”103”>
  <host>tekelec.com</host>
  <user>Jane.Doe</user>
  <ltehss>junk</ltehss>
</updateSubscriberNai>
```

Response:

```
<updateSubscriberNaiResp id=”102”>
  <res error=”2006” affected=”0” description="destination not found”/>
</updateSubscriberNaiResp>
```

Delete Subscriber NAI

Request

The <deleteSubscriberNai> request removes NAI routing data.

Each NAI value is defined as a combination of an NAI host and NAI user value. For example, "John.Smith@tekelec.com" would have "John.Smith" as the NAI user value and "tekelec.com" as the

NAI host value. The <deleteSubscriberNai> command removes the NAI user value, but does not affect the NAI host value.

Semantic Rules

- Between 1 and 10 user names must be specified.
- The host name must already exist in the database.

Request Format

```
<deleteSubscriberNai ent="subscriberRouting" ns="dsr" [resonly="resonly"]
                     [id="id"] [timeout="timeout"]>
    <host>host</host>
    <user>user</user>
[
    <user>user</user>
    …
    <user>user</user>
]
</deleteSubscriberNai>
```

Request Parameters

Table 60: <deleteSubscriberNai> Request Parameters (XML)

Parameter	Description	Values
ent	The entity name within the global schema.	subscriberRouting
ns	The namespace within the global schema.	dsr
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction ID value provided in request and passed back in the response.	1-4294967295
timeout (Optional)	The amount of time (in seconds) to wait to before being able to perform a write if another connection is performing a write, or has a transaction open. Clients waiting to write will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as	0 (return immediately if not available) to 3600 seconds. The default is 0.

Parameter	Description	Values
	the client already has a transaction open.	
host	A host name.	A string with 1 to 64 characters.
user	A user name to be associated with the host to form an NAI.	A string with 1 to 64 characters.

Response

The `<deleteSubscriberNaiResp>` response returns the result of the request to delete subscriber routing entities. A single result that applies to all routing entities supplied. Either all routing entities were successfully deleted, or no deletes were made.

If applying all of the delete requests results in no database records being deleted (because they already did not exist in the database), the NO_UPDATES error code is returned and the number of affected records is 0.

Response Format

```
lengthInBytes
<deleteSubscriberNaiResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
</deleteSubscriberNaiResp>
```

Response Parameters

The parameters for all of the XML response commands are shown in *XML Response Messages*.

Response Error Codes

Table 61: <deleteSubscriberNaiResp> Error Codes (XML) lists the common error codes for `<deleteSubscriberNaiResp>`. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 61: `<deleteSubscriberNaiResp>` Error Codes (XML)

Error Code	Description
SUCCESS	The delete request was successfully completed.
NO_UPDATES	All of the records were already deleted from the database.
NAI_HOST_NOT_FOUND	Host name does not exist.
TOO_MANY_NAI	Too many NAI values supplied.
NO_NAI_VAL	No NAI value supplied.

Examples

Delete NAI Routing Entities

This example successfully deletes three NAI routing entities.

Request:

```
<deleteSubscriberNai ent=”subscriberRouting” ns=”dsr” id=”101”>
  <host>tekelec.com</host>
  <user>John.Smith</user>
  <user>Jane.Doe</user>
  <user>Mike.Jones</user>
</deleteSubscriberNai>
```

Response:

```
<deleteSubscriberNaiResp id=”101”>
  <res error=”0” affected=”3”/>
</deleteSubscriberNaiResp>
```

Delete Several NAI Routing Entities

This example successfully deletes two NAI routing entities. Other NAI values were not found and were not deleted.

Request:

```
<deleteSubscriberNai ent=”subscriberRouting” ns=”dsr” id=”102”>
  <host>tekelec.com</host>
  <user>John.Smith</user>
  <user>Ann.Jones</user>
  <user>Jane.Doe</user>
  <user>Mike.Jackson</user>
</deleteSubscriberNai>
```

Response:

```
<deleteSubscriberNaiResp id=”102”>
  <res error=”0” affected=”2”/>
</deleteSubscriberNaiResp>
```

Read Subscriber NAI

Request

The `<readSubscriberNai>` request extracts (reads) NAI routing entities and displays the first eight destination values for each routing entity.

Semantic Rules

- Between 1 and 10 user names must be specified.
- The host name must already exist in the database.

Request Format

```
<readSubscriberNai ent="subscriberRouting" ns="dsr" [resonly="resonly"]
                   [id="id"]>
    <host>host</host>
    <user>user</user>
[
    <user>user</user>
    …
    <user>user</user>
]
</readSubscriberNai>
```

Request Parameters

Table 62: `<readSubscriberNai>` Request Parameters (XML)

Parameter	Description	Values
ent	The entity name within the global schema.	subscriberRouting
ns	The namespace within the global schema.	dsr
resonly (Optional)	Indicates whether the response should consist of the result only, without including the original request in the response.	• y - Only provide the result, do not include the original request (default). • n - Include the original request in the response.
id (Optional)	Transaction id value provided in request and passed back in the response.	1-4294967295
timeout (Optional)	The amount of time (in seconds) to wait before being able to perform a write if another connection is performing a write, or has a transaction open. Clients waiting to write will be processed in the order that their requests were received. If the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.	0 (return immediately if not available) to 3600 seconds. The default is 0.

Parameter	Description	Values
host	A host name.	A string with 1 to 64 characters.
user	A user name to be associated with the host to form an NAI.	A string with 1 to 64 characters.

Response

The <readSubscriberNaiResp> response returns the result of the request to read NAI subscriber routing entities. Only those NAI subscriber routing entities that are found are returned. The response message contains up to eight destinations (one for each destination type, such as <ltehss>) for each routing entity. Only provisioned destination names are displayed. (i.e. destination names=*"none"* are not displayed).

Response Format

```
lengthInBytes
<readSubscriberNaiResp [id="id"]>
[
    originalXMLRequest
]
    <res error="error" affected="affected" [description="description"]/>
[
    <rset>
        <nai host="host" user="user">
    [       <imshss>imshss</imshss>       ]
    [       <ltehss>ltehss</ltehss>       ]
    [       <pcrf>pcrf</pcrf>             ]
    [       <ocs>ocs</ocs>                ]
    [       <ofcs>ofcs</ofcs>             ]
    [       <aaa>aaa</aaa>                ]
    [       <userdef1>userdef1</userdef1> ]
    [       <userdef2>userdef2</userdef2> ]
        </nai>
    [
        ...
        <nai host="host" user="user">
    [       <imshss>imshss</imshss>       ]
    [       <ltehss>ltehss</ltehss>       ]
    [       <pcrf>pcrf</pcrf>             ]
    [       <ocs>ocs</ocs>                ]
    [       <ofcs>ofcs</ofcs>             ]
    [       <aaa>aaa</aaa>                ]
    [       <userdef1>userdef1</userdef1> ]
    [       <userdef2>userdef2</userdef2> ]
        </nai>
    ]
    </rset>
]
</readSubscriberResp>
```

Response Parameters

Table 63: `<readSubscriberNaiResp>` Parameters (XML)

Parameter	Description	Values
lengthInBytes	Number of bytes following to form XML request. This is a 4 byte binary value.	0-4294967295
id (Optional)	Transaction id value provided in request and passed back in the response.	1-4294967295
originalXMLRequest (Optional)	The text of the original `<readSubscriber>` XML request that was sent. This is only present if the resonly=”n” attribute is set in the original request.	A string with 1 to 4096 characters.
error	Whether or not operation was successfully executed by the SDS.	0 - success, non zero - failure.
affected	The number of routing entities read.	0-10
description (Optional)	A textual description associated with the response. This may contain more information as to why a request failed. Only present when the request fails.	A string with 1 to 1024 characters.
<rset> XML tag (Optional)	Indicates rows of data are returned. If no records are being returned, this tag is not be present.	
host	A host name, which is used with all user values.	A string with 1 to 64 characters.
user	The NAI user name to be associated with the host to form an NAI.	A string with 1 to 64 characters. Must have 1-10 user values.
imshss (Optional)	The name of the IMS HSS destination.	A string with 1 to 32 characters.
ltehss (Optional)	The name of the LTE HSS destination.	A string with 1 to 32 characters.
pcrf (Optional)	The name of the PCRF destination.	A string with 1 to 32 characters.
ocs (Optional)	The name of the OCS destination.	A string with 1 to 32 characters.

Parameter	Description	Values
ofcs (Optional)	The name of the OFCS destination.	A string with 1 to 32 characters.
aaa (Optional)	The name of the AAA server destination.	A string with 1 to 32 characters.
userdef1 (Optional)	The name of the first user defined destination.	A string with 1 to 32 characters.
userdef2 (Optional)	The name of the second user defined destination.	A string with 1 to 32 characters.

Response Error Codes

Table 64: <readSubscriberNaiResp> Error Codes (XML) lists the common error codes for the `<readSubscriberNaiResp>` command. See *SDS Response Message Error Codes* for a complete list of error codes.

Table 64: `<readSubscriberNaiResp>` Error Codes (XML)

Error Code	Description
SUCCESS	The delete request was successfully completed.
NAI_HOST_NOT_FOUND	Host name does not exist.
NAI_NOT_FOUND	None of the specified NAI exists.
TOO_MANY_NAI	Too many NAI values supplied.
MISSING_PARAMETER	A mandatory parameter is missing.

Examples

Read NAI Routing Entities

This example successfully reads three NAI routing entities.

Request:

```
<readSubscriberNai ent=”subscriberRouting” ns=”dsr” id=”101”>
  <host>tekelec.com</host>
  <user>John.Smith</user>
  <user>Jane.Doe</user>
  <user>Mike.Jones</user>
</readSubscriberNai>
```

Response:

```
<readSubscriberNaiResp id=”101”>
  <res error=”0” affected=”3”/>
    <rset>
      <nai host=”tekelec.com" user=”John.Smith”>
        <imshss>IMS_HSS_1</imshss>
```

```
        <ltehss>LTE_HSS_1</ltehss>
        <aaa>AAA_Texas</aaa>
      </nai>
      <nai host=”tekelec.com" user=”Jane.Doe”>
        <imshss>IMS_HSS_1</imshss>
        <ltehss>LTE_HSS_4</ltehss>
        <pcrf>PCRF_Ohio</pcrf>
        <aaa>AAA_Texas</aaa>
      </nai>
      <nai host=”tekelec.com" user=”Mike.Jones”>
        <imshss>IMS_HSS_1</imshss>
        <ltehss>LTE_HSS_4</ltehss>
        <pcrf>PCRF_Ohio</pcrf>
        <aaa>AAA_Texas</aaa>
      </nai>
    </rset>
</readSubscriberNaiResp>
```

Read NAI Routing Entities

This example successfully reads two NAI routing entities. Other NAI values are not found

Request:

```
<readSubscriberNai ent=”subscriberRouting” ns=”dsr” id=”102”>
  <host>tekelec.com</host>
  <user>John.Smith</user>
  <user>Ann.Jones</user>
  <user>Jane.Doe</user>
  <user>Mike.Jackson</user>
</readSubscriberNai>
```

Response:

```
<readSubscriberNaiResp id=”102”>
  <res error=”0” affected=”2”/>
    <rset>
      <nai host=”tekelec.com" user=”John.Smith”>
        <imshss>IMS_HSS_1</imshss>
        <ltehss>LTE_HSS_1</ltehss>
        <aaa>AAA_Texas</aaa>
      </nai>
      <nai host=”tekelec.com" user=”Jane.Doe”>
        <imshss>IMS_HSS_1</imshss>
        <ltehss>LTE_HSS_4</ltehss>
        <pcrf>PCRF_Ohio</pcrf>
        <aaa>AAA_Texas</aaa>
      </nai>
    </rset>
</readSubscriberNaiResp>
```

Read NAI Routing Entities (failure)

This example fails because no NAI subscribers are found.

Request:

```
<readSubscriberNai ent=”subscriberRouting” ns=”dsr” id=”101”>
  <host>tekelec.com</host>
  <user>Kevin.Smith</user>
```

```
  <user>John.Doe</user>
</readSubscriberNai>
```

Response:

```
<readSubscriberNaiResp id=”101”>
  <res error=”2009” affected=”0” description=”nai not found” />
</readSubscriberNaiResp>
```

Message Flow Example Sessions

The following sections contain examples of exchanging messages between the Customer Provisioning System (CPS) and the XML Data Server process on the Active SDS Server on the Primary Provisioning Site.

All scenarios assume that a TCP/IP connection has already been established between the client and SDS. The first column in the tables is the direction that the message is going. The strings displayed in the Message column are the actual ASCII that would flow over the connection, but do not include the 4 byte binary length which is sent before the XML itself.

The actual request and response messages are a series of characters with no extra spaces or new line characters. New lines and extra spaces were added to these examples for readability purposes.

Single Command Transaction

This example shows three request/response pairs that are exchanged between the CPS and SDS. These requests are processed as "single command transactions", which means that each request is immediately committed to the database. This example creates IMSI and MSISDN routing entities.

Table 65: Single Command Transaction (XML)

Message		Description
CPS—>SDS	<updateSubscriber ent="subscriberRouting" ns="dsr" resonly="n"> <imsi>310910421000106</imsi> <imsi>310910421000307</imsi> <imsi>310910421000309</imsi> <msisdn>15634210106</msisdn> <msisdn>15634210107</msisdn> <ltehss>LTE_HSS_2</ltehss> <aaa>AAA_4</aaa> </updateSubscriber>	Request to create 5 stand-alone routing entities - 3 IMSIs and 2 MSISDNs with an LTE HSS and AAA server destinations. **Note:** Request is made to include the original request in the response. Response to create subscriber routing entities - success. Affected rows = 5 (as 5 new entries created for 3 IMSIs and 2 MSISDNs). **Note:** As requested, the original XML request is included in the response.
CPS<—SDS	<updateSubscriberResp> <updateSubscriber ent="subscriberRouting" ns="dsr" resonly="n"> <imsi>310910421000106</imsi> <imsi>310910421000307</imsi> <imsi>310910421000309</imsi> <msisdn>15634210106</msisdn> <msisdn>15634210107</msisdn> <ltehss>LTE_HSS_2</ltehss> <aaa>AAA_4</aaa> </updateSubscriber> <res error="0" affected="5"> </updateSubscriberResp>	
CPS—>SDS	<updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000106</imsi> <msisdn>15634210106</msisdn> <ltehss>LTE_HSS_5</ltehss> </updateSubscriber>	Request to update existing IMSI and MSISDN stand-alone routing entities with a new LTE HSS value. Response to update subscriber routing entities - success. Affected rows = 2 (as 2 entries for an IMSI and MSISDN were updated with new LTE HSS value).
CPS<—SDS	<updateSubscriberResp> <res error="0" affected="2"/> </updateSubscriberResp>	
CPS—>SDS	<updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000102</imsi> <ltehss>BAD_VALUE</ltehss> </updateSubscriber>	Request to create a subscriber routing entitiy with an invalid LTE HSS destination value. Request fails, as the destination does not exist.
CPS<—SDS	<updateSubscriberResp> <res description=”destination not found” error="2006" affected="0"/> </updateSubscriberResp>	

Multiple Commands Transaction Committed

This example issues several requests within one transaction which is then committed successfully.

Table 66: Multiple Commands Transaction Committed Message Flow Example (XML)

Message		Description
CPS—>SDS	`<startTransaction/>`	Request to start a transaction immediately.

CPS<—SDS	`<startTransactionResp>`
	`<res error="0" affected="0"/>`
	`</startTransactionResp>`
		Response start transaction - success.

CPS—>SDS	`<updateSubscriber`
	`ent="subscriberRouting" ns="dsr">`
	`<imsi>310910421000444</imsi>`
	`<msisdn>15634210444</msisdn>`
	`<ltehss>LTE_HSS_1</ltehss>`
	`</updateSubscriber>`
		Request to add new stand-alone IMSI and MSISDN - success

CPS<—SDS	`<updateSubscriberResp>`
	`<res error="0" affected="2”/>`
	`</updateSubscriberResp>`	

CPS—>SDS	`<updateSubscriber`
	`ent="subscriberRouting" ns="dsr">`
	`<imsi>310910421000555</imsi>`
	`<msisdn>15634210555</msisdn>`
	`<ltehss>LTE_HSS_2</ltehss>`
	`</updateSubscriber>`
		Request to update existing stand-alone IMSI and MSISDN - success

CPS<—SDS	`<updateSubscriberResp>`
	`<res error="0" affected="2”/>`
	`</updateSubscriberResp>`	

CPS—>SDS	`<updateSubscriberNai`
	`ent="subscriberRouting" ns="dsr">`
	`<host>operator.com</host>`
	`<user>roger.brown</user>`
	`<ltehss>LTE_HSS_1</ltehss>`
	`</updateSubscriberNai>`
		Request to update an NAI - success.

CPS<—SDS	`<updateSubscriberNaiResp>`
	`<res error="0" affected="1"/>`
	`</updateSubscriberNaiResp>`	

CPS—>SDS	`<commit/>`	Request to commit the transaction.

CPS<—SDS	`<commitResp>`
	`<res error="0" affected="0"/>`
	`</commitResp>`
		Response to commit transaction - success. All updates were successfully performed

Multiple Commands Transaction Rolled Back

This example issues several requests within one transaction which is rolled back.

Table 67: Multiple Commands Transaction Rolled Back Message Flow Example (XML)

Message		Description
CPS—>SDS	`<startTransaction timeout="10"> </startTransaction>`	Request to start a transaction within 10 seconds. Response to start transaction - success.
CPS<—SDS	`<startTransactionResp> <res error="0" affected="0"/> </startTransactionResp>`	
CPS—>SDS	`<updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000777</imsi> <msisdn>15634210777</msisdn> <ltehss>LTE_HSS_7</ltehss> </updateSubscriber>`	Request to update existing stand-alone IMSI and MSISDN - success.
CPS<—SDS	`<updateSubscriberResp> <res error="0" affected="2”/> </updateSubscriberResp>`	
CPS—>SDS	`<updateSubscriberNai ent="subscriberRouting" ns="dsr"> <host>operator.com</host> <user>david.leno</user> <ltehss>LTE_HSS_1</ltehss> </updateSubscriberNai>`	Request to create an NAI - success.
CPS<—SDS	`<updateSubscriberNaiResp> <res error="0" affected="1"/> </updateSubscriberNaiResp>`	
CPS—>SDS	`<rollback/>`	Transaction is rolled back by the client. None of the previous IMSI, MSISDN or NAI entities will be created. Rollback is successful; no creations/updates are made. At this point the client could still have sent commit if they wanted, which would have resulted in the 2 IMSIs, 2 MSISDNs, and 1 NAI being created.
CPS<—SDS	`<rollbackResp> <res error="0" affected="0"/> </rollbackResp>`	

Block Transaction Committed

This example issues several requests within a block transaction. All of the requests succeed; therefore, the transaction is automatically committed.

Table 68: Block Transaction Committed Message Flow Example

Message		Description
CPS—>SDS	<tx> <updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000109</imsi> <ltehss>LTE_HSS_2</ltehss> </updateSubscriber> <updateSubscriber ent="subscriberRouting" ns="dsr"> <msisdn>156342101009</msisdn> <ltehss>LTE_HSS_2</ltehss> </updateSubscriber> <updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000110</imsi> <msisdn>15634210110</msisdn> <ltehss>LTE_HSS_6</ltehss> </updateSubscriber> <updateSubscriber ent="subscriberRouting" ns="dsr"> <ltehss>LTE_HSS_6</ltehss> </updateSubscriber> </tx>	A single request is sent contain 3 different <updateSubscriber> requests for existing stand-alone IMSI or MSISDN routing entities. Response indicates that 3 requests were within the transaction. Each request indicates that 1 row was affected for each, and every request was successful (as error="0" in all response).
CPS<—SDS	<txResp nbreq="3"> <updateSubscriberResp> <res error="0" affected="1”/> </updateSubscriberResp> <updateSubscriberResp> <res error="0" affected="1”/> </updateSubscriberResp> <updateSubscriberResp> <res error="0" affected="2” </updateSubscriberResp> </txResp>	

Block Transaction Rolled Back

This example issues several requests within a block transaction. One of the requests fails; therefore, the transaction is automatically rolled back.

Table 69: Block Transaction Rolled Back Message Flow Example

Message		Description
CPS—>SDS	<tx resonly=”n”> <updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000111</imsi> <ltehss>LTE_HSS_2</ltehss> </updateSubscriber> <updateSubscriber ent="subscriberRouting" ns="dsr"> <msisdn>156342101011</msisdn> <ltehss>LTE_HSS_2</ltehss> </updateSubscriber> <updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000112</imsi> <ltehss>LTE_HSS_99</ltehss> </updateSubscriber> <updateSubscriber ent="subscriberRouting" ns="dsr"> <msisdn>15634210112</msisdn> <ltehss>LTE_HSS_6</ltehss> </updateSubscriber> </tx>	A single request is sent containing 4 different updateSubscriber requests for existing stand-alone IMSI or MSISDN routing entities. The request is made to include each request in the response for the entire transaction (indicated by the resonly="n" attribute). Response to create subscriber routing entities - success. Affected rows = 1 (as 1 NAI entry was updated). The first two requests that were successful, indicate no error and the correct number of affected rows. The third request that fails gives the correct error and no affected rows. The fourth request that has not been executed has an error code indicating NOT_PROCESSED. All requests are rolled back.
CPS<—SDS	<txResp nbreq="4"> <updateSubscriberResp> <updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000111</imsi> <ltehss>LTE_HSS_2</ltehss> </updateSubscriber> <res error="0" affected="1"/> </updateSubscriberResp> <updateSubscriberResp> <updateSubscriber ent="subscriberRouting" ns="dsr"> <msisdn>156342101011</msisdn> <ltehss>LTE_HSS_2</ltehss> </updateSubscriber> <res error="0" affected="1"/> </updateSubscriberResp> <updateSubscriberResp> <updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000112</imsi> <ltehss>LTE_HSS_99</ltehss> </updateSubscriber> <res description= ”destination not found” error="2006" affected="0"/> </updateSubscriberResp> <updateSubscriberResp> <updateSubscriber ent="subscriberRouting" ns="dsr"> <msisdn>15634210112</msisdn> <ltehss>LTE_HSS_6</ltehss> </updateSubscriber> <res error="1" affected="0"/>	

Message		Description
	</updateSubscriberResp> </txResp>	

Appendix A

SDS Response Message Error Codes

Topics:

This section describes the XML/SOAP error codes that are returned by the XDS/SOAP Server.

SDS Response Message Error Codes

XML/SOAP error codes are returned by the XDS/SOAP Server in the error attribute parameter of the `<requestResp>` messages (see *XML Response Messages*) or in the SOAP Response message (see *SOAP Response Messages*). The error parameter of a response message indicates the success or failure of a request.

The complete set of response error codes and their associated values are defined in the following table.

Table 70: SDS Response Message Error Codes

Error Code	Value	Description
SUCCESS	0000	Request was successful.
NOT_PROCESSED	0001	Not processed. The request was within a block transaction, and was not processed due to an error with another request within the same block transaction.
INTERNAL_ERROR	1001	An internal error occurred. Contact Tekelec.
WRITE_UNAVAILABLE	1005	Another client already has a transaction open. This will only be returned to clients who do have write access permissions.
NO_WRITE_PERMISSION	1006	The client making the connection does not have write access permissions.
NO_ACTIVE_TXN	1009	A read or write transaction is not currently open for this connection.
ACTIVE_TXN	1010	A read or write transaction is already open on this connection, or an open transaction was aborted prior to terminating the connection.
INVALID_VALUE	1012	One of the fields in the request has a invalid value.
PARTIAL_SUCCESS	1016	The request has succeeded, but this is one of several responses. This error code is used for indicating status while processing import files.
NO_UPDATES	1017	All of the changes were already in the database.
DURABILITY_TIMEOUT	1024	The update was not made durable in the database within the configured time interval.
BAD_IMPORT_CMD	1028	The command is not supported by the Import operation.
TXN_TOO_BIG	1029	Transaction too big (more than the configured maximum number of requests). The maximum number of requests within a transaction is configured on the SDS GUI. See the *SDS Online Help* for more information.

Error Code	Value	Description
DURABILITY_DEGRADED	1030	The system's transaction durability is degraded and updates will not be accepted until the transaction durability level is restored. Contact the Tekelec Customer Care Center. The transaction durability level can be temporarily adjusted until the problem is resolved. The associated request should be resent after durability is restored or manually adjusted since it has not been committed or is no longer committed to the database due to a rollback when the system durability became degraded.
DB_EXCEPTION	1031	An unexpected exception was thrown during the database commit. The entire transaction was rolled back to ensure predictable behavior. Contact the Tekelec Customer Care Center.
PROV_PROHIBITED	1051	Database access has been manually disabled.
INV_REQUEST_NAME	2001	The XML request name does not indicate a valid request.
INVALID_XML	2002	The request does not contain a valid XML data structure and cannot be parsed.
MISSING_PARAMETER	2003	A mandatory parameter is missing.
INVALID_MULT_INST	2004	Multiple instances of a parameter that only allows a single instance has occurred.
UNKNOWN_PARAM_NAME	2005	The specified parameter name is unknown for this request.
DEST_NOT_FOUND	2006	The specified destination name does not exist.
IMSI_NOT_FOUND	2007	The specified IMSI does not exist.
MSISDN_NOT_FOUND	2008	The specified MSISDN does not exist.
NAI_NOT_FOUND	2009	The specified NAI (host/user) does not exist.
NAI_HOST_NOT_FOUND	2010	The specified host name does not exist.
TXN_TIMED_OUT	2011	The Transaction that was in progress has timed out, and automatically rolled back.
TOO_MANY_ADDR	2012	Too many IMSI/MSISDN routing entities were specified in the request
NO_DEST_VAL	2013	At least one destination value must be specified.
NO_ADDR_VAL	2014	No IMSI/MSISDN value and no Account ID value was supplied.
TOO_MANY_NAI	2015	Too many NAI routing entities were specified.
NO_NAI_VAL	2016	No NAI value was supplied.
DEST_TYPE_MISMATCH	2017	Destination has a different destination type than the desired destination type.

Error Code	Value	Description
INVALID_ARG	2018	The arguments are not valid. Each individual value is valid, but the combination of specified values and database values is not allowed.
INSTANCE_LIMIT	2019	Operation would exceed the maximum number of allowed records in the table.
INV_REQ_IN_BLOCK_TX	2020	An invalid request has been sent in a block transaction (e.g. startTransaction, commit, or rollback).
INV_REQ_IN_NORMAL_TX	2021	An invalid request has been sent in a normal transaction (e.g. a block transaction).
SUBSCRIBER_NOT_FOUND	2022	The specified subscriber does not exist.
MULTIPLE_SUBSCRIBERS	2023	The specified parameters refer to multiple subscribers.
SUBSCRIBER_TOO_BIG	2024	The resulting subscriber would exceed the 6 IMSI or 6 MSISDN limit.
ACCTID_UPDATE_PROHIBITED	2025	An attempt was made to change an accountId without specifying the **<deleteAccountId>** tag.
ROUTE_TYPE_MISMATCH	2026	Standalone and subscriber routes are not allowed in same command.
DEL_ROUTE_NOT_PERMITTED	2027	Cannot delete the last route from a subscriber.
NO_ROUTES_SPECIFIED	2028	At least one MSISDN or IMSI must be specified.
ROUTE_DEST_MISMATCH	2029	Specified routes have different destinations.

Appendix B

XML/SOAP Interface System Variables

Topics:

This section describes the XML/SOAP interfaces that have a set of system variables that affect the operation as it runs.

XML/SOAP Interface System Variables

The XML/SOAP Interfaces have a set of system variables that affect its operation as it runs. XML/SOAP Interface System variables (shown below in *Table 71: XML/SOAP Interface System Variables*) can be set via the SDS GUI and can be changed at runtime to effect dynamic server reconfiguration. See the *SDS Online Help* for more information.

Table 71: XML/SOAP Interface System Variables

Parameter	Description
XML Interface Port	XML Interface TCP (unsecure) Listening Port. The TCP listening port can be disabled by setting it to 0. NOTE: Changes to the TCP listening port do not take affect until the 'xds' process is restarted. Also, you must specify a different port than the SOAP interface. DEFAULT = 5875; RANGE = 0-65535
SOAP Interface Port	SOAP Interface TCP Listening Port. The TCP listening port can be disabled by setting it to 0. NOTE: Changes to the TCP listening port do not take affect until the 'xds' process is restarted. Also, you must specify a different port than the XML interface. DEFAULT = 5876 (when SOAP Secure Mode is set to UNSECURE) or 5877 (when SOAP Secure Mode is set to SECURE) RANGE = 0-65535
XML Interface Idle Timeout	The maximum time (in seconds) that an open XML connection will remain active without a request being sent, before the connection is dropped. DEFAULT = 1200; RANGE = 1-86400
SOAP Interface Idle Timeout	The maximum time (in seconds) that an open SOAP connection will remain active without a request being sent, before the connection is dropped. DEFAULT = 1200; RANGE = 1-86400
Maximum XML Connections	Maximum number of simultaneous XML Interface client connections. DEFAULT = 120; RANGE = 1-120
Maximum SOAP Connections	Maximum number of simultaneous SOAP Interface client connections. DEFAULT = 120; RANGE = 1-120
SOAP Secure Mode	Whether the SOAP Interface operates in secure mode (using SSL), or unsecure mode (plain text). NOTE: Changes to the SOAP Secure Mode do not take affect until the 'xds' process is restarted. DEFAULT = UNSECURE

Parameter	Description
Allow Connections*	Whether or not to allow incoming connections on the XML/SOAP Interface. DEFAULT = ALLOWED
Max Transaction Size*	Maximum number of database manipulation commands per transaction. DEFAULT = 50; RANGE = 10-1000
Maximum Transaction Lifetime	The maximum time (in seconds) that a transaction can remain open before automatically being rolled back if a commit or rollback is not explicitly performed. Timeout can be disabled by setting to 0. DEFAULT=60; RANGE = 0-3600
Remote Import Mode*	Whether updates are allowed (Non-Blocking) or not allowed (Blocking) on all XDS connections while the remote import operation is in progress. In blocking mode, XML and SOAP provisioning requests will be rejected if a bulk import is in operation. In non-blocking mode, XML and SOAP provisioning requests will be allowed as normal. DEFAULT = NON-BLOCKING
Export Mode*	Whether updates are allowed (Non-Blocking) or not allowed (Blocking) on all XDS connections while the export operation is in progress. In blocking mode, XML and SOAP provisioning requests will be rejected if a bulk export is in operation. In non-blocking mode, XML and SOAP provisioning requests will be allowed as normal. DEFAULT = NON-BLOCKING
Transaction Durability Timeout*	The amount of time (in seconds) allowed between a transaction being committed and it becoming durable. If Transaction Durability Timeout lapse, DURABILITY_TIMEOUT response is sent to the originating client. The associated request should be resent to ensure that the request was committed. DEFAULT = 5; RANGE = 2-3600

Note: Parameters labeled with a "*" are existing system variables defined and used by other components of UDR.

Appendix C

Database Object Model

Topics:

This section describes the database object model and shows all tables associated with SDS provisioning.

Database Object Model

Figure 3: SDS Provisioning Database Object Model shows the database object model for subscriber-related data. All of the tables are available to the user.

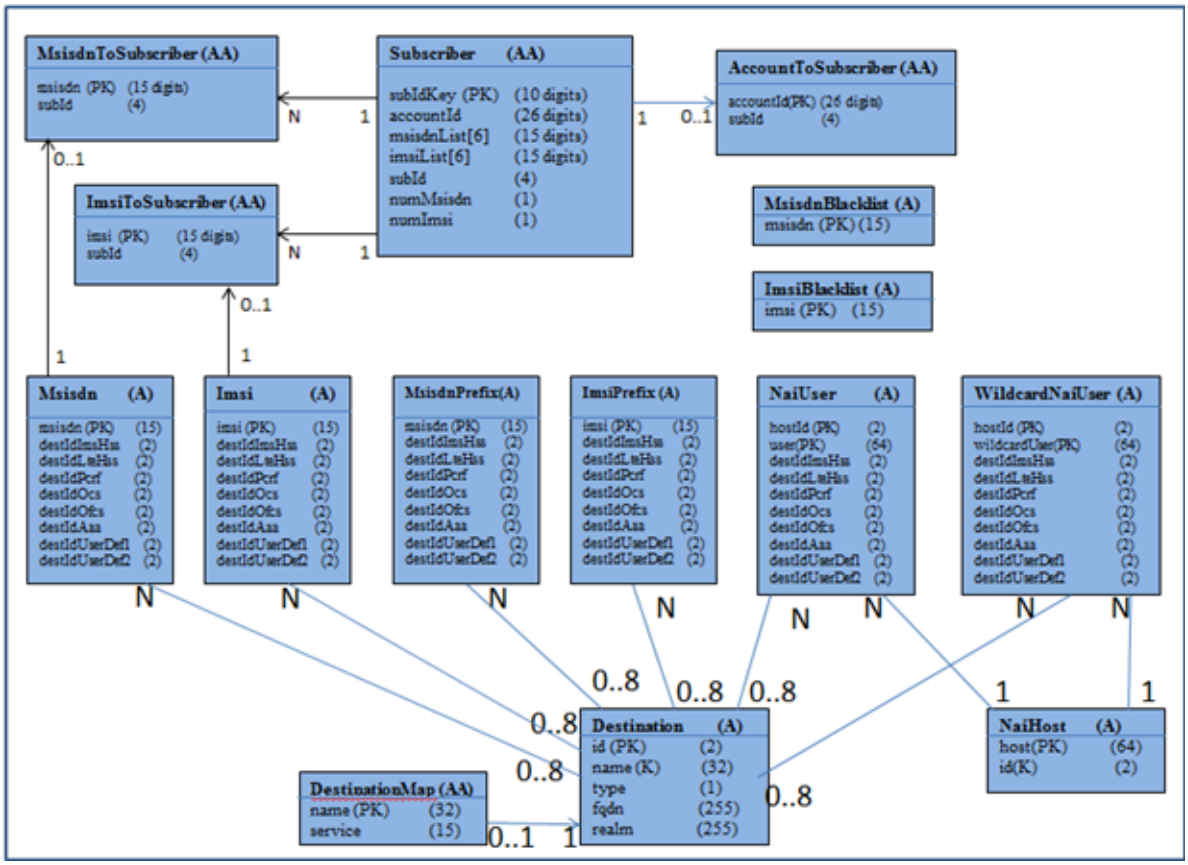

Figure 3: SDS Provisioning Database Object Model

MsisdnBlacklist

Table 72: MsisdnBlacklist Table Attributes

Attribute	Description
msisdn	A unique string of 8-15 decimal digits.

ImsiBlacklist

Table 73: ImsiBlacklist Table Attributes

Attribute	Description
imsi	A unique string of 10-15 decimal digits.

Msisdn

Table 74: Msisdn Table Attributes

Attribute	Description
msisdn	A unique string of 8-15 decimal digits.
destIdImsHss	Index to an existing IMS HSS Destination record (with type= imsHss) or 0 (for none).
destIdLteHss	Index to an existing LTE HSS Destination record (with type= lteHss) or 0 (for none).
destIdPcrf	Index to an existing PCRF Destination record (with type= pcrf) or 0 (for none).
destIdOcs	Index to an existing OCS Destination record (with type= ocs) or 0 (for none).
destIdOfcs	Index to an existing OFCS Destination record (with type= ofcs) or 0 (for none).
destIdAaa	Index to an existing AAA Destination record (with type= aaa) or 0 (for none).
destIdUserDef1	Index to an existing UserDef1 Destination record (with type= userDef1) or 0 (for none).
destIdUserDef2	Index to an existing UserDef2 Destination record (with type= userDef2) or 0 (for none).

Imsi

Table 75: Imsi Table Attributes

Attribute	Description
imsi	A unique string of 8-15 decimal digits.
destIdImsHss	Index to an existing IMS HSS Destination record (with type= imsHss) or 0 (for none).
destIdLteHss	Index to an existing LTE HSS Destination record (with type= lteHss) or 0 (for none).
destIdPcrf	Index to an existing PCRF Destination record (with type= pcrf) or 0 (for none).
destIdOcs	Index to an existing OCS Destination record (with type= ocs) or 0 (for none).
destIdOfcs	Index to an existing OFCS Destination record (with type= ofcs) or 0 (for none).
destIdAaa	Index to an existing AAA Destination record (with type= aaa) or 0 (for none).

Attribute	Description
destIdUserDef1	Index to an existing UserDef1 Destination record (with type= userDef1) or 0 (for none).
destIdUserDef2	Index to an existing UserDef2 Destination record (with type= userDef2) or 0 (for none).

MsisdnPrefix

Table 76: MsisdnPrefix Table Attributes

Attribute	Description
msisdnPrefix	A unique string of 1-15 decimal digits. Can have overlapping prefix values.
destIdImsHss	Index to an existing IMS HSS Destination record (with type= imsHss) or 0 (for none).
destIdLteHss	Index to an existing LTE HSS Destination record (with type= lteHss) or 0 (for none).
destIdPcrf	Index to an existing PCRF Destination record (with type= pcrf) or 0 (for none).
destIdOcs	Index to an existing OCS Destination record (with type= ocs) or 0 (for none).
destIdOfcs	Index to an existing OFCS Destination record (with type= ofcs) or 0 (for none).
destIdAaa	Index to an existing AAA Destination record (with type= aaa) or 0 (for none).
destIdUserDef1	Index to an existing UserDef1 Destination record (with type= userDef1) or 0 (for none).
destIdUserDef2I	ndex to an existing UserDef2 Destination record (with type= userDef2) or 0 (for none).

ImsiPrefix

Table 77: ImsiPrefix Table Attributes

Attribute	Description
imsiPrefix	A unique string of 1-15 decimal digits. Can have overlapping prefix values.
destIdImsHss	Index to an existing IMS HSS Destination record (with type= imsHss) or 0 (for none).
destIdLteHss	Index to an existing LTE HSS Destination record (with type= lteHss) or 0 (for none).

Attribute	Description
destIdPcrf	Index to an existing PCRF Destination record (with type= pcrf) or 0 (for none).
destIdOcs	Index to an existing OCS Destination record (with type= ocs) or 0 (for none).
destIdOfcs	Index to an existing OFCS Destination record (with type= ofcs) or 0 (for none).
destIdAaa	Index to an existing AAA Destination record (with type= aaa) or 0 (for none).
destIdUserDef1	Index to an existing UserDef1 Destination record (with type= userDef1) or 0 (for none).
destIdUserDef2	Index to an existing UserDef2 Destination record (with type= userDef2) or 0 (for none).

NaiUser

Table 78: NaiUser Table Attributes

Attribute	Description
user	A string of 1-64 characters for the NAI User Name.
destIdImsHss	Index to an existing IMS HSS Destination record (with type= imsHss) or 0 (for none).
destIdLteHss	Index to an existing LTE HSS Destination record (with type= lteHss) or 0 (for none).
destIdPcrf	Index to an existing PCRF Destination record (with type= pcrf) or 0 (for none).
destIdOcs	Index to an existing OCS Destination record (with type= ocs) or 0 (for none).
destIdOfcs	Index to an existing OFCS Destination record (with type= ofcs) or 0 (for none).
destIdAaa	Index to an existing AAA Destination record (with type= aaa) or 0 (for none).
destIdUserDef1	Index to an existing UserDef1 Destination record (with type= userDef1) or 0 (for none).
destIdUserDef2	Index to an existing UserDef2 Destination record (with type= userDef2) or 0 (for none).

WildcardNaiUser

Table 79: WildcardNaiUser Table Attributes

Attribute	Description
wildcardUser	A string of 1-64 characters for the wild-carded NAI User Name.
hostId	Index to an existing NAI Host record. The wildcardUser/hostId combination must be unique.
destIdImsHss	Index to an existing IMS HSS Destination record (with type= imsHss) or 0 (for none).
destIdLteHss	Index to an existing LTE HSS Destination record (with type= lteHss) or 0 (for none).
destIdPcrf	Index to an existing PCRF Destination record (with type= pcrf) or 0 (for none).
destIdOcs	Index to an existing OCS Destination record (with type= ocs) or 0 (for none).
destIdOfcs	Index to an existing OFCS Destination record (with type= ofcs) or 0 (for none).
destIdAaa	Index to an existing AAA Destination record (with type= aaa) or 0 (for none).
destIdUserDef1	Index to an existing UserDef1 Destination record (with type= userDef1) or 0 (for none).
destIdUserDef2	Index to an existing UserDef2 Destination record (with type= userDef2) or 0 (for none).

Destination

Table 80: Destination Table Attributes

Attribute	Description
name	A unique string of 1-32 characters to identify the Destination.
id	A unique, generated number used to identify a Destination record
type	Destination type
fqdn	A 1-255 character string for the Diameter FQDN for the Destination. The value can be null.
realm	A 1-255 character string for the Diameter Realm for the Destination. The value can be null.

destinationMap

Table 81: DestinationMap Table Attributes

Attribute	Description
name	A unique string of 1-32 characters to identify an existing Destination record.
service	A string of 8-15 decimal digits that contains the E.164 node address of an HLR Router.

NaiHost

Table 82: NaiHost Table Attributes

Attribute	Description
host	A unique string of 1-64 characters for the NAI Host Name.
id	A unique, generated number used to identify a NaiHost record.

Subscriber

Table 83: Subscriber Table Attributes

Attribute	Description
subIdKey	A unique string of 1-10 decimal digits. This string is based on the numeric subId value.
subId	A unique, internal, numeric Subscriber ID. This number is assigned by SDS software.
accountId	An optional unique string of 6-26 decimal digits. This value is assigned by the customer.
msisdnList[6]	A list of MSISDN values for the Subscriber. Each MSISDN value must be 8-15 decimal digits.
imsiList[6]	A list of IMSI values for the Subscriber. Each IMSI value must be 10-15 decimal digits.
numMsisdn	Number of MSISDN values defined for the Subscriber.
numImsi	Number of IMSI values defined for the Subscriber.

AccountToSubscriber

Table 84: AccountToSubscriber Table Attributes

Attribute	Description
accountId	An optional unique string of 6-26 decimal digits. This value is assigned by the customer.
subId	An internal, numeric Subscriber ID. This number is assigned by SDS software.

MsisdnToSubscriber

Table 85: MsisdnToSubscriber Table Attributes

Attribute	Description
msisdn	A unique string of 8-15 decimal digits
subId	An internal, numeric Subscriber ID. This number is assigned by SDS software.

ImsiToSubscriber

Table 86: ImsiToSubscriber Table Attributes

Attribute	Description
imsi	A unique string of 10-15 decimal digits.
subId	An internal, numeric Subscriber ID. This number is assigned by SDS software.

Appendix D

Copyright, notice, trademarks, and patents

This section provides important information about copyrights, notices, trademarks, and patents associated with this product.

Glossary

A

ACID	Atomicity, Consistency, Isolation and Durability

C

CA	Canada (NPAC Region) Conditioning Action NPP CAs indicate what digit conditioning actions to execute when processing a digit string. Certificate Authority: An entity that issues digital certificates

F

FTP	File Transfer Protocol A client-server protocol that allows a user on one computer to transfer files to and from another computer over a TCP/IP network. Feature Test Plan

G

GUI	Graphical User Interface The term given to that set of items and facilities which provide the user with a graphic means for manipulating screen data rather than being limited to character based commands.

H

HA	High Availability High Availability refers to a system or component that operates on a continuous basis by utilizing

H

redundant connectivity, thereby circumventing unplanned outages.

K

KPI — Key Performance Indicator

N

NOC — Network Operations Center

O

OAM — Operations, Administration, and Maintenance

The application that operates the Maintenance and Administration Subsystem which controls the operation of many products.

P

pSBR — Policy SBR

S

SDS — Subscriber Database Server

Subscriber Database Server (SDS) provides the central provisioning of the Full-Address Based Resolution (FABR) data. The SDS, which is deployed geo-redundantly at a Primary and Disaster recovery site, connects with the Query Server and the Data Processor System Operations, Administration, and Maintenance (DP SOAM) servers at each Diameter Signaling Router (DSR) site or a standalone DP site to replicate and recover provisioned data to the associated components.

SNMP — Simple Network Management Protocol.

S

	An industry-wide standard protocol used for network management. The SNMP agent maintains data variables that represent aspects of the network. These variables are called managed objects and are stored in a management information base (MIB). The SNMP protocol arranges managed objects into groups.
SOAM	System Operations, Administration, and Maintenance Site Operations, Administration, and Maintenance
SSL	Secure Socket Layer (SSL) is an industry standard protocol for clients needing to establish secure (TCP-based) SSL-enabled network connections

U

UDR	User-Data-Request - A user-identity and service indication sent by a Diameter client to a Diameter server in order to request user data. User Data Repository - A logical entity containing user data

V

VIP	Virtual IP Address Virtual IP is a layer-3 concept employed to provide HA at a host level. A VIP enables two or more IP hosts to operate in an active/standby HA manner. From the perspective of the IP network, these IP hosts appear as a single host.