Subscriber Database Server

XML/SOAP Provisioning Interface

910-6371-001 Revision B
August 2012

Tekelec

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 describes the XML and SOAP Provisioning Applications to be used by local and remote provisioning client applications to administer the Provisioning Database of the SDS system. Remote applications include independent information systems supplied and maintained by network operators. Through XML or SOAP interfaces, independent information systems can add, change, delete, or retrieve information about any IMSI/MSISDN/NAI association.

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.

Manual Organization

This document is organized into the following chapters:

- *Overview* contains general information about the SDS documentation, the organization of this manual, and how to get technical assistance.
- *System Architecture* gives and 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.
- *XML Message Definitions* describes XML requests and responses syntax and parameters.
- *SOAP Operations Definitions* describes the SOAP operations 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.
- *XML Schema Definition Files (XSD)* describes the XML requests, responses, and data types.
- *SOAP WSDL Files* describes the SOAP requests and responses that are documented in the WSDL files.

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

🛑	**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*.)

Customer Care Center

The Tekelec Customer Care Center is your initial point of contact for all product support needs. A representative takes your call or email, creates a Customer Service Request (CSR) and directs your requests to the Tekelec Technical Assistance Center (TAC). Each CSR includes an individual tracking number. Together with TAC Engineers, the representative will help you resolve your request.

The Customer Care Center is available 24 hours a day, 7 days a week, 365 days a year, and is linked to TAC Engineers around the globe.

Tekelec TAC Engineers are available to provide solutions to your technical questions and issues 7 days a week, 24 hours a day. After a CSR is issued, the TAC Engineer determines the classification of the trouble. If a critical problem exists, emergency procedures are initiated. If the problem is not critical, normal support procedures apply. A primary Technical Engineer is assigned to work on the CSR and provide a solution to the problem. The CSR is closed when the problem is resolved.

Tekelec Technical Assistance Centers are located around the globe in the following locations:

Tekelec - Global

Email (All Regions): support@tekelec.com

- **USA and Canada**

  Phone:

  1-888-FOR-TKLC or 1-888-367-8552 (toll-free, within continental USA and Canada)

  1-919-460-2150 (outside continental USA and Canada)

  TAC Regional Support Office Hours:

  8:00 a.m. through 5:00 p.m. (GMT minus 5 hours), Monday through Friday, excluding holidays
- **Caribbean and Latin America (CALA)**

  Phone:

  USA access code +1-800-658-5454, then 1-888-FOR-TKLC or 1-888-367-8552 (toll-free)

TAC Regional Support Office Hours (except Brazil):

10:00 a.m. through 7:00 p.m. (GMT minus 6 hours), Monday through Friday, excluding holidays

- **Argentina**

  Phone:

  0-800-555-5246 (toll-free)
- **Brazil**

  Phone:

  0-800-891-4341 (toll-free)

  TAC Regional Support Office Hours:

  8:00 a.m. through 5:48 p.m. (GMT minus 3 hours), Monday through Friday, excluding holidays
- **Chile**

  Phone:

  1230-020-555-5468
- **Colombia**

  Phone:

  01-800-912-0537
- **Dominican Republic**

  Phone:

  1-888-367-8552
- **Mexico**

  Phone:

  001-888-367-8552
- **Peru**

  Phone:

  0800-53-087
- **Puerto Rico**

  Phone:

  1-888-367-8552 (1-888-FOR-TKLC)
- **Venezuela**

  Phone:

  0800-176-6497

- **Europe, Middle East, and Africa**

  Regional Office Hours:

  8:30 a.m. through 5:00 p.m. (GMT), Monday through Friday, excluding holidays

  - **Signaling**

Phone:

+44 1784 467 804 (within UK)

- **Software Solutions**

  Phone:

  +33 3 89 33 54 00

- **Asia**
  - **India**

    Phone:

    +91 124 436 8552 or +91 124 436 8553

    TAC Regional Support Office Hours:

    10:00 a.m. through 7:00 p.m. (GMT plus 5 1/2 hours), Monday through Saturday, excluding holidays
  - **Singapore**

    Phone:

    +65 6796 2288

    TAC Regional Support Office Hours:

    9:00 a.m. through 6:00 p.m. (GMT plus 8 hours), Monday through Friday, excluding holidays

Emergency Response

In the event of a critical service situation, emergency response is offered by the Tekelec Customer Care Center 24 hours a day, 7 days a week. 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 the Tekelec Customer Care Center.

Related Publications

For information about additional publications that are related to this document, refer to the *Related Publications* document. The *Related Publications* document is published as a part of the *Release Documentation* and is also published as a separate document on the Tekelec Customer Support Site.

Documentation Availability, Packaging, and Updates

Tekelec provides documentation with each system and in accordance with contractual agreements. For General Availability (GA) releases, Tekelec publishes a complete EAGLE 5 ISS documentation set. For Limited Availability (LA) releases, Tekelec may publish a documentation subset tailored to specific feature content or hardware requirements. Documentation Bulletins announce a new or updated release.

The Tekelec EAGLE 5 ISS documentation set is released on an optical disc. This format allows for easy searches through all parts of the documentation set.

The electronic file of each manual is also available from the *Tekelec Customer Support* site. This site allows for 24-hour access to the most up-to-date documentation, including the latest versions of Feature Notices.

Printed documentation is available for GA releases on request only and with a lead time of six weeks. The printed documentation set includes pocket guides for commands and alarms. Pocket guides may also be ordered separately. Exceptions to printed documentation are:

- Hardware or Installation manuals are printed without the linked attachments found in the electronic version of the manuals.
- The Release Notice is available only on the Customer Support site.

Note: Customers may print a reasonable number of each manual for their own use.

Documentation is updated when significant changes are made that affect system operation. Updates resulting from Severity 1 and 2 Problem Reports (PRs) are made to existing manuals. Other changes are included in the documentation for the next scheduled release. Updates are made by re-issuing an electronic file to the customer support site. Customers with printed documentation should contact their Sales Representative for an addendum. Occasionally, changes are communicated first with a Documentation Bulletin to provide customers with an advanced notice of the issue until officially released in the documentation. Documentation Bulletins are posted on the Customer Support site and can be viewed per product and release.

Locate Product Documentation on the Customer Support Site

Access to Tekelec's Customer Support site is restricted to current Tekelec customers only. This section describes how to log into the Tekelec Customer Support site and locate a document. Viewing the document requires Adobe Acrobat Reader, which can be downloaded at www.adobe.com.

1. Log into the *Tekelec Customer Support* site.

Note: If you have not registered for this new site, click the **Register Here** link. Have your customer number available. The response time for registration requests is 24 to 48 hours.

2. Click the **Product Support** tab.
3. Use the Search field to locate a document by its part number, release number, document name, or document type. The Search field accepts both full and partial entries.
4. Click a subject folder to browse through a list of related files.
5. To download a file to your location, right-click the file name and select **Save Target As**.

Chapter 2

System Architecture

Topics:

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

SDM Overview

Tekelec's SDM product family allows customers to consolidate and manage their cross-domain subscriber data - location, network authentication, access preferences, services, identities and presence - as a single logical profile. Data is stored in the solution's back-end database, which supports multiple front-end applications. With the power to integrate data from 2G, 3G, 4G and Value Added Services (VAS) sources under a single subscriber identity, customers can enhance, personalize and rapidly deploy multiple revenue generating services.

Architecture Overview

The following diagram gives an overview of the SDS.

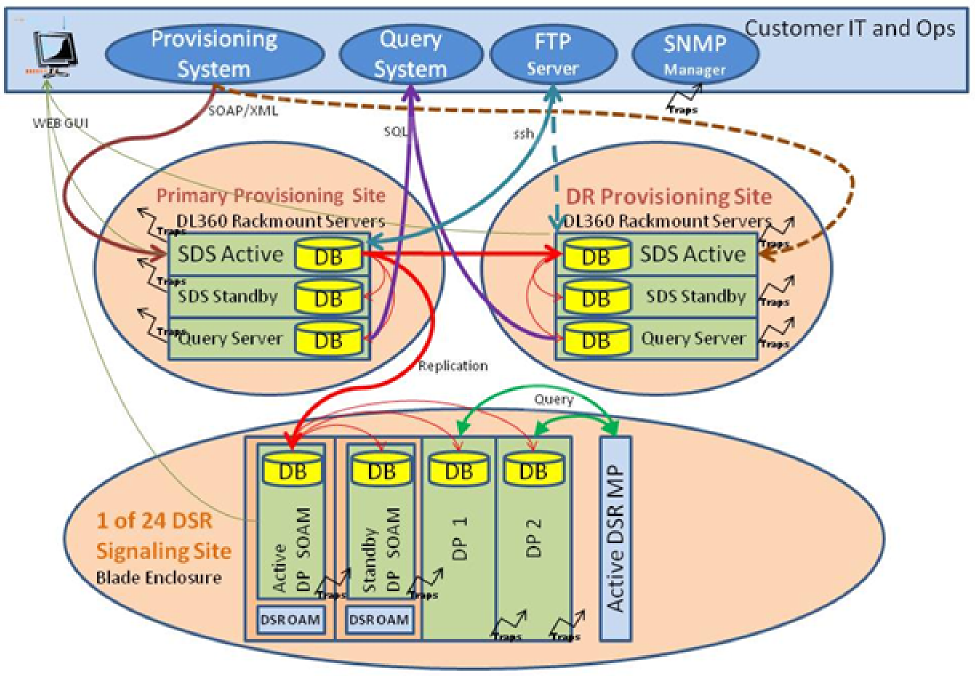


Figure 1: SDS Architecture Overview

The XML Data Server (XDS) and SOAP Server run within the same process, on every active SDS server.

The Client Provisioning Systems connect using the Primary SDS's Virtual IP (VIP). In the event of the failure of the active SDS server, the standby SDS server will be activated, and the VIP moved over to that server.

In the event of a failure of the primary provisioning site, the disaster recovery site will become active. The Client Provisioning Systems must manually switch over to use the Primary SDS's VIP to the Disaster Recovery Site's SDS VIP.

XML Data Server (XDS)

The process containing the XML Data Server runs on every active SDS server. The XDS is responsible for:

- Accepting and authorizing TCP/XML provisioning client connections
- Processing and responding to XML requests received from provisioning clients
- Updating and maintaining the Provisioning Database (only on the Active SDS server of the Primary SDS site)

Note: The XML Data Server and SOAP Server run within the same process.

SOAP Server

The process containing the SOAP Server runs on every active SDS server. The SOAP Server can be configured to operate in unsecure mode (where the SOAP requests are sent in clear text), or secure mode (where an SSL connection is used). The SOAP Server is responsible for:

- Accepting and authorizing HTTP(S)/SOAP provisioning client connections
- Processing and responding to SOAP requests received from provisioning clients
- Updating and maintaining the Provisioning Database (only on the Active SDS server of the Primary SDS site)

Note: The XML Data Server and SOAP Server run within the same process.

Provisioning Clients

Provisioning Clients establish TCP/IP connections to either the XDS or SOAP Server running on the Active SDS server using the Primary SDS's VIP. Customer Provisioning Systems (CPS) use XML or SOAP to send requests to manipulate and query data in the Provisioning Database.

Provisioning clients will need to re-establish connections with the XDS or SOAP Server using the Primary SDS's VIP upon switchover from the Primary's Active to its Standby SDS server and redirect connections to the Secondary's VIP upon switchover from the Primary SDS site to the Secondary 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 shop drop the connection and re-establish it before trying again. Note: by dropping the connection, any transaction that is in progress on that connection will be automatically rolled back. Consequently, the entire transaction should be started and resent again.

Provisioning clients are expected to re-send XML/SOAP requests for those database manipulation requests that resulted in a temporary error or for which no responses were received. The list of permanent/temporary error codes is described in *SDS Response Message Error Codes*.

System Provisioning

The SDS must be provision with routing destinations and valid NAI host values before routing entites can be provisioned. This configuration can be either bulk loaded, or manually provisioned using the GUI, as described in the SDS GUI.

Destinations

Destination names can be configured for the following destination types :

- IMS HSS
- LTE HSS
- PCRF
- OCS
- OFCS
- PCRF
- AAA
- User Defined #1
- User Defined #2

Each destination is configured with:

- FQDN (Fully Qualified Domain Name)
- Realm

NAI Hosts

Host names are the realm part of an NAI, and must also be configured.

Chapter 3

Interface Description

Topics:

This chapter provides an overview of the interface provided by the XML Data Server (XDS) and the SOAP Server at a high level.

XML Based

The XDS uses an XML based protocol, in which a client communicates with the XDS 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 XDS.

Each and every XML request and response message (as described in *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 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 example illustrates an update subscriber request sent from an XDS client to the XDS.

```
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 (as described in *XML Message Definitions* 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 example illustrates an update subscriber response message string sent from XDS to the XDS 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>
```

An XML based interface provides the following benefits:

- It makes it simple to send and receive the messages from any language that has TCP/socket capability.
- User readable messages facilitate debugging errors in messages.
- Messages can easily be stored in a request log for review or replayed later.

TCP/IP Based

XML messages are sent across a TCP/IP connection between a provisioning client and the XDS/SOAP Server.

XML Provisioning Client

An XML provisioning client application is responsible for:

- Establishing a TCP/IP connection with the XSD using the Primary SDS's VIP and the XDS's well-known listening port designated for XDS connections (configurable by the SDS GUI, see *XML/SOAP Interface System Variables*).

  Note: There is no secure port available.

- Creating and sending XML requests (as specified in *XML Message Definitions*) to the XDS.
- Receiving and processing XML responses (as specified in *XML Message Definitions*) received from the XDS.
- Detecting and handling connection errors. It is recommended that the TCP keep-alive interval on the TCP/IP connection be set such that a disconnection problem is promptly detected and reported.

Whether or not XML connections are allowed (interface enabled/disabled) is controlled by the `Connections Allowed` system option (see *XML/SOAP Interface System Variables*) configurable by the SDS GUI. If a connection attempt is made while connections are not allowed, the connection is rejected by the XDS. All active connections are immediately disconnected when the `Connections Allowed` system-wide option is set to disabled.

The number of XDS connections allowed at any given time is controlled by the `Max XML Interface Connections` system option (see *XML/SOAP Interface System Variables*) configurable by the SDS GUI. If an attempt is made to connect more than the number of XDS connections allowed, the connection is rejected by the XDS.

SOAP Provisioning Client

A SOAP provisioning client application is responsible for:

- Establishing a TCP/IP connection with the SOAP Server using the Primary SDS's VIP and the SOAP Servers well-known listening port designated for SOAP connections (configurable by the SDS GUI, see *XML/SOAP Interface System Variables*).

  Note: the SOAP Server can be configured to operate in either secure (using SSL), or unsecure (clear text) mode (configurable by the SDS GUI, see *XML/SOAP Interface System Variables*).

- Creating and sending SOAP request messages (as specified in *SOAP Operations Definitions*) to the SOAP Server.
- Receiving and processing SOAP response messages (as specified in *SOAP Operations Definitions*) received from the SOAP Server.
- Detecting and handling connection errors. It is recommended that the TCP keep-alive interval on the TCP/IP connection be set such that a disconnection problem is promptly detected and reported.

Whether or not SOAP connections are allowed (interface enabled/disabled) is controlled by the `Connections Allowed` system option (see *XML/SOAP Interface System Variables*. If a connection attempt is made while connections are not allowed, the connection is rejected by the SOAP Server. All active connections are immediately disconnected when the `Connections Allowed` system-wide option is set to disabled.

The number of remote SOAP connections allowed at any given time is controlled by the `Max SOAP Interface Connections` system option (see *XML/SOAP Interface System Variables*) configurable by the SDS GUI If an attempt is made to connect more than the number of SOAP connections allowed, the connection is rejected by the SOAP Server.

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 from which clients are authorized to establish a TCP/IP connection from. Each IP address on the list has either read-only or read/write permissions. Any connect request coming from an IP address that is not on the list is denied (connection is immediately closed). All active connections established from an IP address which is removed from the Authorized IP list are immediately closed. The list is administered using the SDS GUI.

Secure Connection Using SSLv3

The SOAP Server supports secure (encrypted) connections between provisioning clients and the SOAP Server using Secure Sockets Layer version 3 (SSLv3) protocol implemented using OpenSSL based on SSLeay library developed by Eric A. Young and Tim J. Hudson.

SSL is an industry standard protocol for clients needing to establish secure (TCP-based) SSL-enabled network connections. SSL provides data confidentiality, data integrity, and server and client authentication based on digital certificates that comply with X.509v3 standard and public/private key pairs. These services are used to stop a wide variety of network attacks including: Snooping, Tampering, Spoofing, Hijacking, and Capture-replay.

The following capabilities of SSL 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 has 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 has been signed by a trusted certificate authority (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. Message integrity provides a means to determine if the data has been tampered with since it was sent by the partner application.

Depending upon which mode the SOAP Server is configured to operate in (secure/unsecure), provisioning clients can connect using unsecure or secure connections to the SOAP Server's well-known TCP/SSL listening port (configurable via SDS GUI).

Note: An SSL-enabled connection is slower than an unsecure TCP/IP connection. This is a direct result of providing adequate security. On an SSL-enabled connection, more data is transferred than normal. Data is transmitted in packets, which contain information required by the SSL protocol as well as any padding required by the cipher that is in use. There is also the overhead of encryption and decryption for each read and write performed on the connection.

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 a 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 2: 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 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 a 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 3: 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
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 open 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 mentioned above may be specified by the user in the start transaction/update/delete request, and can be any value 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. Of course, if any user's time out period expires, their 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

1. If multiple clients simultaneously issues requests, each request is queued and processed in the order in which it was received on a per connection basis. The client is not required to wait for a response from one request before issuing another.
2. Incoming requests, whether multiple requests from a single client or requests from multiple clients, 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 in which they are received, but this is not always guaranteed. A client could send a number of valid update requests, which are performed, and executed in the order they are received. If the client were to then send an invalid request (such as if the XML could not be parsed), this would be responded to immediately, potentially before the any/some/all of the previous requests have been responded to.

Syncronous/Asyncronous Mode

1. 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.
2. If a client wishes to send a request before waiting for the response to the previous one (i.e. work in ***asyncronous*** mode), then the client must populate the "**id**" attribute in the request with a transaction id value that will also 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 XDS will simply return the value passed in the response.
3. If a client wishes to send a single request, and then wait for the response before sending another one (i.e. work in ***syncronous*** mode), then there is technically no need for the client to populate the "**id**" attribute in the request, as the response will always be for the request last sent (although they can do so if they wish - it will be still passed back in the response just as in asyncronous mode).

Transaction Oriented

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

Transaction Modes

The XML Interface supports the following database transactions modes:

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

The SOAP Interface supports the following database transactions 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/>`

Note that all requests within a transaction must be sent on the same TCP/IP connection, whether the interface is XML/TCP or SOAP. If the TCP/IP connection is disconnected when a transaction is in progress, the transaction will be automatically rolled back.

The maximum number of requests that can be performed within a transaction are configured by the `Maximum Transaction Size` system variable, as described in *XML/SOAP Interface System Variables*. Requests are counted within a transaction as they are sent, so if the maximum number of requests was set to 50, the 51st (and subsequent) requests sent would fail with a `TXN_TOO_BIG` error. The transaction is not automatically committed or rolled back. It is the responsibility of the client to commit/rollback the first 50 requests.

Once a transaction has been started, it must be commited or rolled back within a configured period as configured by the `Maximum Transaction Lifetime` system variable, as described in *XML/SOAP Interface System Variables*. If a transaction is started, and not completed within this time period, it is automatically rolled back, and the next request sent (regardless of what it is) will be rejected with a `TXN_TIMED_OUT` error (even if it a request to commit, rollback, or start a new transaction). This is to indicate that the previous transaction was aborted.

In normal database transaction mode:

- Many updates can be sent and committed to the database all 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 will be 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, thereby preventing multiple clients from updating the database at the same time. If the transaction mechanism did not exist, a client could do a retrieve of a subscription and get one answer, and then do the exact same retrieve and get a different answer if a different connection had modified or deleted the subscription being queried.
- The maximum number of database manipulation requests that can be contained within a transaction is user configurable via the SDS GUI. If the number of database manipulation requests exceeds the configurable limit, each subsequent request within the transaction will fail with `TXN_TOO_BIG` error code.

Note: It is not allowed to send a block transaction (<tx> … </tx>) within 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. If it is, this will result in the block transaction request being rejected with a `INV_REQ_IN_NORMAL_TX` error (which will not affect/abort the currently open transaction).

Block Transaction Mode

The block database transaction mode requires explicit**<tx>** tags around all of the requests within 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 the entire transaction is automatically rolled back. If all requests are successful then the transaction is automatically committed.

If a block transaction fails, the request within the block that encountered an error will have the appropriate error code set, and all requests **after** the failed one will have the error code set to NOT_PROCESSED. Any requests **before** the one that failed will indicate success, and the number of affected rows (as normal).

The maximum number of requests that can be performed within a block transaction are configured by the **Maximum Transaction Size** system variable, as described in *XML/SOAP Interface System Variables*.

The maximum number of requests that can be performed within a block transaction is configured by the**Maximum Transaction Size** system variable, as described in *XML/SOAP Interface System Variables*. If too many requests are sent within a block transaction, the request is failed with a**TXN_TOO_BIG** error. In this case, the transaction is not even executed.

Note: The block transaction mode is not supported on the SOAP interface.

Format:

```
<tx>

    <requestName ...>
    …
    </requestName>

    …

    <requestName ...>
    …
    </requestName>

</tx>
```

Note: It is not allowed to send a block transaction within 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. It is also not allowed to have any normal database transaction requests (i.e. **<startTransaction/>**, **<commit/>** or **<rollback/>**) within a block transaction. If there are any, this will result in the block transaction failing with a **INV_REQ_IN_BLOCK_TX** error.

When incrementing measurements related to block transactions, the whole block is treated as a single provisioning command. Hence, if a block contains 4 requests (such as <updateSubscriber>), then the subsequent measurements will be incremented by 1, not by 4.

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. On the SOAP interface, operations such as **updateSubscriber** are simply sent without sending**startTransaction**/**commit** operations.

Note: When sending Single Database Transaction Mode update/delete requests, each command is implicitely done within a transaction by the SDS, which is like sending**<startTransaction/>**,**<request>**, and**<commit/>**. For read requests, ***no*** transaction is used by the SDS.

ACID-Compliant Transactions

The XML/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

Only one transaction can be open/active at a time across all clients. While one transaction is opened, all other transactions or update/delete requests are blocked until the opened transaction is completed and closed. 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)

Provisioning Data Import (XML)

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

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

- **Blocking** - an import runs while updates are blocked on all other XDS/SOAP 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.

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. The following database manipulation requests are supported in an XML import file:

Table 4: XDS Operations

XDS Operation	Description	Section
<updateSubscriber>	Update Subscriber Routing Data (type IMSI/MSISDN)	*Update Subscriber*
<deleteSubscriber>	Delete Subscriber Routing Data (type IMSI/MSISDN)	*Delete Subscriber*
<updateSubscriberNAI>	Update Subscriber Routing Data (type NAI)	*Update Subscriber NAI*
<deleteSubscriberNAI>	Delete Subscriber Routing Data (type NAI)	*Delete Subscriber NAI*

Unsupported requests (such as read and transaction based requests) are skipped with each occurrence being recorded as BAD_IMPORT_CMD in the import log file as shown in *Figure 3: Import Log File - Import Successfully Completed Example* . All errors encountered while processing the import file are recorded in the import log. Unknown/invalid requests are skipped with each occurrence being recorded as INV_REQUEST_NAME in the import log file

Blank and comment lines will be skipped. The format of a comment line is :

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

Import files that are placed in a specific location on a remote server (configurable via the SDS GUI) are detected within 5 minutes and automatically downloaded via Secure CoPy (SCP) to the file management storage area on the Active SDS server. Once fully downloaded, each file is automatically imported into the provisioning database sequentially in the order in which their download completed. See for more information on automatic file import.

Import file names on the remote server must be suffixed with ".xml" (as shown below) to be automatically downloaded and imported into the SDS Provisioning Database.

```
<filename>.xml
```

Update requests may be unavailable to other clients for the entire 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*). However, read requests are always available.

Note that XML requests must be formatted on a single line. This will both increase performance, and aid in detecting/recovering from badly formatted XML requests. Exported records are also 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 (as shown below).

```
<filename>.xml
<filename>.xml.log
```

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 import log file entries have the following format:

```
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
```

Figure 2: Import Log File Format

Where:

```
mm/dd/yy  Date (in UTC) the entry was logged.

Values: 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.

Values: hh = 00-23 (hours)
mm = 00-59 (minutes)
ss = 00-59 (seconds)
```

```
linesImported Number of lines of the import file that has been processed.

linesToImport Total number of lines of the import file to be processed.

percentCplt Percentage of import file (lines) processed.

reqMsg  XML Request Message that resulted in error (see XML Message Definitions).

errorValueXML Message Response Error Value (see SDS Response Message Error Codes).

errorStringXML Message Response Error String (see SDS Response Message Error Codes).

lineOfFailure Line number of the failed XML Request Message.

description Description (if any) of XML Request Message failure.

successfulCmds Total number of XML Request Messages successfully committed.

failedCmds Total number of XML Request Messages that resulted in failure.
```

```
totalCmds Total number of XML Request Messages that were processed.
```

```
08/06/08 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></deleteSubscribe r>
[error 2007 IMSI_NOT_FOUND : line 36]

<startTransaction/>
[error 1028 BAD_IMPORT_CMD : line 77]

08/06/08 13:28:03 Completed (200 of 200) 100% complete

Successful: 196  Failures: 4  Total: 200
```

Figure 3: Import Log File - Import Successfully Completed Example

In the event the import operation is interrupted/terminated (i.e. abnormally terminated), the number and percentage of requests attempted is reported.

```
08/06/08 13:28:01 Started (0 of 200) 0% complete

08/06/08 13:28:03 Connection terminated

08/06/08 13:28:03 Interrupted (100 of 200) 50% complete

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

Figure 4: Import Log File - Import Interrupted Example

Import log files on the local system are viewable for up to 7 days or until manually removed via the SDS GUI. The failed request and associated response entries have the format as shown in *XML Message Definitions*

Provisioning Data Export (XML)

Provisioning Data can be exported to an ASCII text file in XML format (see *XML ASCII Text File Format*). Export of Provisioning Data can be initiated immediately or scheduled to occur daily, weekly, monthly, or yearly using the SDS GUI XML Export command. Data can only be scheduled to be exported a maximum of once a day, seven days a week.

All of the following Provisioning Data types can be exported in a single export operation or individually using multiple operations:

- IMSI
- MSISDN
- NAI

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

- **Blocking** – an export runs while updates are blocked on all other XDS/SOAP connections. This allows for a logically complete export file created in the fastest time possible at the cost of delaying any new provisioning updates until the export is completed and the transaction is closed.

- **Non-Blocking (Real-time) –** an export runs while updates are continued to be received and committed to the database. As the export progresses, any new instances that have not been passed in table iteration will be included when the export gets to that point. Any new instances whose logical place in the export file has already been passed will not be included. As an additional point of data, the level of the database when the export finished will be inserted at the end of the export file (for this mode only).

Export files are stored locally on the SDS in the file management storage area using the following naming convention:

```
export_<id>.<datatype>.<yyyymmdd><hhmm>.<format>
```

Where:

```
id Export file identifier.
Values: a string up to 100 characters.

datatype Provisioning data type exported (Scheduled exports only).
Values: all All data types
imsi IMSIs
msisdn MSISDNs
nai NAIs

yyyymmdd Date the export was started (Scheduled exports only).
Values: yyyy 1970+ (year)
mm 01-12 (month)
dd 01-31 (day of month)

hhmm Time the export was started (Scheduled exports only).
Values: hh = 00-23 (hours)
mm = 00-59 (minutes)

format Export file format.
Values: xml (XML format)
```

Optionally, export files can be automatically transferred via Secure CoPy (SCP) to a specific location on a remote server (configurable via the SDS GUI). Export files transferred to a remote server are removed from the active server of the Primary SDS cluster.

Export operation status is viewable for up to 7 days or until manually removed the SDS GUI).

XML Export File

The `export` requests exports the provisioning data to an ASCII text file in XML format.

The first line in the export file indicates the mode of export used and what the level was when the export started:

In Blocking mode, the last line of the file will contain the time when the export completed:

In Non-Blocking mode, the last line of the file will contain the current database level and time when the export completed:

Where:

```
modeExport mode.
Values:blockingUpdates are blocked on all other XDS connections until the export
operation is
complete.
realtimeUpdates are allowed on all other XDS connections during the export operation.
```

```
levelDurable database level.
Values:0-4294967295

YYYYMMDDHHMMSSDate and time export operation started and completed.
Values: YYYYYear (decimal number including the century)
MMMonth (01 – 12)
DDDay (01 – 31)
HHHour  (00 – 23)
MMMinute (00 – 59)
SSSecond (00 – 59)
```

The exported file contains a separate section for each data type exported. The start of each section will have a comment line (line starting with #) as its header.

XML ASCII Text File Format

Exporting the provisioning database to an ASCII text file in XML Format creates a file that can be used as an import file. The format of the XML requests in the exported file are exactly the same as the XML requests specified in this document. Each XML request will be formatted onto a single line.

The requests that are placed in the exported file may not be the actual requests that originally created the instances. For example, if an IMSI was created at the same time as another IMSI and an MSISDN, there would be one request for each IMSI and MSISDN.

The provisioning data within the exported file is ordered as follows:

- IMSIs
- MSISDNs
- NAIs

IMSIs

The first section in the exported file contains IMSI data in the following format.

```


<updateSubscriber ent="entityName"
ns="namespace"><imsi>imsi</imsi>[<imshss>imshss</imshss>][<lte hss>
ltehss</ltehss>][<pcrf>pcrf</pcrf>][<ocs>ocs</ocs>][<ofcs>ofcs</ofcs>][<aaa>aaa</aaa>][<userdef1>
userdef1</userdef1>][<userdef2>userdef2</userdef2>]</updateSubscriber>
```

Where:

```

entityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing.

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

imsiA single IMSI.
Values: A string with 10 to 15 characters where each character is a decimal digit
from 0 to 9.

imshss(Optional) The name of the IMS HSS to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ltehss(Optional) The name of the LTE HSS to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
```

```
configured in the SDS.

pcrf(Optional) The name of the PCRF to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ocs(Optional) The name of the OCS to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ofcs(Optional) The name of the OFCS to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

aaa(Optional) The name of the AAA server to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef1 (Optional) The name of the first user defined destination to which the IMSI
 is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef2 (Optional) The name of the second user defined destination to which the
IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.
```

For example, the following fields in each record are defined in the XML:

```
<updateSubscriber ent="subscriberRouting"
ns="dsr"><imsi>310910421000103</imsi><ltehss>LTE_HSS_2
</ltehss><aaa>AAA_4</aaa></updateSubscriber>
<updateSubscriber ent="subscriberRouting"
ns="dsr"><imsi>310910421000305</imsi><ltehss>LTE_HSS_1
</ltehss><aaa>AAA_2</aaa></updateSubscriber>
<updateSubscriber ent="subscriberRouting"
ns="dsr"><imsi>310910421000302</imsi><ltehss>LTE_HSS_1
<ltehss><pcrf>PCRF_3</pcrf><ocs>OCS_1</ocs><aaa>AAA_6</aaa><userdef1>NODE_2</userdef1></updateSubscriber
>
<updateSubscriber ent="subscriberRouting"
ns="dsr"><imsi>310910421000307</imsi><imshss>IMS_HSS_1
</imshss><ltehss>LTE_HSS_1</ltehss><pcrf>PCRF_3</pcrf><ocs>OCS_1</ocs><ofcs>OFCS_2</ofcs><aaa>AAA_6</aaa>
<userdef1>NODE_2</userdef1><userdef2>NODE_1</userdef2></updateSubscriber>
```

MSISDNs

The second section in the export file contains MSISDN data in the following format:

```
<updateSubscriber ent="entityName"
ns="namespace"><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>
```

Where:

```
entityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing.

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

msisdnMSISDN (specified in E.164 international public telecommunication numbering
plan format).
Values: A string with 8 to 15 characters where each character is a decimal digit
from 0 to 9.

imshssThe name of the IMS HSS to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ltehssThe name of the LTE HSS to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

pcrfThe name of the PCRF to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ocsThe name of the OCS to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ofcsThe name of the OFCS to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

aaaThe name of the AAA server to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef1 The name of the first user defined destination to which the MSISDN is
associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef2 The name of the second user defined destination to which the MSISDN is
associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.
```

For example, the following fields in each record are defined in the XML:

```
<updateSubscriber ent="subscriberRouting"
ns="dsr"><msisdn>15634210106</msisdn><imshss>IMS_HSS_1<
/imshss><ltehss>LTE_HSS_1</ltehss><pcrf>PCRF_3</pcrf><aaa>AAA_6</aaa></updateSubscriber>
<updateSubscriber ent="subscriberRouting"
ns="dsr"><msisdn>15634210103</msisdn><imshss>IMS_HSS_1<
```

```
/imshss><ltehss>LTE_HSS_1</ltehss><ocs>OCS_1</ocs><ofcs>OFCS_2</ofcs><userdef1>NODE_2</userdef1></update
Subscriber>
<updateSubscriber ent="subscriberRouting"
ns="dsr"><msisdn>15634210109</msisdn><imshss>IMS_HSS_1<
/imshss><ltehss>LTE_HSS_1</ltehss><pcrf>PCRF_3</pcrf><ocs>OCS_1</ocs><ofcs>OFCS_2</ofcs><aaa>AAA_6</aaa>
<userdef1>NODE_2</userdef1><userdef2>NODE_1</userdef2></updateSubscriber>
```

NAIs

The third section in the export file contains NAI data in the following format:

```


<updateSubscriberNai ent="entityName"
ns="namespace"><host>host</host><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>]</updateSubscriber>
```

Where:

```

entityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing.

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

host A host name.
Values: A string with 1 to 64 characters.

userA user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.

imshssThe name of the IMS HSS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ltehssThe name of the LTE HSS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

pcrfThe name of the PCRF to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ocsThe name of the OCS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ofcsThe name of the OFCS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

aaaThe name of the AAA server to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef1 The name of the first user defined destination to which the NAI is associated
 with.
```

```
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef2 The name of the second user defined destination to which the NAI is
associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.
```

For example, the following fields in each record are defined in the XML:

```
<updateSubscriberNai ent="subscriberRouting"
ns="dsr"><host>operator.com</host><user>john.smith</
user><ltehss>LTE_HSS_1</ltehss><aaa>AAA_6</aaa></updateSubscriber>
<updateSubscriberNai ent="subscriberRouting"
ns="dsr"><host>operator.com</host><user>peter.black<
/user><ltehss>LTE_HSS_1</ltehss><pcrf>PCRF_3</pcrf><ocs>OCS_1</ocs><aaa>AAA_6</aaa><userdef1>NODE_2
</userdef1></updateSubscriber>
<updateSubscriberNai ent="subscriberRouting"
ns="dsr"><host>operator.com</host><user>simon.wells<
/user><imshss>IMS_HSS_1</imshss><ltehss>LTE_HSS_1</ltehss><pcrf>PCRF_3</pcrf><ocs>OCS_1</ocs><ofcs>OFCS_2
</ofcs><aaa>AAA_6</aaa><userdef1>NODE_2</userdef1><userdef2>NODE_1</userdef2></updateSubscriber>
```

Provisioning Data Import (CSV)

Provisioning Data can be imported from ASCII text files in CSV format (see *CSV Import Files*).

The following Provisioning Data types can be imported:

- Destination
- IMSI
- MSISDN
- NAI User
- Wildcard NAI User
- NAI Host

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 files are stored locally on the SDS in the file management storage area using the following naming convention:

```
import_<id>_<datatype>.<format>
```

Where:

```
id Import file identifier. Values: a string up to 100 characters.
```

```
datatype Provisioning data type imported.
Values: destination Destinations
imsi IMSIs
msisdn MSISDNs
naiuser NAI Users
wcnaiuser Wilcard NAI Users
naihost NAI Hosts

format Import file format.
Values: csv (CSV format)
```

Import files that are placed in a specific location on a remote server (configurable via the SDS GUI) are detected within 5 minutes and automatically downloaded via Secure CopPu (SCP) to the file management storage area on the Active SDS server. Once fully downloaded, each file is automatically imported into the provisioning database sequentially in the order in which their download completed. See for more information on automatic file import.

Import file names on the remote server must be suffixed with ".csv" to be automatically downloaded and imported into the SDS Provisioning Database.

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

In all import files, blank lines and lines beginning with the '#' character are considered comments and will be skipped.

Note: CSV import file requests are converted to PDBI commands before being processed. It is the PDBI requests that are reported/logged in the import log files.

An PDBI command 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 `".pdbi"` appended (as shown below).

```
<filename>.csv
<filename>.csv.pdbi
```

An import log file is created for each file that is imported and a copy is automatically uploaded to the same location the PDBI import file above. The log file has the same name as its corresponding import file with ".log" appended (as shown below).

```
<filename>.csv.pdbi
<filename>.csv.pdbi.log
```

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 response message return code (value and string representation), line of the import file containing the failure, and reason (if any).
- Total number of requests successfully committed and failed.

The import log file entries have the following format:

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

reqMsg
[rc returnCodeValue returnCodeString : line lineOfFailure]  [failureDesc]

. . .

reqMsg
[rc returnCodeValue returnCodeString : line lineOfFailure]  [failureDesc]

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

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

Figure 5: CSV Import Log File Format

Where:

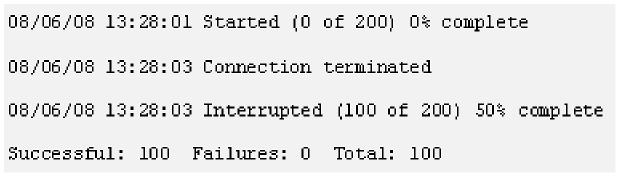
```
mm/dd/yy Date (in UTC) the entry was logged.

Values: 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.

Values: hh = 00-23 (hours)
mm = 00-59 (minutes)
ss = 00-59 (seconds)
```

```
linesImported Number of lines of the import file that has 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 (see section 4).

returnCodeValueMessage Response Error Value (see Section Appendix A).

returnCodeStringMessage Response Error String (see Section Appendix A).

lineOfFailure Line number of the failed CSV Request Message.

failureDesc Description (if any) of  CSV Request Message failure.

successfulCmds Total number of CSV Request Messages successfully committed.

failedCmds Total number of CSV Request Messages that resulted in failure.

totalCmds Total number of CSV Request Messages that were processed.
```

```
08/06/08 13:28:01 Started (0 of 200) 0% complete

upd_route(imsi 666666661000000, ltehss mh_hss61, timeout 2)
[rc 1017 NO_UPDATES : line 23] reason "The change was already in the database"

rtrv sub(dn 9195551000)
[rc 1028 BAD_IMPORT_CMD : line 36]

08/06/08 13:28:03 Completed (200 of 200) 100% complete

Successful: 198  Failures: 2  Total: 200
```

Figure 6: CSV Import Log File – Import Successfully Completed Example

In the event the import operation is interrupted/terminated (i.e. abnormally terminated), the number and percentage of requests attempted is reported.



```
08/06/08 13:28:01 Started (0 of 200) 0% complete

08/06/08 13:28:03 Connection terminated

08/06/08 13:28:03 Interrupted (100 of 200) 50% complete

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

Figure 7: CSV Import Log File – Import Interrupted Example

Import log files on the local system are viewable for up to 7 days or until manually removed via the SDS GUI. The failed request and associated response entries have the format as shown in *XML Message Definitions*.

CSV Import Files

The import request imports the provisioning data from ASCII text files in CSV format, the format being dependant on the type of data.

Destinations

The import file contains Destination data in the following formats.

For updating/creating a Destination:

```
# Destinations
U,<name>,<type>,<fqdn>,<realm>
```

For deleting a Destination:

```
# Destinations
D,<name>,<type>
```

Note: Both update/create and delete requests are allowed in the same file, in any order.

Where:

```
name The name of the destination.
Values: A string with 1 to 32 characters.

type The type of destination.
Values: imshss Destination is an IMS HSS.
ltehss Destination is an LTE HSS.
pcrf Destination is a PCRF.
ocs Destination is an OCS.
ofcs Destination is an OFCS.
aaa Destination is a AAA server.
userdef1 Destination is user defined #1.
userdef2 Destination is user defined #2.

fqdn (Optional) The Diameter FQDN for the Destination. The value can be null.
Values: A string with 1 to 255 characters.

realm (Optional) The Diameter Realm for the Destination. The value can be null.
Values: A string with 1 to 255 characters..
```

Note: Either the FDQN or Realm, or FDQN and Realm must be defined.

For example, the following fields in each record are defined in the CSV:

```
# Destinations
U,LTE_HSS_1,ltehss,operator.com,
U,LTE_HSS_2,ltehss,network.com,
U,HSS_West,imshss,wireless.com,hsswest
D,AAA_2,aaa
```

IMSIs

The import file contains IMSI data in the following formats.

For updating/creating an IMSI:

```
# IMSIs
U,<imsi>,<imshss>,<ltehss>,<pcrf>,<ocs>,<ofcs>,<aaa>,<userdef1>,<userdef2>
```

For deleting an IMSI:

```
# IMSIs
D,<imsi>
```

Note: Both update/create and delete requests are allowed in the same file, in any order.

Where:

```
imsi A single IMSI.
Values: A string with 10 to 15 characters where each character is a decimal digit
from 0 to 9.

imshss (Optional) The name of the IMS HSS to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ltehss (Optional) The name of the LTE HSS to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

pcrf (Optional) The name of the PCRF to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ocs (Optional) The name of the OCS to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ofcs(Optional) The name of the OFCS to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

aaa (Optional) The name of the AAA server to which the IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef1 (Optional) The name of the first user defined destination to which the IMSI
 is associated with.
```

```
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef2 (Optional) The name of the second user defined destination to which the
IMSI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.
```

For example, the following fields in each record are defined in the CSV:

```
# IMSIs
U,310910421000103,,LTE_HSS_2,,,,AAA_4,,
U,310910421000105,,LTE_HSS_1,,,,AAA_2,,
U,310910421000102,,LTE_HSS_1,PCRF_3,OCS_1,,AAA_6,NODE_2,
U,310910421000107,IMS_HSS_1,LTE_HSS_1,PCRF_3,OCS_1,OFCS_2,AAA_6,NODE_2,NODE_1
D,310910421000110
```

MSISDNs

The import file contains MSISDN data in the following formats

For updating/creating an MSISDN:

```
# MSISDNs
U,<msisdn>,<imshss>,<ltehss>,<pcrf>,<ocs>,<ofcs>,<aaa>,<userdef1>,<userdef2>
```

For deleting an MSISDN:

```
# MSISDNs
D,<msisdn>
```

Note: Both update/create and delete requests are allowed in the same file, in any order.

Where:

```
msisdn MSISDN (specified in E.164 international public telecommunication numbering
 plan format).
Values: A string with 8 to 15 characters where each character is a decimal digit
from 0 to 9.

imshss The name of the IMS HSS to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ltehss The name of the LTE HSS to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

pcrf The name of the PCRF to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ocs The name of the OCS to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.
```

```
ofcs The name of the OFCS to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

aaa The name of the AAA server to which the MSISDN is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef1 The name of the first user defined destination to which the MSISDN is
associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef2 The name of the second user defined destination to which the MSISDN is
associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.
```

For example, the following fields in each record are defined in the CSV:

```
# MSISDNs
U,15634210103,IMS_HSS_1,LTE_HSS_1,PCRF_3,,,AAA_6,,
U,15634210106,IMS_HSS_1,LTE_HSS_1,,OCS_1,OFCS_2,,NODE_2,
U,15634210109,IMS_HSS_1,LTE_HSS_1,PCRF_3,OCS_1,OFCS_2,AAA_6,NODE_2,NODE_1
D,15634210110
```

NAI Users

The import file contains NAI User data in the following formats.

For updating/creating an NAI User:

```
# NAI Users
U,<user>,<host>,<imshss>,<ltehss>,<pcrf>,<ocs>,<ofcs>,<aaa>,<userdef1>,<userdef2>
```

For deleting an NAI User:

```
# NAI Users
D,<user>,<host>
```

Note: Both update/create and delete requests are allowed in the same file, in any order.

Where:

```
user A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.

host A host name.
Values: A string with 1 to 64 characters.

imshss The name of the IMS HSS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ltehss The name of the LTE HSS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
```

```
configured in the SDS.

pcrf The name of the PCRF to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ocs The name of the OCS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ofcs The name of the OFCS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

aaa The name of the AAA server to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef1 The name of the first user defined destination to which the NAI is associated
 with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef2 The name of the second user defined destination to which the NAI is
associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.
```

For example, the following fields in each record are defined in the CSV:

```
# NAI Users
U,john.smith,operator.com,,LTE_HSS_1,,,,AAA_6,,
U,peter.black,operator.com,,LTE_HSS_1,PCRF_3,OCS_1,,AAA_6,NODE_2,
U,simon.wells,operator.com,IMS_HSS_1,LTE_HSS_1,PCRF_3,OCS_1,OFCS_2,AAA_6,NODE_2,NODE_1

D,david.jones,operator.com
```

Wildcard NAI Users

The import file contains Wildcard NAI User data in the following formats.

For updating/creating a WildCard NAI User:

```
# Wildcard NAI Users
U,<wcuser>,<host>,<imshss>,<ltehss>,<pcrf>,<ocs>,<ofcs>,<aaa>,<userdef1>,<userdef2>
```

For deleting a WildCard NAI User:

```
# Wildcard NAI Users
D,<wcuser>,<host>
```

Note: Both update/create and delete requests are allowed in the same file, in any order.

Where:

```
wcuser A wildcarded user name to be associated with the host to form a Wildcard NAI.
```

```
Values: A string with 1 to 64 characters.

host A host name.
Values: A string with 1 to 64 characters.

imshss The name of the IMS HSS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ltehss The name of the LTE HSS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

pcrf The name of the PCRF to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ocs The name of the OCS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

ofcs The name of the OFCS to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

aaa The name of the AAA server to which the NAI is associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef1 The name of the first user defined destination to which the NAI is associated
 with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.

userdef2 The name of the second user defined destination to which the NAI is
associated with.
Values: A string with 1 to 32 characters. The destination for which the name refers
 must already be
configured in the SDS.
```

For example, the following fields in each record are defined in the CSV:

```
# Wildcard NAI Users
U,chattserver-,operator.com,,LTE_HSS_1,,,,AAA_6,,
U,chattserver-,network.com,,LTE_HSS_1,PCRF_3,OCS_1,,AAA_6,NODE_2,
U,chattserver-,wireless.com,IMS_HSS_1,LTE_HSS_1,PCRF_3,OCS_1,OFCS_2,AAA_6,NODE_2,NODE_1

D,chattserver-,telephone.com
```

NAI Hosts

The import file contains NAI Host data in the following formats.

For updating/creating an NAI Host:

```
# NAI Hosts
U,<host>
```

For deleting an NAI Host:

```
# NAI Hosts
D,<host>
```

Note: Both update/create and delete requests are allowed in the same file, in any order.

Where:

```
host A host name.
Values: A string with 1 to 64 characters.
```

For example, the following fields in each record are defined in the CSV:

```
# NAI Hosts
U,operator.com
U,hotmail.com
U,google.com
D,yahoo.com
```

Provisioning Data Export (CSV)

Provisioning Data can be exported to an ASCII text file in CSV format (see *CSV Export Files*). Export of Provisioning Data can be initiated immediately or scheduled to occur daily, weekly, monthly, or yearly using the SDS GUI CSV Export command. Data can only be scheduled to be exported a maximum of once a day, seven days a week.

All of the following Provisioning Data types can be exported in a single export operation or individually using multiple operations:

- Destination
- IMSI
- MSISDN
- NAI User
- Wildcard NAI User
- NAI Host

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

- **Blocking** – an export runs while updates are blocked on all other PDBI connections. This allows for a logically complete export file created in the fastest time possible at the cost of delaying any new provisioning updates until the export is completed and the transaction is closed.
- **Non-Blocking (Real-time)** – an export runs while updates are continued to be received and committed to the database. As the export progresses, any new instances that have not been passed in table iteration will be included when the export gets to that point. Any new instances whose logical place in the export file has already been passed will not be included. As an additional point of data, the level of the database when the export finished will be inserted at the end of the export file (for this mode only).

Export files are stored locally on the SDS in the file management storage area using the following naming convention:

```
export_<id>.<datatype>.<yyyymmdd><hhmm>.<format>
```

Where:

```
id Export file identifier.
Values: a string from 4 to 12 characters.

datatype Provisioning data type exported (Scheduled exports only).
Values: all All data types
destination Destinations
imsi IMSIs
msisdn MSISDNs
naiuser NAI Users
wcnaiuser Wildcard NAI Users
naihost NAI Hosts
nai NAI Users

yyyymmdd Date the export was started (Scheduled exports only).
Values: yyyy 1970+ (year)
mm 01-12 (month)
dd 01-31 (day of month)

hhmm Time the export was started (Scheduled exports only).
Values: hh = 00-23 (hours)
mm = 00-59 (minutes)

format Export file format.
Values: xml (XML format)
```

Optionally, export files can be automatically transferred via Secure CoPy (SCP) to a specific location on a remote server (configurable via the SDS GUI). Export files transferred to a remote server are removed from the active server of the Primary SDS cluster.

Export operation status is viewable for up to 7 days or until manually removed the SDS GUI).

CSV Export Files

The export requests exports the provisioning data to an ASCII text file in CSV format.

The first line in the export file indicates the mode of export used and what the level was when the export started:

```
# mode, db level=level, start time=YYYYMMDDHHMMSS,,
```

In Blocking mode, the last line of the file will contain the time when the export completed:

```
# finish time=YYYYMMDDHHMMSS,,,
```

In Non-Blocking mode, the last line of the file will contain the current database level and time when the export completed:

```
# db level=level,finish time=YYYYMMDDHHMMSS,,,
```

Where:

```
mode Export mode.
Values: blocking Updates are blocked on all other XDS connections until the export
 operation is
complete.

realtime Updates are allowed on all other XDS connections during the export operation.
```

```
level Durable database level.
Values: 0–4294967295

YYYYMMDDHHMMSS Date and time export operation started and completed.
Values: YYYY Year (decimal number including the century)
MM Month (01 – 12)
DD Day (01 – 31)
HH Hour (00 – 23)
MM Minute (00 – 59)
SS Second (00 – 59)
```

The exported file contains a separate section for each data type exported. The start of each section will have a comment line (line starting with #) as its header.

Delimiter-Separated Values ASCII Text File Format

Exporting the provisioning database to an ASCII text file in delimiter-separated values format creates an exported file which can easily be read by other client applications, but cannot be used to be imported into the provisioning database like the PDBI format.

The delimiter-separated values file format stores data by separating the values in each row with specific delimiter characters. Comma, pipe, colon, semicolon, and space character may be used to separate the values. Comma-separated values (CSV) file format is one of the most widely used since it can be opened by most spreadsheet programs without designating which delimiter has been used.

The requests that are placed in the exported file may not be the actual requests that originally created the instances. For example, if an IMSI was created at the same time as another IMSI and an MSISDN, there would be one request for each IMSI and MSISDN.

All of the following Provisioning Data types can be exported in a single export operation or individually using multiple operations. When exporting all data types, the provisioning data within the exported file is ordered as follows:

- Destinations
- IMSIs
- MSISDNs
- NAI Users
- Wildcard NAI Users
- NAI Hosts

The following sections describe the data included in the delimiter-separated values ASCII text file.

Note that optional parameters may be missing (appearing as two consecutive commas “,,”) which means the parameter has no value defined for this record.

Destinations

The format of the Destinations section is as described in *Destinations*.

IMSIs

The format of the IMSIs section is as described in *IMSIs*.

MSISDNs

The format of the MSISDNs section is as described in *MSISDNs*.

NAI Users

The format of the NAI Users section is as described in *NAI Users*.

Wildcard NAI Users

The format of the Wildcard NAI Users section is as described in *Wildcard NAI Users*.

NAI Hosts

The format of the NAI Hosts section is as described in *NAI Hosts*.

MSISDN Format

North America MSISDNs shall start with “1” followed by exactly ten digits. This is required in order to support the NPA Split mechanism which is ONLY supported in North America, so it does not apply to any MSISDNs in other regions.

Connection Management

It is possible to enable/disable provisioning on the XML/SOAP interfaces in a number of different ways.

Disable Interface

The configured listening ports for the XML and SOAP interfaces (see *XML/SOAP Interface System Variables*XML Interface Port and SOAP Interface Port) can each be set to 0 to completely disable the respective interface. In this case, no alarms or events are generated for the interfaces indicating they are disabled.

If the XML interface is enabled, then paired alarm 14101 (as described in *Alarms*) is raised on startup, and cleared when at least one client is connected to the XML interface. Alarm 14101 is not raised on the SOAP interface (if enabled), as SOAP is effectively a connectionless protocol, and a system might not always be connected.

Note: The port number cannot be changed to 0 while the system is running. To disable the interface port must be changed to 0, and the system restarted.

Connections Allowed

The configuration variable `Connections Allowed` (see *XML/SOAP Interface System Variables*) controls whether XML/SOAP connections are allowed to the configured port. If this variable is set to `NOT_ALLOWED`, then all existing connections are immediately dropped, and all outstanding transactions rolled back. Alarm 14100 (as described in *Alarms*) is raised. New attempts to connect are rejected.

When `Connections Allowed` is set back to `ALLOWED`, the alarm is cleared, and connections are accepted again.

Disable Provisioning

When the GUI option to disable provisioning is selected, existing connections remain up, and new connections are allowed. But, any provisioning request that is sent will be rejected with an error of `PROV_PROHIBITED`. No alarms are raised when provisioning is disabled, or re-enabled.

Measurements

XDS and SOAP Server specific measurements that are collected and made available to the user via the SDS GUI. The XDS, 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 5: XDS Measurements

ID	Group	Tag	Coll Interval	Scope	Description
4100	PROV	ProvConnectsAttempted	5 min	A	The total number of client initiated connect attempts to establish a connection with the server.
4101	PROV	ProvConnectsAccepted	5 min	A	The total number of client initiated connect attempts that have been accepted.
4102	PROV	ProvConnectsDenied	5 min	A	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	A	The total number of client initiated connect attempts that failed due to errors during initialization.
4105	PROV	ProvConnectionIdleTimeouts	5 min	A	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*
4110	PROV	ProvMsgsReceived	5 min	A	The total number of provisioning messages that have been received.

ID	Group	Tag	Coll Interval	Scope	Description
4111	PROV	ProvMsgsSuccessful	5 min	A	The total number of provisioning messages that have been successfully processed.
4112	PROV	ProvMsgsFailed	5 min	A	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	A	The total number of provisioning messages that have been sent.
4114	PROV	ProvMsgsDiscarded	5 min	A	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	A	The total number of provisioning messages that have been received from an import operation.
4140	PROV	ProvTxnCommitted	5 min	A	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	A	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	A	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	A	The total number of transactions that have been successfully aborted.
4144	PROV	ProvTxnTotal	5 min	A	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	Scope	Description
4145	PROV	ProvTxnDurabilityTimeouts	5 min	A	The total number of committed, non-durable transaction that have failed to become durable within the amount of time specified by **Transaction Durability Timeout** , as described in *XML/SOAP Interface System Variables*.
4160	PROV	ProvImportsSuccessful	5 min	A	The total number of files imported successfully.
4161	PROV	ProvImportsFailed	5 min	A	The total number of files that had failed to be imported due to errors
4162	PROV	ProvExportsSuccessful	5 min	A	The total number of successful CSV/XML export requests.
4163	PROV	ProvExportsFailed	5 min	A	The total number of CSV/XML export requests that have failed due to errors.

Key Performance Indicators (KPIs)

XDS and SOAP Server specific Key Performance Indicators (KPIs) are available to the user via the SDS GUI

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

Table 6: XDS KPI Measurements

ID	Name	Avg. Interval	Scope	Description
4104	ProvConnections	60 sec	A	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	A	The number of provisioning messages that have been received per second.
4111	ProvMsgsSuccessful	60 sec	A	The number of provisioning messages that have been successfully processed per second.
4112	ProvMsgsFailed	60 sec	A	The number of provisioning messages that have failed to be processed due to errors per second. See *SDS Response Message Error Codes* for a list and description of possible errors.

ID	Name	Avg. Interval	Scope	Description
4113	ProvMsgsSent	60 sec	A	The number of provisioning messages sent per second.
4114	ProvMsgsDiscarded	60 sec	A	The number of provisioning messages discarded per second. Provisioning messages being discarded is 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	A	The number of provisioning messages imported per second.
4140	ProvTxnCommitted	60 sec	A	The number of provisioning transactions that have been successfully committed per second to the database (memory and on disk) on the active server of the primary SDS cluster.
4142	ProvTxnFailed	60 sec	A	The number of provisioning transactions that have failed to be started, committed, or aborted due to errors per second. See *SDS Response Message Error Codes* for a list and description of possible errors.
4143	ProvTxnAborted	60 sec	A	The number of provisioning transactions aborted per second.
4150	ProvTxnActive	60 sec	A	The number of provisioning transactions that are currently active (normal transaction mode only).
4151	ProvTxnNonDurable	60 sec	A	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.

Alarms

XDS 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.

Table 7: XDS 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.

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
			PROV Interface manually enabled	Clear					Provisioning interface is manually enabled.
14101	PROV	No Remote Client 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 ID, IP IP Address)	Clear					Alarm automatically cleared after 5 minutes.
14103	PROV	Invalid Provisioning Configuration	Conflicting port numbers	Major	N/A	Normal	5	0	XML & SOAP Provisioning interfaces are disabled since same port is configured for both interface.

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
			Valid provisioning configuration	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 from 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	43200	43200	Provisioning import initialization failed due to an error specified in addl info.
			Initialization error cleared	Clear					Provisioning import initialization completed successfully.
14151	PROV	Import Generation Failed	Failed to import file, see trace log for details	Major	provimport	Normal	43200	43200	Provisioning import operation failed due to an error specified in addl info.
			Generation error cleared	Clear					Provisioning import operation completed successfully.

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
14152	PROV	Import Transfer Failed	Failed to transfer file from remote host, see trace log for details	Major	provimport	Normal	43200	43200	Provisioning import operation failed due to a file transfer error specified in addl info.
			Failed to transfer file to remote host, see trace log for details						
			Transfer error cleared	Clear					Provisioning import operation completed successfully.
14153	PROV	Export Initialization Failed	Initialization error, see trace log for details	Major	provimport	Normal	43200	43200	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	43200	43200	Provisioning export operation failed due to an error specified in addl info.
			Generation error cleared	Clear					Provisioning export operation completed successfully.

ID	Group	Name	Addl Info	Severity	Instance	HA Score	Throttle Secs	Auto Clear Secs	Assert/clear condition(s)
14155	PROV	Export Transfer Failed	Failed to transfer file to remote host, see trace log for details	Major	provimport	Normal	43200	43200	Provisioning export operation failed due to a file transfer error specified in addl info.
			Transfer error cleared	Clear					Provisioning export operation completed successfully.

Events

XDS 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.

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

Table 8: XDS Events

ID	Name/Descr Text	Addl Info	Description
14120	Connection Established	Remote client connection established (CID *Connection ID*, IP *IP Address*)	This event is generated each time a remote provisioning client has successfully established a connection.
14121	Connection Terminated	Remote client connection terminated (CID *Connection ID* , IP *IP Address*)	This event is generated each time a remote provisioning client connection terminates.
14122	Connection Denied	Interface Disabled	This event is generated each time a local or remote provisioning client initiated connection establishment is denied due to 1) XDS interface not enabled; 2) connection originating from an unauthorized IP address; 3) maximum number of
		Connection Unauthorized	
		Too Many Connections	
		Connection to standby XDS not permitted	

ID	Name/Descr Text	Addl Info	Description
			allowed remote client connections have been reached; 4) connection to standby XDS not permitted.
14160	Import Successful	See XML Import screen for details.	This event is generated each time an XML import is successful.
14161	Export Successful	See XML Export screen for details.	This event is generated each time an XML export is successful.

Chapter 4

XML Message Definitions

Topics:

This chapter describes XML requests and responses syntax and parameters.

Message Conventions

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

Table 9: Message Conventions

Symbol	Description
Bold	Text that appears literally in the message (eg, keywords, commas, parentheses, and error text are **bold**)
italics	Parameter values that will be replaced by an actual parameter name or numeric value.
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.
…	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. For example, in **parameter1** < *1* | *2* | *3* >, the numbers represent specific value choices. In **parameter2** < *ServerName* > , the *ServerName* represents the actual value. In **parameter3** < *0..3600* >, the numbers represent a choice in the range from 0 to 3600.
[ ]	Square brackets are used to enclose an optional parameter and its value, such as [, **parameter1** < *1* | *2* | *3* >]. A parameter and its value that are not enclosed in square brackets are mandatory.
|	When the pipe symbol is used in a parameter value list, such as Parameter1 < *1* | *2* | *3* >, it indicates a choice between available values.
,	A literal comma is used in the message to separate each parameter that is specified.

XML Request/Response Length Indication

Every XML request/response sent to or from the XDS must be preceded by a 4 byte binary length indication, indicating the number of bytes that follow making up the XML request/response.

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.

Transaction Id (ID)

Each message has a Transaction Id called the id as an attribute. The id attribute is used by the XDS 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 user specifies the id attribute in a request, the same id attribute and value are returned by XDS in the corresponding response, so use a unique id value in each request message to differentiate responses.

Response Messages

An XML response message is sent by the XDS in response to an XML request.

A response message is sent by the XDS consisting of a 4 byte binary length value indicating how many bytes are to follow which form the XML response, then followed by the XML response itself in ASCII characters.

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 (as in the `<readSubscriber>` and `<readSubscriberNai>` requests).

A generic response type can be generated in the case where an XML request cannot be parsed, the request itself is not a valid request, and in some other cases. In this case the response name will be `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.

The syntax of the response is the same for all requests and is as follows:

```
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>
```

Parameters:

```
lengthInBytesNumber of bytes following to form XML request. Note this is a 4 byte
binary value.
Values: 0-4294967295

respNameThe name of the response based on the original XML request sent. Value will
 be request name
appended with “Resp”. E.g. for the “updateSubscriber” request, the response name
will be
“updateSubscriberResp”. In the case where the request name is invalid, or the XML
cannot be
parsed the response name will be “errorResp”.
Values: A string with 1 to 64 characters.

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295

originalXMLRequest(Optional) The text of the original XML request that was sent.
Note: this is only present if the resonly=”n” attribute is set in the original
request.
Values: A string with 1 to 4096 characters.

errorError code. Whether or not operation was successfully executed by the XDS. See
 Table 9 for generic values, and the table below for values specific to this request.
Values: 0success
non zerofailure

affected  The number of routing entities created/updated.
  Values: 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.
Values: A string with 1 to 1024 characters.

rowNameThe name of the row type returned.
Values: Name of the row dependant on the result set returned.

rowValueThe value of the row type returned.
Values: Type and size dependant on result set returned.

rowAttributeNameThe name of the row attribute name returned.
Values: Name of the row attribute name dependant on the result set returned.

rowAttributeValueThe value of the row attribute name returned.
Values: Type and size dependant on result set returned.

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295.

imsi(Optional – see List item.) An IMSI (specified in E.212 format).
```

```
Values: 10 to 15 numeric digits.
```

Common Error Codes

Table 10: Common Error Codes

Error Code	Description
SUCCESS	Request was successful.
PROV_PROHIBITED	DB access has been manually disabled.
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.
INV_REQUEST_NAME	The XML request name does not indicate a valid request.
INVALID_VALUE	One of the fields in the request has a invalid value.
INVALID_XML	The request does not contain a valid XML data structure and cannot be parsed.
MISSING_PARAMETER	A mandatory parameter is missing.
INVALID_MULT_INST	Multiple instances of a parameter than only allows a single instance has been encountered,
UNKNOWN_PARAM_NAME	The specified parameter name is unknown for this request.
DB_EXCEPTION	An unexpected exception was thrown during the database commit. The entire transaction was rolled back to ensure predictable behavior. Contact Tekelec.
DURABILITY_TIMEOUT	The update was not made durable in the database within the configured time interval
TXN_TOO_BIG	Transaction too big (more than the configured maximum number of requests). The maximum number of requests within a transaction is configured using the **Maximum Transaction Size** system variable, as described in *XML/SOAP Interface System Variables*.
NO_UPDATES	The transaction did not have any successful updates.

Error Code	Description
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_NORMAL_TX	An invalid request has been sent in a normal transaction (e.g. a block transaction)

Note: If a request results in a `DURABILITY_TIMEOUT` or `DURABILITY_DEGRADED` error, this means that the request was successfully performed, but the database durability requirements have not ***yet*** been reached. The request has been executed on the local database, but full replication (based on the configured database durability level) has not yet been achieved within the configured period. Sending the request again will not achieve anything different, and will likely result in a `NO_UPDATES` response.

List of Request Messages

The XDS supports the requests listed in *Table 11: Supported XDS Requests*. Unsupported operations/requests are rejected with an `INV_REQUEST_NAME` error code. XDS clients are to construct requests as specified in the sections referenced in *Table 11: Supported XDS Requests*.

Table 11: Supported XDS Requests

Request	Description	Section
startTransaction	Start Database Transaction	*Start Transaction*
commit	Commit Database Transaction	*Commit Transaction*
rollback	Abort Database Transaction	*Rollback Transaction*
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.

Take note of the following:

- If a **<startTransaction>** command is sent, and then the connection is lost or the user logs off without sending a **<commit>** or **<rollback>** command, all pending requests are rolled back.
- One XDS session can have one transaction open at a time. If a **<startTransaction>** command is sent, another **<startTransaction>** command will fail with an `ACTIVE_TXN` error.
- There is a timeout between the**<startTransaction>** and the**<commit>** commands. If the**commit** command is not sent out within the configured Maximum Transaction Lifetime (see *XML/SOAP Interface System Variables*) of the**<startTransaction>** command, the XML provisioning requests are rolled back (changes not applied to database).
- Opened only 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`.
- Accepts database manipulation and query requests (update, delete, and read)
- 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.
- Can be aborted and rolled back with a **<rollback>** request anytime before the transaction is ended with a **<commit>** request.
- A block transaction (i.e. <tx> … </tx>) is not allowed with a normal transaction, and will result in a `INV_REQ_IN_NORMAL_TX` error being returned for that request.

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

Parameters:

```
resonly(Optional) Indicates whether the response should consist of  the result only,
 without including
the original request in the response.
Values: y only provide the result, do NOT include the original request (default)
ninclude the original request in the response

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 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
their <startTransaction> requests were received..
Values: 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.

Table 12: Start Transaction rEsponse Error Codes

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.

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. It is very important to understand that all previous updates, even though they received a successful error code, are not committed to the database until the **commit** request is received.

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

Parameters:

```
resonly(Optional) Indicates whether the response should consist of  the result only,
 without including
the original request in the response.
Values: y only provide the result, do NOT include the original request (default)
ninclude the original request in the response

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 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 will cause 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.

Table 13: Commit Transaction Response Error Codes

Return Code	Description
SUCCESS	Database transaction was committed successfully
NO_ACTIVE_TXN	A transaction is not currently open on this connection.
DB_EXCEPTION	An unexpected exception was thrown during the database commit. The entire transaction was rolled back to ensure predictable behavior. Contact Tekelec.

Rollback Transaction

Request

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

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

Parameters:

```
resonly(Optional) Indicates whether the response should consist of  the result only,
 without including the original request in the response.
Values: y only provide the result, do NOT include the original request (default)
ninclude the original request in the response

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295.
```

Response

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

Table 14: Rollback Transaction Response Error Codes

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

Update Subscriber

The `<updateSubscriber>` request provisions IMSI and MSISDN routing data

Request

The <updateSubscriber> request allows for the provisioning of either IMSI, MSISDN, or combinations of IMSI and MSISDNs to be associated with 8 different destinations.

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

The update command will both update existing routing entities, and it will create any new routing entities that do not exist.

```
<updateSubscriber ent="entityName" ns="namespace" [resonly="resonly"] [id="id"]
                  [timeout="timeout"]>
[
    <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>
```

Parameters:

```
entityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

resonly(Optional) Indicates whether the response should consist of  the result only,
 without including
the original request in the response.
Values: y only provide the result, do NOT include the original request (default)
ninclude the original request in the response

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 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. Valid values are 0
(return immediately
if not available) to 3600 seconds (default is 0). Note: if the request is being
performed within a
transaction, this parameter will have no effect, as the client already has a
transaction open.
```

```
Values: 0 (return immediately if not available) to 3600 seconds (default is 0).

imsi(Optional – see List item.) An IMSI (specified in E.212 format).
Values: 10 to 15 numeric digits.

msisdn  (Optional – see List item.) An MSISDN (specified in E.164 international
public
  telecommunication numbering plan format).
Values: 8 to 15 numeric digits.

imshss(Optional – see List item.) The name of the IMS HSS for which the specified
routing entity(s) are
to be associated.
Values: A string with 1 to 32 characters.

ltehss(Optional – see List item.) The name of the LTE HSS for which the specified
routing entity(s) are
to be associated.
Values: A string with 1 to 32 characters.

pcrf(Optional – see List item.) The name of the PCRF for which the specified routing
 entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

ocs(Optional – see List item.) The name of the OCS for which the specified routing
 entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

ofcs(Optional – see List item.) The name of the OFCS for which the specified routing
 entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

aaa(Optional – see List item.) The name of the AAA server for which the specified
routing entity(s) are
to be associated.
Values: A string with 1 to 32 characters.

userdef1(Optional – List item.) The name of the first user defined destination for
 which the
specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.

userdef2(Optional – List item.) The name of the second user defined destination for
 which the
specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.
```

Semantic Rules:

1. There must be a minimum of 1 routing entity (IMSI or MSISDN)
2. There must be no more than 10 routing entities (IMSI, MSISDN, or combinations of the two)
3. There must be at least one destination specified
4. A destination name must already exist in the database
5. Each destination name type may only be specified once
6. All specified routing entities will be provisioned with the same destination value(s)
7. Any existing destination(s) for a routing entity will not be changed/removed if not specified in the request

8. Specifying a destination name of "none" will remove the association of that destination from the specified routing entity(s).

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 the same destination values that already exist, this may result in `NO_UPDATES` being returned, which is not treated as an error. When a subscriber record is not updated, this means that the affected rows count is not incremented for that IMSI/MSISDN.

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

Parameters:

```
id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295

originalXMLRequest(Optional) The text of the original <updateSubscriber> XML request
 that was sent.
Note: this is only present if the resonly=”n” attribute is set in the original
request.
Values: A string with 1 to 4096 characters.

errorError code. Whether or not operation was successfully executed by the XDS. See

Table 10:  Common Error CodesTable 9 for generic values, and the table below for
values specific
to this request.
Values: 0success
non zerofailure

affected  The number of routing entities created/updated. Note: if a routing entity
 is updated with the
same destination(s), causing no change to be made, the affected value will not be
incremented.
Values: 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.
Values: A string with 1 to 1024 characters.
```

Table 15: Update Subscriber Response Error Codes

Error Code	Description
SUCCESS	The update request was successfully completed.
DEST_NOT_FOUND	Destination name does not exist.

Error Code	Description
TOO_MANY_ADDR	Too many address values supplied.
NO_DEST_VAL	No destination name supplied.
NO_ADDR_VAL	No address value supplied.

Delete Subscriber

The `<deleteSubscriber>` request removes IMSI and MSISDN routing data.

Request

The `<deleteSubscriber>` request allows for the removal of either IMSI, MSISDN, or combinations of IMSI and MSISDN routing entity(s).

```
<deleteSubscriber ent="entityName" ns="namespace" [resonly="resonly"] [id="id"]
                  [timeout="timeout"]>
 [
    <imsi>imsi</imsi>

    <imsi>imsi</imsi>
]
[
    <msisdn>msisdn</msisdn>
    …
    <msisdn>msisdn</msisdn>
]
</deleteSubscriber>
```

Parameters:

```
entityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

resonly(Optional) Indicates whether the response should consist of  the result only,
 without including
the original request in the response.
Values: y only provide the result, do NOT include the original request (default)
ninclude the original request in the response

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 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. Valid values are 0
(return immediately
if not available) to 3600 seconds (default is 0). Note: if the request is being
performed within
a transaction, this parameter will have no effect, as the client already has a
transaction open.
Values: 0 (return immediately if not available) to 3600 seconds (default is 0).

imsi(Optional – see List item.) An IMSI (specified in E.212 format).
Values: 10 to 15 numeric digits.

msisdn  (Optional – see List item.) An MSISDN (specified in E.164 international
public
  telecommunication numbering plan format).
Values: 8 to 15 numeric digits.
```

Semantic Rules:

1. There must be a minimum of 1 routing entity (IMSI or MSISDN)
2. There must be no more than 10 routing entities (IMSI, MSISDN, or combinations of the two)

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.

If only one IMSI or MSISDN is supplied in a request, and the IMSI/MSISDN does not exist, or all IMSI/MSISDNs provided in the request do not exist, then this is not treated as an error, and the request will return with `NO_UPDATES`. When a subscriber record does not exist, this means that the affected rows count is not incremented for that IMSI/MSISDN.

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

Parameters:

```
id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295

originalXMLRequest(Optional) The text of the original <deleteSubscriber> XML request
 that was sent.
Note: this is only present if the resonly=”n” attribute is set in the original
request.
Values: A string with 1 to 4096 characters

errorError code. Whether or not operation was successfully executed by the XDS. See

Common Error Codes for generic values, and the table below for values specific to
this request.
Values: 0success
non zerofailure

affected  The number of routing entities deleted.
Values: 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.
Values: A string with 1 to 1024 characters.
```

Table 16: Delete Subscriber Response Error Codes

Error Code	Description
SUCCESS	The delete request was successfully completed.
TOO_MANY_ADDR	Too many address values supplied.
NO_ADDR_VAL	No address value supplied.

Read Subscriber

The `<readSubscriber>` request gets destination entities associated with the provided IMSIs and/or MSISDNs.

Request

The `<readSubscriber>` request allows for the display of either IMSI, MSISDN, or combinations of IMSI and MSISDN routing entity(s).

```
<readSubscriber ent="entityName" ns="namespace" [resonly="resonly"] [id="id"]>
[
    <imsi>imsi</imsi>
    …
    <imsi>imsi</imsi>
]
[
    <msisdn>msisdn</msisdn>
    …
    <msisdn>msisdn</msisdn>
]
</readSubscriber>
```

Parameters:

```
entityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

resonly(Optional) Indicates whether the response should consist of  the result only,
 without including
the original request in the response.
Values: y only provide the result, do NOT include the original request (default)
ninclude the original request in the response

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295.
```

```
imsi(Optional – see List item.) An IMSI (specified in E.212 format).
Values: 10 to 15 numeric digits.

msisdn  (Optional – List item.) An MSISDN (specified in E.164 international public
  telecommunication numbering plan format).
Values: 8 to 15 numeric digits.
```

Semantic Rules:

1. There must be a minimum of 1 routing entity (IMSI or MSISDN)
2. There must be no more than 10 routing entities (IMSI, MSISDN, or combinations of the two)

Response

The `<readSubscriberResp>` response returns the result of the request to read subscriber routing entities. Only those subscriber routing entities that are found are returned.

Each destination entity (such as `<ltehss>`) are only present in the response if the destination name is set for the routing entity.

```
<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>
```

Parameters:

```
id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 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.
Values: A string with 1 to 4096 characters.

errorError code. Whether or not operation was successfully executed by the XDS. See

Table 10:  Common Error Codes for generic values, and the table below for values
specific to
this request.
Values: 0success
non zerofailure

affected  The total number of entities returned within the <rset> (of type either
<imsi> or <msisdn>),
which, if the request was successful, should correspond to the number of routing
entities provided.
Values: 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.
Values: A string with 1 to 1024 characters.

imsiAn IMSI (specified in E.212 format).
Values: 10 to 15 numeric digits.

msisdn  An MSISDN (specified in E.164 international public telecommunication numbering
 plan format).
Values: 8 to 15 numeric digits.

imshssThe name of the IMS HSS for which the specified routing entity is associated.
Values: A string with 1 to 32 characters.

ltehssThe name of the LTE HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

pcrfThe name of the PCRF for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.
```

```
ocsThe name of the OCS for which the specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.

ofcsThe name of the OFCS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

aaaThe name of the AAA server for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

userdef1The name of the first user defined destination for which the specified
routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

userdef2The name of the second user defined destination for which the specified
routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.
```

Semantic Rules:

1. If the request is successful (i.e. error = 0) then one row entity will be returned for every routing entity found.
2. The affected field will indicate how many entities/rows have been returned.
3. The <rset> entity is only present if there are 1 or more row entities.

Table 17: Read Subscriber Response Error Codes

Error Code	Description
SUCCESS	The read request was successfully completed.
TOO_MANY_ADDR	Too many address values supplied.
NO_ADDR_VAL	No address value supplied.
IMSI_NOT_FOUND	The specified IMSI does not exist.
MSISDN_NOT_FOUND	The specified MSISDN does not exist.

Update Subscriber NAI

The <updateSubscriberNai> request provisions NAI routing data.

Request

The `<updateSubscriberNai>` request allows for the provisioning of NAI values to be associated with 8 different destinations.

A destination name can be specified as “none” which will remove the association of that destination from the specified routing entity(s).

The update command will both update existing routing entities, and it will create any new routing entities that do not exist.

```
<updateSubscriberNai ent="entityName" ns="namespace" [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>
```

Parameters:

```
e
ntityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

resonly(Optional) Indicates whether the response should consist of  the result only,
 without including
the original request in the response.
Values: y only provide the result, do NOT include the original request (default)
ninclude the original request in the response

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 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. Valid values are 0 (return
 immediately if
not available) to 3600 seconds (default is 0). Note: if the request is being performed
 within a
transaction, this parameter will have no effect, as the client already has a
transaction open.
Values: 0 (return immediately if not available) to 3600 seconds (default is 0).

host(See List item.) A host name.
Values: A string with 1 to 64 characters.

user  (See List item.) A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.

imshss(Optional – see List item.) The name of the IMS HSS for which the specified
routing entity(s) are
to be associated.
Values: A string with 1 to 32 characters.
```

```
ltehss(Optional – see List item.) The name of the LTE HSS for which the specified
routing entity(s) are
to be associated.
Values: A string with 1 to 32 characters.

pcrf(Optional – see List item.) The name of the PCRF for which the specified routing
 entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

ocs(Optional – see List item.) The name of the OCS for which the specified routing
 entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

ofcs(Optional – see List item.) The name of the OFCS for which the specified routing
 entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

aaa(Optional – see List item.) The name of the AAA server for which the specified
routing entity(s) are
to be associated.
Values: A string with 1 to 32 characters.

userdef1(Optional – see List item.) The name of the first user defined destination
 for which the
specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.

userdef2(Optional – see List item.) The name of the second user defined destination
 for which the
specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.
```

Semantic Rules:

1. There must be between 1 and 10 user names specified
2. There must be at least one destination specified
3. The host name must already exist in the database
4. A destination name must already exist in the database
5. Each destination name type may only be specified once
6. All specified routing entities will be provisioned with the same destination value(s)
7. Any existing destination(s) for a routing entity will not be changed/removed if not specified in the request
8. Specifying a destination name of "none" will remove the association of that destination from the specified routing entity(s).

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 an NAI is updated with the same destination values that already exist, this may result in `NO_UPDATES` being returned, which is not treated as an error. When a subscriber record is not updated, this means that the affected rows count is not incremented for that NAI.

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

Parameters:

```
id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295

originalXMLRequest(Optional) The text of the original <updateSubscriberNai> XML
request that was sent.
Note: this is only present if the resonly=”n” attribute is set in the original
request.
Values: A string with 1 to 4096 characters.

errorError code. Whether or not operation was successfully executed by the XDS. See

Table 10:  Common Error Codes for generic values, and the table below for values
specific to
this request.
Values: 0success
non zerofailure

affected  The number of routing entities created/updated. Note: if a routing entity
 is updated with the
same destination(s), causing no change to be made, the affected value will not be
incremented.
Values: 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.
Values: A string with 1 to 1024 characters.
```

Table 18: Update Subscriber NAI Response Error Codes

Error Code	Description
SUCCESS	The update request was successfully completed.
DEST_NOT_FOUND	Destination name does not exist.
TOO_MANY_NAI	Too many NAI values supplied.
NO_DEST_VAL	No destination name supplied.
NO_NAI_VAL	No NAI value supplied.
HOST_NOT_FOUND	Host name does not exist.

Delete Subscriber NAI

The `<deleteSubscriberNai>` request removes NAI routing data.

Request

The <deleteSubscriberNai> request allows for the removal of NAI routing entity(s).

```
<deleteSubscriberNai ent="entityName" ns="namespace" [resonly="resonly"] [id="id"]
                     [timeout="timeout"]>
    <host>host</host>
    <user>user</user>
 [
    <user>user</user>
    …
    <user>user</user>
]
</deleteSubscriberNai>
```

Parameters:

```
entityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

resonly(Optional) Indicates whether the response should consist of  the result only,
 without including
the original request in the response.
Values: 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.
Values: 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. Valid values are 0 (return immediately if not
available) to
3600 seconds (default is 0). Note: if the request is being performed within a
transaction, this
parameter will have no effect, as the client already has a transaction open.
Values: 0 (return immediately if not available) to 3600 seconds (default is 0).

host(See List item.) A host name.
Values: A string with 1 to 64 characters.

user  (See List item.) A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.
```

Semantic Rules:

1. There must be between 1 and 10 user names specified
2. The host name must already exist in the database

Response

The `<deleteSubscriberNaiResp>` response returns the result of the request to delete subscriber routing entities. If any failure response is returned, then no routing entity(s) were deleted.

If only one NAI is supplied in a request, and the NAI does not exist, or all NAIs provided in the request do not exist, then this is not treated as an error, and the request will return with `NO_UPDATES`. When a subscriber record does not exist, this means that the affected rows count is not incremented for that NAI.

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

Parameters:

```
id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295

originalXMLRequest(Optional) The text of the original <deleteSubscriberNai> XML
request that was sent.
Note: this is only present if the resonly=”n” attribute is set in the original
request.
Values: A string with 1 to 4096 characters.

errorError code. Whether or not operation was successfully executed by the XDS. See

Table 10:  Common Error Codes for generic values, and the table below for values
specific to
this request.
Values: 0success
non zerofailure

affected  The number of routing entities deleted.
Values: 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.
Values: A string with 1 to 1024 characters.
```

Table 19: Delete Subscriber NAI Response Error Codes

Error Code	Description
SUCCESS	The delete request was successfully completed.
TOO_MANY_NAI	Too many NAI values supplied.
NO_NAI_VAL	No NAI value supplied.

Error Code	Description
HOST_NOT_FOUND	Host name does not exist.

Read Subscriber NAI

The `<readSubscriberNai>` request gets destination entities associated with the provided NAIs.

Request

The `<readSubscriberNai>` request allows for the display of NAI routing entity(s).

```
<readSubscriberNai ent="entityName" ns="namespace" [resonly="resonly"] [id="id"]>
    <host>host</host>
    <user>user</user>
 [
    <user>user</user>
    …
    <user>user</user>
]
</readSubscriberNai>
```

Parameters:

```
entityNameThe entity name within the global schema.
Values: subscriberRouting Entity name is always for subscriber routing

namespaceThe namespace within the global schema.
Values: dsr Namespace is always for the DSR.

resonly(Optional) Indicates whether the response should consist of  the result only,
 without including
the original request in the response.
Values: 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.
Values: 1-4294967295.

host(See List item.) A host name.
Values: A string with 1 to 64 characters.

user  (See List item.) A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.
```

Semantic Rules:

1. There must be between 1 and 10 user names specified
2. The host name must already exist in the database

Response

The `<readSubscriberNaiResp>` response returns the result of the request to read subscriber routing entities. Only those subscriber routing entities that are found are returned.

Each destination entity (such as <ltehss>) are only present in the response if the destination name is set for the routing entity.

```
<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>
```

Parameters:

```
id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 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.
Values: A string with 1 to 4096 characters.

errorError code. Whether or not operation was successfully executed by the XDS. See

Table 10:  Common Error Codes for generic values, and the table below for values
specific to
this request.
Values: 0success
non zerofailure
```

```
affected  The total number of entities returned within the <rset> (of type either
<imsi> or <msisdn>),
which, if the request was successful, should correspond to the number of routing
entities provided.
  Values: 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.
Values: A string with 1 to 1024 characters.

hostA host name.
Values: A string with 1 to 64 characters.

user  A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.

ishssThe name of the IMS HSS for which the specified routing entity is associated.
Values: A string with 1 to 32 characters.

ltehssThe name of the LTE HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

pcrfThe name of the PCRF for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

ocsThe name of the OCS for which the specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.

ofcsThe name of the OFCS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

aaaThe name of the AAA server for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

userdef1The name of the first user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

userdef2The name of the second user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.
```

Semantic Rules:

1. If the request is successful (i.e. error = 0) then one row entity will be returned for every routing entity found.
2. The affected field will indicate how many entities/rows have been returned.
3. The `<rset>` entity is only present if there are 1 or more row entities.

Table 20: Read Subscriber NAI Response Error Codes

Error Code	Description
SUCCESS	The delete request was successfully completed.

Error Code	Description
TOO_MANY_NAI	Too many NAI values supplied.
NO_NAI_VAL	No NAI value supplied.
NAI_NOT_FOUND	The specified NAI does not exist.
HOST_NOT_FOUND	Host name does not exist.

Block Transactions

A block transaction allows the user to group a number of requests within a transaction and send them as one "chunk" of data. Requests will be executed when the whole "chunk" has been sent.

A "chunk" consists of the block transaction tags, with a number of requests contained within it.

It is possible to select if the result to each request is included in the response or not, by use of the `resonly` attribute in the `<tx>` entity. The selection (even the default when not included) is applied to every request within the block transaction. If any request itself sets the resonly attribute, this is overridden with the value from the block transaction.

Note: 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

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

    …

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

Parameters:

```
resonly(Optional) Indicates whether the response should consist of  the result only,
 without
including the original request in the response.
```

```
Values: y only provide the result, do NOT include the original request (default)
ninclude the original request in the response
```

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.

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

    …

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

Parameters:

```
nbreqNumber of requests within the transaction. The response will contain responses
 (and optionally the
requests themselves) for each and every request.
Values: 0-50.

id(Optional) Transaction id value provided in request, and will be passed back in
the response.
Values: 1-4294967295.
```

Table 21: Block Transaction Response 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

Error Code	Description
	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)

Example Sessions

The following sections contain example usages of the XDS. All scenarios assume that a TCP/IP connection has already been established between the client and the XDS. 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.

IMSI/MSISDN Routing Entity Creation

These examples create IMSI and MSISDN routing entities that will be needed for all subsequent examples.

Table 22: Create IMSI/MSISDN Routing Entity Message Flow Example

Message		Description
CPS—>XDS	`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 subscriber routing entities - 3 IMSIs and 2 MS>ISDNs with an LTE HSS and AAA server destinations. **Note:** request is made to include the original request in the response.
CPS<—XDS	`<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>`	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—>XDS	`<updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000106</imsi> <msisdn>15634210106</msisdn> <ltehss>LTE_HSS_5</ltehss> </updateSubscriber>`	Request to update existing IMSI and MSISDN subscriber routing entities with a new LTE HSS value.
CPS<—XDS	`<updateSubscriberResp> <res error="0" affected="2"/> </updateSubscriberResp>`	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—>XDS	`<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.
CPS<—XDS	`<updateSubscriberResp> <res error="2006" affected="0"/> </updateSubscriberResp>`	Request fails, as the destination does not exist.

IMSI/MSISDN Routing Entity Updating

These examples update existing IMSI and MSISDN routing entities.

Table 23: Update IMSI/MSISDN Routing Entity Message Flow Example

Message		Description

CPS—>XDS	`<updateSubscriber ent="subscriberRouting" ns="dsr">`
	`<imsi>310910421000309</imsi>`
	`<ltehss>LTE_HSS_5</ltehss>`
	`</updateSubscriber>`
		Request to update existing IMSI subscriber routing entity with a new LTE HSS value.

CPS<—XDS	`<updateSubscriberResp> <res error="0" affected="1"/> </updateSubscriberResp>`	Response to update subscriber routing entity - success. Affected rows = 1 (as 1 entry for an IMSI was updated with a new LTE HSS value).

CPS—>XDS	`<updateSubscriber ent="subscriberRouting" ns="dsr" id="2231845009">`
	`<msisdn>15634210106</msisdn>`
	`<aaa>none</aaa> </updateSubscriber>`
		Request to update existing MSISDN subscriber routing entity, and remove the AAA destination association. Transaction id value passed in request.

CPS<—XDS	`<updateSubscriberResp id="2231845009">`
	`<res error="0" affected="1"/>`
	`</updateSubscriberResp>`
		Response to update subscriber routing entity - success. Affected rows = 1 (as 1 entry for an IMSI was updated to remove the AAA destination). Transaction id supplied in request is returned in response.

CPS—>XDS	`<updateSubscriber ent="subscriberRouting" ns="dsr">`
	`<imsi>310910421000106</imsi>`
	`<ltehss>BAD_VALUE</ltehss>`
	`</updateSubscriber>`
		Request to update existing IMSI subscriber routing entities with an invalid LTE HSS value.

CPS<—XDS	`<updateSubscriberResp> <res error="2006" affected="0"/>`
	`</updateSubscriberResp>`
		Request fails, as the destination does not exist.

IMSI/MSISDN Routing Entity Deletion

These examples delete existing IMSI and MSISDN routing entities.

Table 24: Delete IMSI/MSISDN Routing Entity Message Flow Example

Message		Description

CPS—>XDS	`<deleteSubscriber ent="subscriberRouting" ns="dsr">`
	`<imsi>310910421000307</imsi>`
	`</deleteSubscriber>`
		Request to delete an existing IMSI subscriber routing entity.
		Response to delete subscriber routing entity - success. Affected rows = 1 (as 1 entry for an IMSI was deleted).

CPS<—XDS	`<deleteSubscriberResp> <res error="0" affected="1"/> </deleteSubscriberResp>`	

CPS—>XDS	`<deleteSubscriber ent="subscriberRouting" ns="dsr">`
	`<msisdn>15634210107</msisdn>`
	`</deleteSubscriber>`
		Request to delete an existing MSISDN subscriber routing entity.
		Response to delete subscriber routing entity - success. Affected rows = 1 (as 1 entry for an MSISDN was deleted).

CPS<—XDS	`<deleteSubscriberResp> <res error="0" affected="1"/> </deleteSubscriberResp>`	

CPS—>XDS	`<deleteSubscriber ent="subscriberRouting" ns="dsr">`
	`<imsi>310910421000302</imsi>`
	`</deleteSubscriber>`
		Request to delete an IMSI subscriber routing entity that does not exist.
		Request fails, as the IMSI does not exist.

CPS<—XDS	`<deleteSubscriberResp> <res error="2007" affected="0"/>`
	`</deleteSubscriberResp>`	

NAI Routing Entity Creation

These examples create NAI routing entities that will be needed for all subsequent examples.

Table 25: Create NAI Routing Entity Message Flow Example

Message		Description

CPS—>XDS	`<updateSubscriberNai`
	`ent="subscriberRouting"`
	`    ns="dsr">`
	`<host>operator.com</host>`
	`<user>john.smith</user>`
	`<user>peter.black</user>`
	`<user>simon.wells</user>`
	`<user>andrew.green</user>`
	`<ltehss>LTE_HSS_2</ltehss>`
	`<aaa>AAA_4</aaa> </updateSubscriberNai>`
		Request to create 4 NAI subscriber routing entities with an LTE HSS and AAA server destinations.
		Response to create subscriber routing entities - success. Affected rows = 4 (as 4 new NAI entries created.

CPS<—XDS	`<updateSubscriberNaiResp>    <res`
	`error="0" affected="4"/>`
	`</updateSubscriberNaiResp>`	

CPS—>XDS	`<updateSubscriberNai`
	`ent="subscriberRouting"`
	`    ns="dsr">`
	`<host>operator.com</host>`
	`<user>william.walters</user>`
	`<ltehss>LTE_HSS_2</ltehss>`
	`<aaa>NOT_VALID</aaa>`
	`</updateSubscriberNai>`
		Request to create an NAI subscriber routing entitiy with an invalid AAA destination value.
		Request fails, as the destination does not exist.

CPS<—XDS	`<updateSubscriberNaiResp>    <res`
	`error="11" affected="0"/>`
	`</updateSubscriberNaiResp>`	

CPS—>XDS	`<updateSubscriberNai`
	`ent="subscriberRouting"`
	`    ns="dsr">`
	`<host>INVALID_HOST</host>`
	`<user>jane.green</user>`
	`<ltehss>LTE_HSS_2</ltehss>`
	`</updateSubscriberNai>`
		Request to create an NAI subscriber routing entitiy with an invalid host name.
		Request fails, as the host name does not exist.

CPS<—XDS	`<updateSubscriberNaiResp>    <res`
	`error="2010" affected="0"/>`
	`</updateSubscriberNaiResp>`	

NAI Routing Entity Updating

These examples update existing NAI routing entities.

Table 26: Update NAI Routing Entity Message Flow Example

Message		Description
CPS—>XDS	`<updateSubscriberNai ent="subscriberRouting" ns="dsr"> <host>operator.com</host> <user>john.smith</user> <pcrf>PCRF_1</pcrf> <ocs>OCS_4</ocs> <aaa>AAA_2</aaa> </updateSubscriberNai>`	Request to update an NAI subscriber routing entity, adding PCRF and OCS values, and updating the existing AAA destination value.
CPS<—XDS	`<updateSubscriberNaiResp> <res error="0" affected="1"/> </updateSubscriberNaiResp>`	Response to create subscriber routing entities - success. Affected rows = 1 (as 1 NAI entry was updated).
CPS—>XDS	`<updateSubscriberNai ent="subscriberRouting" ns="dsr" id="2231846099"> <host>operator.com</host> <user>andrew.green</user> <aaa>none</aaa> </updateSubscriberNai>`	Request to update existing NAI subscriber routing entity, and remove the AAA destination association. Transaction id value passed in request.
CPS<—XDS	`<updateSubscriberNaiResp id="2231846099"> <res error="0" affected="1"/> </updateSubscriberNaiResp>`	Response to update subscriber routing entity - success. Affected rows = 1 (as 1 entry for an NAI was updated to remove the AAA destination). Transaction id supplied in request is returned in response.
CPS—>XDS	`<updateSubscriberNai ent="subscriberRouting" ns="dsr"> <host>operator.com</host> <user>john.smith</user> <pcrf>NOT_VALID</pcrf> </updateSubscriberNai>`	Request to update existing NAI subscriber routing entities with an invalid PCRF value.
CPS<—XDS	`<updateSubscriberNaiResp> <res error="2006" affected="0"/> </updateSubscriberNaiResp>`	Request fails, as the destination does not exist.

NAI Routing Entity Deletion

These examples delete existing NAI routing entities.

Table 27: Delete NAI Routing Entity Message Flow Example

Message		Description

CPS—>XDS	`<deleteSubscriberNai`
	`ent="subscriberRouting"`
	`    ns="dsr">`
	`<host>operator.com</host>`
	`<user>peter.black</user>`
	`<user>simon.wells</user>`
	`</deleteSubscriberNai>`
		Request to delete 2 existing NAI subscriber routing entities.
		Response to delete NAI subscriber routing entities - success. Affected rows = 2 (as 2 entries for NAIs were deleted).

CPS<—XDS	`<deleteSubscriberNaiResp>    <res`
	`error="0" affected="2"/>`
	`</deleteSubscriberNaiResp>`	

CPS—>XDS	`<deleteSubscriberNai`
	`ent="subscriberRouting"`
	`    ns="dsr">`
	`<host>operator.com</host>`
	`<user>john.smith</user>`
	`<user>richard.daniels</user>`
	`</deleteSubscriberNai>`
		Request to delete 2 NAI subscriber routing entities. One NAI value exists, the other does not.
		Request is successful. The entity which does exist is deleted, and the number of affected rows is 1.

CPS<—XDS	`<deleteSubscriberNaiResp>    <res`
	`error="0" affected="1"/>`
	`</deleteSubscriberNaiResp>`	

IMSI/MSISDN Routing Entity Reading

These examples read IMSI and MSISDN routing entities.

Table 28: Read IMSI/MSISDN Routing Entity Message Flow Example

Message		Description

CPS—>XDS	`<readSubscriber ent="subscriberRouting"`
	`               ns="dsr">`
	`<imsi>310910421000106</imsi>`
	`<msisdn>15634210106</msisdn>`
	`</readSubscriber>`
		Request to read 2 subscriber routing entities (1 IMSI and 1 MSISDN).
		Response to read subscriber routing entities - success. Affected rows = 2 (as 1 IMSI and 1 MSISDN row are returned). IMSI row has an LTE HSS and AAA destination set. MSISDN row has just an LTE HSS destination set.

CPS<—XDS	`<readSubscriberResp>    <res error="0"`
	`affected="2"/>    <rset>        <imsi`
	`imsi=”310910421000106”>`
	`<ltehss>LTE_HSS_5</ltehss>`
	`<aaa>AAA_4</aaa>        </imsi>`
	`<msisdn msisdn=”15634210106”>`
	` <ltehss>LTE_HSS_5</ltehss>`
	`</msisdn>    </rset>`
	`</readSubscriberResp>`	

CPS—>XDS	`<readSubscriber ent="subscriberRouting"`
	`               ns="dsr">`
	`<imsi>310910421000107</imsi>`
	`</readSubscriber>`
		Request to read an IMSI subscriber routing entity that does not exist.
		Request fails, as the IMSI does not exist.

CPS<—XDS	`<readSubscriberResp>    <res error="2007"`
	` affected="0"/> </readSubscriberResp>`	

NAI Routing Entity Reading

These examples read NAI routing entities.

Table 29: Read NAI Routing Entity Message Flow Example

Message		Description
CPS—>XDS	`<readSubscriberNai ent="subscriberRouting" ns="dsr"> <host>operator.com</host> <user>john.smith</user> <user>andrew.green</user> </readSubscriberNai>`	Request to read 2 NAI subscriber routing entities. Response to read subscriber routing entities - success. Affected rows = 2 (First NAI row has an LTE HSS, PCRF, OCS and AAA destination set. Second NAI row has just an LTE HSS destination set.
CPS<—XDS	`<readSubscriberNaiResp> <res error="0" affected="2"/> <rset> <nai host="operator.com" user="john.smith"> <ltehss>LTE_HSS_2</ltehss> <pcrf>PCRF_1</pcrf> <ocs>OCS_4</ocs> <aaa>AAA_2</aaa> </nai> <nai host="operator.com" user="andrew.green"> <ltehss>LTE_HSS_2</ltehss> </nai> </rset> </readSubscriberNaiResp>`	
CPS—>XDS	`<readSubscriberNai ent="subscriberRouting" ns="dsr"> <host>operator.com</host> <user>peter.black</user> </readSubscriberNai>`	Request to read an NAI subscriber routing entity that does not exist. Request fails, as the NAI does not exist.
CPS<—XDS	`<readSubscriberNaiResp> <res error="2009" affected="0"/> </readSubscriberNaiResp>`	

Standard Transaction Committed

This example issues a number of different requests within the transaction which is then committed successfully.

Table 30: Standard Transaction Committed Message Flow Example

Message		Description
CPS—>XDS	`<startTransaction/>`	Request to start a transaction.
CPS<—XDS	`<startTransactionResp>    <res error="0" affected="0"/> </startTransactionResp>`	Response to start transaction - success.

CPS—>XDS	`<updateSubscriber ent="subscriberRouting" ns="dsr">`
	`<imsi>310910421000444</imsi>`
	`<msisdn>15634210444</msisdn>`
	`<ltehss>LTE_HSS_1</ltehss>`
	`</updateSubscriber>`
		Request to update an IMSI and MSISDN - success.

CPS<—XDS	`<updateSubscriberResp>    <res error="0" affected="2"/> </updateSubscriberResp>`	

CPS—>XDS	`<updateSubscriber ent="subscriberRouting" ns="dsr">`
	`<imsi>310910421000555</imsi>`
	`<msisdn>15634210555</msisdn>`
	`<ltehss>LTE_HSS_2</ltehss>`
	`</updateSubscriber>`
		Request to update an IMSI and MSISDN - success.

CPS<—XDS	`<updateSubscriberResp>    <res error="0" affected="2"/> </updateSubscriberResp>`	

CPS—>XDS	`<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<—XDS	`<updateSubscriberNaiResp>    <res`
	`error="0" affected="1"/>`
	`</updateSubscriberNaiResp>`	

CPS—>XDS	`<updateSubscriberNai`
	`ent="subscriberRouting"`
	`    ns="dsr">`
	`<host>operator.com</host>`
	`<user>jennifer.green</user>`
	`<ltehss>LTE_HSS_2</ltehss>`
	`</updateSubscriberNai>`
		Request to update an NAI - success.

CPS<—XDS	`<updateSubscriberNaiResp>    <res`
	`error="0" affected="1"/>`
	`</updateSubscriberNaiResp>`	

CPS—>XDS	`<commit/>`	Request to commit the transaction.

Message		Description
CPS<—XDS	`<commitResp>    <res error="0" affected="0"/> </commitResp>`	Response to commit transaction - success. All updates were successfully performed

Standard Transaction Rolled Back

This example authenticates to the XDS, and then issues a number of different requests within the transaction which is then rolled back.

Table 31: Standard Transaction Rolled Back Message Flow Example

Message		Description
CPS—>XDS	`<startTransaction timeout="10"> </startTransaction>`	Request to start a transaction. Response to start transaction - success.
CPS<—XDS	`<startTransactionResp> <res error="0" affected="0"/> </startTransactionResp>`	
CPS—>XDS	`<updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000666</imsi> <msisdn>15634210666</msisdn> <ltehss>LTE_HSS_1</ltehss> </updateSubscriber>`	Request to update an IMSI and MSISDN - success.
CPS<—XDS	`<updateSubscriberResp> <res error="0" affected="2"/> </updateSubscriberResp>`	
CPS—>XDS	`<updateSubscriber ent="subscriberRouting" ns="dsr"> <imsi>310910421000777</imsi> <msisdn>15634210777</msisdn> <ltehss>LTE_HSS_2</ltehss> </updateSubscriber>`	Request to update an IMSI and MSISDN - success.
CPS<—XDS	`<updateSubscriberResp> <res error="0" affected="2"/> </updateSubscriberResp>`	
CPS—>XDS	`<updateSubscriberNai ent="subscriberRouting" ns="dsr"> <host>operator.com</host> <user>david.leno</user> <ltehss>LTE_HSS_1</ltehss> </updateSubscriberNai>`	Request to update an NAI - success.
CPS<—XDS	`<updateSubscriberNaiResp> <res error="0" affected="1"/> </updateSubscriberNaiResp>`	
CPS—>XDS	`<updateSubscriberNai ent="subscriberRouting" ns="dsr"> <host>operator.com</host> <user>rebecca.wilson</user> <ltehss>INVALID_VALUE</ltehss> </updateSubscriberNai>`	Request to create an NAI. Creation of NAI fails because LTE HSS destination name is not found
CPS<—XDS	`<updateSubscriberNaiResp> <res error="2006" affected="0"/> </updateSubscriberNaiResp>`	
CPS—>XDS	`<rollback/>`	

Message		Description
CPS<—XDS	`<rollbackResp> <res error="0" affected="0"/> </rollbackResp>`	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.

Block Transaction Committed

This example issues a number of different requests within a block transaction, for which all are successful and is therefore automatically committed.

Table 32: Block Transaction Committed Message Flow Example

Message		Description
CPS—>XDS	<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> <ltehss>LTE_HSS_6</ltehss> </updateSubscriber> <updateSubscriber ent="subscriberRouting" ns="dsr"> <msisdn>15634210110</msisdn> <ltehss>LTE_HSS_6</ltehss> </updateSubscriber> </tx>	A single request is sent contain 4 different updateSubscriber requests. Response indicates that 4 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 responese).
CPS<—XDS	<txResp nbreq="4"> <updateSubscriberResp> <res error="0" affected="1"/> </updateSubscriberResp> <updateSubscriberResp> <res error="0" affected="1"/> </updateSubscriberResp> <updateSubscriberResp> <res error="0" affected="1"/> </updateSubscriberResp> <updateSubscriberResp> <res error="0" affected="1"/> </updateSubscriberResp> </txResp>	

Block Transaction Rolled Back

This example issues a number of different requests within a block transaction, for which on fails, and therefore the transaction is automatically rolled back.

Table 33: Block Transaction Rolled Back Message Flow Example

Message		Description

CPS—>XDS	`< 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 contain 4 different updateSubscriber requests. The request is made to include each request in the response for the entire transaction (indicated by the resonly="n" attribute).
		
		Response indicates that 4 requests were within the transaction. The third request failed because the destination does not exist. Hence, all requests fail, and all are automatically rolled back.
		
		Note here that 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<—XDS	`<<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`
	`error="2006" affected="0"/>`
	`</updateSubscriberResp>`
	`<updateSubscriberResp>`
	`<updateSubscriber`
	`ent="subscriberRouting"`
	`    ns="dsr">`
	`<msisdn>15634210112</msisdn>`	

Message		Description
	`<ltehss>LTE_HSS_6</ltehss> </updateSubscriber> <res error="1" affected="0"/> </updateSubscriberResp> </txResp>`	

Chapter 5

SOAP Operations Definitions

Topics:

This chapter describes the SOAP operations syntax and parameters.

The SOAP operations operate under the same conditions/restrictions as their equivalent XML requests as detailed in *XML Message Definitions*, and so these conditions are not repeated here.

Transaction Id

SOAP operations are synchronous requests. i.e. the client sends a request and must wait to get a response before sending another request. Hence a transaction id to correlate a response to a request is not required.

Response Messages

A SOAP response message is sent by the SOAP Server in response to a SOAP request, and each response is specific to each request. Each and every response contains a result SOAP message, which is an XML data structure of type sdsResult as defined in the XSD file (see *XML Schema Definition Files (XSD)*).

The `result` message contains the following:

1. error – the success/failure error code relating to the request made (see *SDS Response Message Error Codes*)
2. affected – the number of rows affected by the request
3. description – an optional text string which may provide additional information as to why a request failed, for example which field was invalid, and why

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

Parameters:

```
errorError code. Whether or not operation was successfully executed by the SDS.
Values: 0success
non zerofailure (see SDS Response Message Error Codes)

affected  The number of routing entities affected.
  Values: 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.
Values: A string with 1 to 1024 characters.
```

Common Error Codes

The common error codes are the same as on the XML interface, as defined in *Common Error Codes*.

List of Request Operations

The SOAP Server supports the SDS related SOAP operations listed in *Table 34: Supported SDS SOAP Operations*, and the DSR related SOAP operations listed in *Table 35: Supported DSR SOAP Operations*.

Table 34: Supported SDS SOAP Operations

Operation Name	Description	Section
`startTransaction`	Start Database Transaction	*Start Transaction*
`commit`	Commit Database Transaction	*Commit Transaction*
`rollback`	Abort Database Transaction	*Rollback Transaction*

Table 35: Supported DSR SOAP Operations

Operation Name	Description	
`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*

SOAP URLs for Operations

SOAP operations on the SDS are sent using SOAP over HTTP (for unsecure) and SOAP over HTTPS (for secure). The URL format for accessing the SOAP operations is as follows :

For an unsecure (TCP) interface:

```
http://ipAddress:soapPort/<path>/operationName
```

For an secure (SSL) interface:

```
https://ipAddress:soapPort/<path>/operationName
```

Parameters:

```
ipAddress The TCP/IP address for the SOAP interface. This should be the Primary
System Controller s VIP.
Values: 10 to 15 numeric digits.

soapPort The TCP/IP port for the SOAP interface. This is configured via the SDS GUI.
Values: 1 - 65535.

path The path relating to the operation.
Values: For SDS operations (see Table 34: Supported SDS SOAP Operations) “sds”.
For DSR operations (see Table 35: Supported DSR SOAP Operations) “sds/dsr”.

operationName The operation name as listed in Table 35: Supported DSR SOAP Operations.
Values: A string with 1 to 32 characters.
```

For example to access the update subscriber NAI operation using an unsecure connection:

```
http://10.50.1.102:5678/sds/dsr/updateSubscriberNai
```

And to start a transaction using a secure connection:

```
https://10.50.1.102:5678/sds/startTransaction
```

Start Transaction

The SDS `startTransaction` SOAP operation is used to begin a database transaction.

Input

The `startTransactionRequest` message is sent to begin a database transaction. The message contains one parameter part `startTransactionRequest`.

The `startTransactionRequest` is an integer, which indicates 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 `startTransaction` requests were received. Valid values are 0 (return immediately if not available) to3600 seconds (default is 0).

Output

The `startTransactionResponse` message returns the result of the request to begin a database transaction.

The response message contains a `startTransactionResponse` part which is a result element as described in *Response Messages*. 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.

Possible error codes are the same as for the XML request and are listed in *Response*.

Commit Transaction

The SDS `commit` SOAP operation is used to commit a database transaction.

Input

The `commitRequest` message is sent to commit a database transaction. The message contains one parameter part `startTransactionRequest`. This parameter is a place holder and should be left as an empty string.

If the currently opened transaction **has one or more successful updates**, then committing the transaction will cause all the database changes to be committed. It is very important to understand that all previous updates, even though they received a successful error code, are not committed to the database until the `commit` request is received.

Output

The commitResponse message returns the result of the request to commit a database transaction.

The response message contains a commitResponse part which is a result element as described in *Response Messages*. 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 will cause the transaction to end regardless of whether any updates were actually made to the database.

Possible error codes are the same as for the XML request and are listed in *Response*.

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.

Rollback Transaction

The SDS `rollback` SOAP operation is used to abort a database transaction.

Input

The `rollbackRequest` message is sent to abort a database transaction. This message contains one parameter part `rollbackRequest`. This parameter is a place holder and should be left as an empty string.

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

Output

The `rollbackResponse` message returns the result of the request to abort a database transaction.

The response message contains a `rollbackResponse` part which is a result element as described in *Response Messages*. 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.

Possible error codes are the same as for the XML request and are listed in *Response*.

Update Subscriber

The DSR `updateSubscriber` SOAP operation provisions IMSI and MSISDN routing data.

Input

The `updateSubscriberRequest` message is sent to provision either IMSI, MSISDN, or combinations of IMSI and MSISDNs to be associated with 8 different destinations. The message contains one part `updateSubscriberRequest`, which consists of a timeout attribute, and two elements; `addressList` and `destinationList`.

The `timeout` is an integer, which indicates 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. Valid values are 0 (return immediately if not available) to 3600 seconds (default is 0). Note: if the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.

The `addressList` is an XML data structure of type `dsrAddressList` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- A sequence of :
    - 0-10 IMSI elements
    - 0-10 MSISDN elements

The syntax of addressList is as follows:

```
[
    <imsi>imsi</imsi>
    …
    <imsi>imsi</imsi>
]
[
    <msisdn>msisdn</msisdn>
    …
    <msisdn>msisdn</msisdn>
]
```

Parameters:

```
imsiAn IMSI (specified in E.212 format).
Values: 10 to 15 numeric digits.

msisdn  An MSISDN (specified in E.164 international public telecommunication numbering
 plan format).
Values: 8 to 15 numeric digits.
```

The `destinationList` is an XML data structure of type `dsrDestinationList` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- An optional IMS HSS name
- An optional LTE HSS name
- An optional PCRF name
- An optional OCS name
- An optional OFCS name
- An optional AAA server name
- An optional user defined destination name (#1)
- An optional user defined destination name (#2)

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

The syntax of destinationList is as follows:

```
[<imshss>imshss</imshss>       ]
[<ltehss>ltehss</ltehss>       ]
[<pcrf>pcrf</pcrf>             ]
[<ocs>ocs</ocs>                ]
[<ofcs>ofcs</ofcs>             ]
[<aaa>aaa</aaa>                ]
[<userdef1>userdef1</userdef1> ]
```

```
[<userdef2>userdef2</userdef2> ]
```

Parameters:

```
imshssThe name of the IMS HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

ltehssThe name of the LTE HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

pcrfThe name of the PCRF for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

ocsThe name of the OCS for which the specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.

ofcsThe name of the OFCS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

aaaThe name of the AAA server for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

userdef1The name of the first user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

userdef2The name of the second user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.
```

Output

The `updateSubscriberResponse` message 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.

The response message contains an `updateSubscriberResponse` part which is a result element as described in *Response Messages*. If any failure response is returned, then no updates were made to any routing entity.

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

Possible error codes are the same as for the XML request and are listed in *Response*.

Delete Subscriber

The DSR `deleteSubscriber` SOAP operation deletes IMSI and MSISDN routing data.

Input

The `deleteSubscriberRequest` message is sent to delete either IMSI, MSISDN, or combinations of IMSI and MSISDN routing entity(s). The message contains one part `deleteSubscriberRequest`, which consists of a timeout attribute, and an element `addressList`.

The `timeout` is an integer, which indicates 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. Valid values are 0 (return immediately if not available) to 3600 seconds (default is 0). Note: if the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.

The `addressList` is an XML data structure of type `dsrAddressList` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- A sequence of :
  - 0-10 IMSI elements
  - 0-10 MSISDN elements

The syntax of `addressList` is as follows:

```
[
    <imsi>imsi</imsi>
    …
    <imsi>imsi</imsi>
]
[
    <msisdn>msisdn</msisdn>
    …
    <msisdn>msisdn</msisdn>
]
```

Parameters:

```
imsiAn IMSI (specified in E.212 format).
Values: 10 to 15 numeric digits.

msisdn  An MSISDN (specified in E.164 international public
  telecommunication numbering plan format).
Values: 8 to 15 numeric digits.
```

Output

The `deleteSubscriberResponse` message returns the result of the request to delete subscriber routing entities. There is a single result that applies to all routing entities supplied.

If only one IMSI or MSISDN is supplied in a request, and the IMSI/MSISDN does not exist, or all IMSI/MSISDNs provided in the request do not exist, then this is not treated as an error, and the request will return with `NO_UPDATES`. When a subscriber record does not exist, this means that the affected rows count is not incremented for that IMSI/MSISDN.

The response message contains a `deleteSubscriberResponse` part which is a result element as described in *Response Messages*. If any failure response is returned, then no routing entity(s) were deleted.

Possible error codes are the same as for the XML request and are listed in *Response*.

Read Subscriber

The DSR `readSubscriber` SOAP operation gets destination entities associated with the provided IMSIs and/or MSISDNs.

Input

The `readSubscriberRequest` message is sent to get either IMSI, MSISDN, or combinations of IMSI and MSISDN routing entity(s). The message contains one part `readSubscriberRequest`, which consists of an element addressList.

The `addressList` is an XML data structure of type `dsrAddressList` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- A sequence of :
    - 0-10 IMSI elements
    - 0-10 MSISDN elements

The syntax of `addressList` is as follows:

```
[
    <imsi>imsi</imsi>
    …
    <imsi>imsi</imsi>
]
[
    <msisdn>msisdn</msisdn>
    …
    <msisdn>msisdn</msisdn>
]
```

Parameters:

```
imsiAn IMSI (specified in E.212 format).
Values: 10 to 15 numeric digits.

msisdn  An MSISDN (specified in E.164 international public telecommunication numbering
 plan format).
Values: 8 to 15 numeric digits.
```

Output

The readSubscriberResponse message returns the result of the request to read subscriber routing entities.

The response message contains one part readSubscriberResponse, which consists of two elements; result and resultSet. The result element as described in *Response Messages*. Only those subscriber routing entities that are found are returned.

Possible error codes are the same as for the XML request and are listed in *Response*.

The resultSet element contains a result set containing:

- For each IMSI or MSISDN found, a list of all the provisioned destination names

The resultSet is an XML data structure of type dsrResultSetDestinationsAddress as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- A sequence of 1-10 IMSI or MSISDN elements (based on the order of the requested routing entites)
    - Each IMSI or MSISDN element contains:
        - Attributes of the IMSI or MSISDN requested
        - A list of provisioned destination name entites

The syntax of resultSet is as follows:

```
<<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>>
...
<<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>>
```

Parameters:

```
imsiAn IMSI (specified in E.212 format).
Values: 10 to 15 numeric digits.

msisdn  An MSISDN (specified in E.164 international public
  telecommunication numbering plan format).
Values: 8 to 15 numeric digits.

imshssThe name of the IMS HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

ltehssThe name of the LTE HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

pcrfThe name of the PCRF for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

ocsThe name of the OCS for which the specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.

ofcsThe name of the OFCS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

aaaThe name of the AAA server for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

userdef1The name of the first user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

userdef2The name of the second user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.
```

Update Subscriber NAI

The DSR `updateSubscriberNai` SOAP operation provisions NAI routing data.

Input

The `updateSubscriberNaiRequest` message is sent to provision NAIs to be associated with 8 different destinations. The message contains one part `updateSubscriberNaiRequest`, which consists of a timeout attribute, and two elements; `naiList` and `destinationList`.

The `timeout` is an integer, which indicates 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. Valid values are 0 (return immediately if not available) to 3600 seconds (default is 0). Note: if the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.

The `naiList` is an XML data structure of type `dsrNaiList` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- A hostname
- A sequence of 1-10 username elements

The syntax of `naiList` is as follows:

```
<host>host</host>
<user>user</user>
[
  <user>user</user>
  …
  <user>user</user>
]
```

Parameters:

```
hostA host name.
Values: A string with 1 to 64 characters.

user  A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.
```

The `destinationList` is an XML data structure of type `dsrDestinationList` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- An optional IMS HSS name
- An optional LTE HSS name
- An optional PCRF name
- An optional OCS name
- An optional OFCS name
- An optional AAA server name
- An optional user defined destination name (#1)
- An optional user defined destination name (#2)

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

The syntax of `destinationList` is as follows:

```
[<imshss>imshss</imshss>       ]
[<ltehss>ltehss</ltehss>       ]
[<pcrf>pcrf</pcrf>             ]
[<ocs>ocs</ocs>                ]
[<ofcs>ofcs</ofcs>             ]
[<aaa>aaa</aaa>                ]
[<userdef1>userdef1</userdef1> ]
[<userdef2>userdef2</userdef2> ]
```

Parameters:

```
imshssThe name of the IMS HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.
```

```
ltehssThe name of the LTE HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

pcrfThe name of the PCRF for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

ocsThe name of the OCS for which the specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.

ofcsThe name of the OFCS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

aaaThe name of the AAA server for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

userdef1The name of the first user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

userdef2The name of the second user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.
```

Output

The `updateSubscriberNaiResponse` message 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.

The response message contains an `updateSubscriberNaiResponse` part which is a result element as described in *Response Messages*. If any failure response is returned, then no updates were made to any routing entity.

Note: If an NAI is updated with the same destination values that already exist, this may result in `NO_UPDATES` being returned, which is not treated as an error. When a subscriber record is not updated, this means that the affected rows count is not incremented for that NAI.

Possible error codes are the same as for the XML request and are listed in *Response*.

Delete Subscriber NAI

The DSR `deleteSubscriberNai` SOAP operation deletes NAI routing data.

Input

The `deleteSubscriberNaiRequest` message is sent to delete NAI routing entity(s). The message contains one part `deleteSubscriberNaiRequest`, which consists of a timeout attribute, and an element naiList.

The timeout is an integer, which indicates 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. Valid values are 0 (return immediately if not available) to 3600 seconds (default is 0). Note: if the request is being performed within a transaction, this parameter will have no effect, as the client already has a transaction open.

The `naiList` is an XML data structure of type `dsrNaiList` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- A hostname
- A sequence of 1-10 username elements

The syntax of naiList is as follows:

```
<host>host</host>
<user>user</user>
[
  <user>user</user>
  …
  <user>user</user>
]
```

Parameters:

```
hostA host name.
Values: A string with 1 to 64 characters.

user  A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.
```

Output

The `deleteSubscriberNaiResponse` message returns the result of the request to delete subscriber routing entities.

If only one NAI is supplied in a request, and the NAI does not exist, or all NAIs provided in the request do not exist, then this is not treated as an error, and the request will return with `NO_UPDATES`. When a subscriber record does not exist, this means that the affected rows count is not incremented for that NAI.

The response message contains a `deleteSubscriberNaiResponse` part which is a result element as described in *Response Messages*. If any failure response is returned, then no routing entity(s) were deleted.

Possible error codes are the same as for the XML request and are listed in *Response*.

Read Subscriber NAI

The DSR `readSubscriberNai` SOAP operation gets destination entities associated with the provided NAIs.

Input

The `readSubscriberNaiRequest` message is sent to get NAI routing entity(s). The message contains one part `readSubscriberNaiRequest`, which consists of an element `naiList`.

The `naiList` is an XML data structure of type `dsrNaiList` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- A hostname
- A sequence of 1-10 username elements

The syntax of naiList is as follows:

```
<host>host</host>
<user>user</user>
[
  <user>user</user>
  …
  <user>user</user>
]
```

Parameters:

```
hostA host name.
Values: A string with 1 to 64 characters.

user  A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.
```

Output

The `readSubscriberNaiResponse` message returns the result of the request to read subscriber routing entities.

The response message contains one part `readSubscriberNaiResponse`, which consists of two elements; `result` and `resultSet`. The result element as described in *Response Messages*. Only those subscriber routing entities that are found are returned.

Possible error codes are the same as for the XML request and are listed in *Response*.

The response message also contains a part `resultSet` which contains a result set containing:

- For each NAI found, a list of all the provisioned destination names

The `resultSet` is an XML data structure of type `dsrResultSetDestinationsNai` as defined in the XSD file (see *XML Schema Definition Files (XSD)*). This message part contains the following:

- A sequence of 1-10 NAI elements (based on the order of the requested routing entites)
  - Each NAI element contains:
    - Attributes of the NAI requested
    - A list of provisioned destination name entites

The syntax of `resultSet` is as follows:

```
<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>
]
```

Parameters:

```
hostA host name.
Values: A string with 1 to 64 characters.

user  A user name to be associated with the host to form an NAI.
Values: A string with 1 to 64 characters.

imshssThe name of the IMS HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

ltehssThe name of the LTE HSS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

pcrfThe name of the PCRF for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

ocsThe name of the OCS for which the specified routing entity(s) are to be associated.
Values: A string with 1 to 32 characters.

ofcsThe name of the OFCS for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

aaaThe name of the AAA server for which the specified routing entity(s) are to be
associated.
Values: A string with 1 to 32 characters.

userdef1The name of the first user defined destination for which the specified
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.

userdef2The name of the second user defined destination for which the specified
```

```
routing entity(s) are to
be associated.
Values: A string with 1 to 32 characters.
```

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 *Response Messages*) or in the SOAP Response message (see *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.

The "Type" column indicates if an error is permanent ("P") or temporary ("T"), or indicates success ("S"). A request that results in a permanent error should be discarded and not sent again. A request that results in a temporary can be sent again at a different time, and may be successful.

Error codes that are marked with a "*" are permanent errors that can be fixed by means of configuration, such as allowing read/write access for an IP address, configuring the missing destinations etc

Table 36: SDS Response Message Error Codes

Error Code	Value	Type	Description
SUCCESS	0	S	No Error
NOT_PROCESSED	1	T	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	T	An internal error occurred. Contact Tekelec.
WRITE_UNAVAILABLE	1005	T	Another client already has a transaction open. This will only be returned to clients who do have write access permissions, and also when a bulk import/export is in progress, and the corresponding import/export mode is set to "blocking".
NO_WRITE_PERMISSION	1006	P*	The client making the connection does not have write access permissions.
NO_ACTIVE_TXN	1009	P	No transaction is currently open when a commit or rollback request was sent
ACTIVE_TXN	1010	P	A start transaction request was sent when a transaction was already open on this connection
INVALID_VALUE	1012	P	One of the fields in the request has a invalid value.
NO_UPDATES	1017	S	The write request did not have any successful updates. This is when the data to be written was already the same in the DB, and so the data was not written, or an attempt to delete a non-existant record was made.
DURABILITY_TIMEOUT	1024	S	The update was not made durable in the database within the configured time interval
BAD_IMPORT_CMD	1028	P	The command is not supported by the Import operation.

Error Code	Value	Type	Description
TXN_TOO_BIG	1029	P	Transaction too big (more than the configured maximum number of requests). The maximum number of requests within a transaction is configured using the Maximum Transaction Size system variable, as described in *XML/SOAP Interface System Variables*.
DURABILITY_DEGRADED	1030	T	DB access has been manually disabled.
DB_EXCEPTION	1031	T	An unexpected exception was thrown during the database commit. The entire transaction was rolled back to ensure predictable behavior. Contact Tekelec.
PROV_PROHIBITED	1051	T	DB access has been manually disabled.
INV_REQUEST_NAME	2001	P	The XML request name does not indicate a valid request.
INVALID_XML	2002	P	The request does not contain a valid XML data structure and cannot be parsed.
MISSING_PARAMETER	2003	P	A mandatory parameter is missing.
INVALID_MULT_INST	2004	P	Multiple instances of a parameter than only allows a single instance has been encountered,
UNKNOWN_PARAM_NAME	2005	P	The specified parameter name is unknown for this request.
DEST_NOT_FOUND	2006	P*	The specified destination name could not be found in the database.
IMSI_NOT_FOUND	2007	P	The specified IMSI could not be found in the database.
MSISDN_NOT_FOUND	2008	P	The specified MSISDN could not be found in the database.
NAI_NOT_FOUND	2009	P	The specified NAI (host/user) could not be found in the database.
HOST_NOT_FOUND	2010	P*	The specified host name could not be found in the database.
TXN_TIMED_OUT	2011	P	The Transaction that was in progress has timed out, and automatically rolled back.
TOO_MANY_ADDR	2012	P	Too many IMSI/MSISDN routing entities were specified in the request
NO_DEST_VAL	2013	P	No destination value was supplied in the request, and was expected
NO_ADDR_VAL	2014	P	No IMSI/MSISDN value was supplied in the request, and was expected
TOO_MANY_NAI	2015	P	Too many NAI routing entities were specified in the request

Error Code	Value	Type	Description
NO_NAI_VAL	2016	P	No NAI value was supplied in the request, and was expected
DEST_TYPE_MISMATCH	2017	P	Destination has a different destination type than the desired destination type
INVALID_ARG	2018	P	The argument is not valid
INSTANCE_LIMIT	2019	T	Operation would exceed the maximum number of allowed records in the table
INV_REQ_IN_BLOCK_TX	2020	P	An invalid request has been sent in a block transaction (e.g. startTransaction, commit, or rollback)
INV_REQ_IN_NORMAL_TX	2021	P	An invalid request has been sent in a normal transaction (e.g. a block transaction)

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 37: XML/SOAP Interface System Variables*) can be set via the SDS GUI and can be changed at runtime to effect dynamic server reconfiguration.

Table 37: 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 the SDS.

Appendix C

XML Schema Definition Files (XSD)

Topics:

This section describes the XML requests, responses, and datatypes.

XML Schema Definition Files (XSD)

The definition of the XML requests, responses, and datatypes is documented in the XML Schema Definition (XSD) files called " sdsTypes.xsd " and " dsrTypes.xsd ".

All common SDS related types (prefixed with " sds ") are defined in " sdsTypes.xsd ", and all DSR related types (prefixed with " dsr ") are defined in " dsrTypes.xsd " .

The following table lists the different request names, the XML entity names that need to be created, and the XSD data type of the named entity.

Table 38: XML Schema Requests

Request	XML Request/Response Entity Name	XDS Request/Response Type	Section
Start Database Transaction	startTransaction	sdsStartTransactionRequest	*Request*
	startTransactionResp	sdsStartTransactionResponse	*Response*
Commit Database Transaction	commit	sdsCommitRequest	*Request*
	commitResp	sdsCommitResponse	*Response*
Abort Database Transaction	rollback	sdsRollbackRequest	*Request*
	rollbackResp	sdsRollbackResponse	*Response*
Create/Update IMSI/MSISDN Routing	updateSubscriber	dsrUpdateSubscriberRequest	*Request*
	updateSubscriberResp	dsrUpdateSubscriberResponse	*Response*
Delete IMSI/MSISDN Routing	deleteSubscriber	dsrDeleteSubscriberRequest	*Request*
	deleteSubscriberResp	dsrDeleteSubscriberResponse	*Response*
Get IMSI/MSISDN Routing	readSubscriber	dsrReadSubscriberRequest	*Request*
	readSubscriberResp	dsrReadSubscriberResponse	*Response*
Create/Update NAI Routing	updateSubscriberNai	dsrUpdateSubscriberNaiRequest	*Request*
	updateSubscriberNaiResp	dsrUpdateSubscriberNaiResponse	*Response*
Delete NAI Routing	deleteSubscriberNai	dsrDeleteSubscriberNaiRequest	*Request*
	deleteSubscriberNaiResp	dsrDeleteSubscriberNaiResponse	*Response*
Get NAI Routing	readSubscriberNai	dsrReadSubscriberNaiRequest	*Request*
	readSubscriberNaiResp	dsrReadSubscriberNaiResponse	*Response*
Block Transaction	tx	dsrTransactionRequest	*Request*
	txResp	dsrTransactionResponse	*Response*

SDS

The XSD file " sdsTypes.xsd " is listed in its entirety below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.tekelec.com/sds/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tekelec.com/sds/" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
    <!--Definitions for transaction requests-->
    <xs:complexType name="sdsStartTransactionRequest">
        <xs:annotation>
           <xs:documentation>Definition of a startTransaction request. Element name
 must be
"startTransaction"</xs:documentation>
        </xs:annotation>
        <xs:attribute name="resonly" type="sdsResOnly"/>
        <xs:attribute name="id" type="sdsTransactionId"/>
        <xs:attribute name="timeout" type="sdsTransactionTimeout"/>
    </xs:complexType>
    <xs:complexType name="sdsStartTransactionResponse">
        <xs:annotation>
            <xs:documentation>Definition of a startTransactionResponse response.
Element name must be
"startTransactionResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="startTransaction" type="sdsStartTransactionRequest"
minOccurs="0"/>
            <xs:element name="res" type="sdsResult"/>
        </xs:sequence>
        <xs:attribute name="id" type="sdsTransactionId"/>
    </xs:complexType>
    <xs:complexType name="sdsCommitRequest">
        <xs:annotation>
            <xs:documentation>Definition of a commit request. Element name must be
 "commit"</xs:documentation>
        </xs:annotation>
        <xs:attribute name="resonly" type="sdsResOnly"/>
        <xs:attribute name="id" type="sdsTransactionId"/>
    </xs:complexType>
    <xs:complexType name="sdsCommitResponse">
        <xs:annotation>
           <xs:documentation>Definition of a commitResponse response. Element name
 must be
"commitResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="commit" type="sdsCommitRequest" minOccurs="0"/>
            <xs:element name="res" type="sdsResult"/>
        </xs:sequence>
        <xs:attribute name="id" type="sdsTransactionId"/>
    </xs:complexType>
    <xs:complexType name="sdsRollbackRequest">
        <xs:annotation>
            <xs:documentation>Definition of a rollback request. Element name must
be "rollback"
</xs:documentation>
        </xs:annotation>
        <xs:attribute name="resonly" type="sdsResOnly"/>
        <xs:attribute name="id" type="sdsTransactionId"/>
    </xs:complexType>
    <xs:complexType name="sdsRollbackResponse">
        <xs:annotation>
            <xs:documentation>Definition of a rollbackResponse response. Element
name must be
"rollbackResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="rollback" type="sdsRollbackRequest" minOccurs="0"/>
            <xs:element name="res" type="sdsResult"/>
        </xs:sequence>
        <xs:attribute name="id" type="sdsTransactionId"/>
    </xs:complexType>
    <!--Definition of generic error response-->
    <xs:complexType name="sdsGenericErrorResp">
        <xs:annotation>
            <xs:documentation>Definition of a generic error response. Element name
 must be
"errorResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="res" type="sdsResult"/>
        </xs:sequence>
        <xs:attribute name="id" type="sdsTransactionId"/>
    </xs:complexType>
    <!--Definitions of individual field types/allowed values-->
    <xs:simpleType name="sdsNamespace">
        <xs:annotation>
            <xs:documentation>Definition of namespace attribute</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="sds"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sdsEntity">
        <xs:annotation>
            <xs:documentation>Definition of entity attribute</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="subscriberRouting"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sdsResOnly">
        <xs:annotation>
           <xs:documentation>Definition of result only attribute</xs:documentation>

        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="n"/>
            <xs:enumeration value="y"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sdsTransactionId">
        <xs:annotation>
            <xs:documentation>Definition of transaction id
attribute</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:unsignedInt"/>
    </xs:simpleType>
    <xs:simpleType name="sdsNumberTxReq">
        <xs:annotation>
            <xs:documentation>Definition of number of transaction requests
attribute</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:unsignedInt"/>
    </xs:simpleType>
    <xs:simpleType name="sdsError">
        <xs:annotation>
            <xs:documentation>Definition of error attribute</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:unsignedInt"/>
    </xs:simpleType>
    <xs:simpleType name="sdsAffected">
        <xs:annotation>
          <xs:documentation>Definition of affected rows attribute</xs:documentation>

        </xs:annotation>
        <xs:restriction base="xs:unsignedInt"/>
    </xs:simpleType>
    <xs:simpleType name="sdsErrorDescription">
        <xs:annotation>
            <xs:documentation>Definition of a error description
attribute</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:complexType name="sdsResult">
        <xs:annotation>
            <xs:documentation>Element name must be "res"</xs:documentation>
        </xs:annotation>
        <xs:attribute name="error" type="sdsError" use="required"/>
        <xs:attribute name="affected" type="sdsAffected" use="required"/>
        <xs:attribute name="description" type="sdsErrorDescription"/>
    </xs:complexType>
    <xs:simpleType name="sdsImsi">
        <xs:annotation>
            <xs:documentation>Definition of IMSI parameter</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:minLength value="10"/>
            <xs:maxLength value="15"/>
            <xs:pattern value="([0..9]){10,15}"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sdsMsisdn">
        <xs:annotation>
            <xs:documentation>Definition of MSISDN parameter</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:minLength value="8"/>
            <xs:maxLength value="15"/>
            <xs:pattern value="([0..9]){8,15}"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sdsNaiHost">
        <xs:annotation>
            <xs:documentation>Definition of Nai host parameter</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="64"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sdsNaiUser">
        <xs:annotation>
            <xs:documentation>Definition of Nai user parameter</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="64"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sdsDestinationName">
        <xs:annotation>
            <xs:documentation>Definition of destination name</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="32"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sdsTransactionTimeout">
        <xs:restriction base="xs:unsignedInt">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="3600"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

DSR

The XSD file " dsrTypes.xsd " is listed in its entirety below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.tekelec.com/sds/dsr/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sds="http://www.tekelec.com/sds/"
targetNamespace="http://www.tekelec.com/sds/dsr/"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
    <!--Include common SDS types-->
   <xs:import namespace="http://www.tekelec.com/sds/" schemaLocation="sdsTypes.xsd"/>

    <!--Definitions for IMSI/MSISDN related commands-->
    <xs:complexType name="dsrUpdateSubscriberRequest">
        <xs:annotation>
            <xs:documentation>Definition of an updateSubscriber request. Element
name must be
"updateSubscriber"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:element name="imsi" type="sds:sdsImsi" minOccurs="0" maxOccurs="10"/>

            <xs:element name="msisdn" type="sds:sdsMsisdn" minOccurs="0"
maxOccurs="10"/>
           <xs:element name="imshss" type="sds:sdsDestinationName" minOccurs="0"/>

           <xs:element name="ltehss" type="sds:sdsDestinationName" minOccurs="0"/>

            <xs:element name="pcrf" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ocs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ofcs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="aaa" type="sds:sdsDestinationName" minOccurs="0"/>
          <xs:element name="userdef1" type="sds:sdsDestinationName" minOccurs="0"/>

          <xs:element name="userdef2" type="sds:sdsDestinationName" minOccurs="0"/>

        </xs:sequence>
        <xs:attribute name="ent" type="sds:sdsEntity" use="required"/>
        <xs:attribute name="ns" type="sds:sdsNamespace" use="required"/>
        <xs:attribute name="resonly" type="sds:sdsResOnly"/>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
        <xs:attribute name="timeout">
            <xs:simpleType>
                <xs:restriction base="xs:unsignedInt">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="3600"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="dsrUpdateSubscriberResponse">
        <xs:annotation>
            <xs:documentation>Definition of an updateSubscriber response. Element
name must be
"updateSubscriberResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="updateSubscriber" type="dsrUpdateSubscriberRequest"
minOccurs="0"/>
            <xs:element name="res" type="sds:sdsResult"/>
        </xs:sequence>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
    </xs:complexType>
    <xs:complexType name="dsrDeleteSubscriberRequest">
        <xs:annotation>
           <xs:documentation>Definition of a deleteSubscriber request. Element name
 must be
"deleteSubscriber"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
           <xs:element name="imsi" type="sds:sdsImsi" minOccurs="0" maxOccurs="10"/>

            <xs:element name="msisdn" type="sds:sdsMsisdn" minOccurs="0"
maxOccurs="10"/>
        </xs:sequence>
        <xs:attribute name="ent" type="sds:sdsEntity" use="required"/>
        <xs:attribute name="ns" type="sds:sdsNamespace" use="required"/>
        <xs:attribute name="resonly" type="sds:sdsResOnly"/>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
        <xs:attribute name="timeout">
            <xs:simpleType>
                <xs:restriction base="xs:unsignedInt">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="3600"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="dsrDeleteSubscriberResponse">
        <xs:annotation>
            <xs:documentation>Definition of a deleteSubscriber response. Element
name must be
"deleteSubscriberResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="deleteSubscriber" type="dsrDeleteSubscriberRequest"
minOccurs="0"/>
            <xs:element name="res" type="sds:sdsResult"/>
        </xs:sequence>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
    </xs:complexType>
    <xs:complexType name="dsrReadSubscriberRequest">
        <xs:annotation>
            <xs:documentation>Definition of a readSubscriber request. Element name
 must be
"readSubscriber"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:element name="imsi" type="sds:sdsImsi" minOccurs="0" maxOccurs="10"/>

            <xs:element name="msisdn" type="sds:sdsMsisdn" minOccurs="0"
maxOccurs="10"/>
        </xs:sequence>
        <xs:attribute name="ent" type="sds:sdsEntity" use="required"/>
        <xs:attribute name="ns" type="sds:sdsNamespace" use="required"/>
        <xs:attribute name="resonly" type="sds:sdsResOnly"/>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
        <xs:attribute name="timeout">
            <xs:simpleType>
                <xs:restriction base="xs:unsignedInt">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="3600"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="dsrReadSubscriberResponse">
        <xs:annotation>
            <xs:documentation>Definition of a readSubscriber response. Element name
 must be
"readSubscriberResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="readSubscriber" type="dsrReadSubscriberRequest"
minOccurs="0"/>
            <xs:element name="res" type="sds:sdsResult"/>
            <xs:element name="rset" type="dsrResultSetDestinationsAddress"
minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
    </xs:complexType>
    <!--Definitions for Nai related commands-->
    <xs:complexType name="dsrUpdateSubscriberNaiRequest">
        <xs:annotation>
            <xs:documentation>Definition of an updateSubscriberNai request. Element
 name must be
"updateSubscriberNai"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="host" type="sds:sdsNaiHost"/>
            <xs:element name="user" type="sds:sdsNaiUser" maxOccurs="10"/>
           <xs:element name="imshss" type="sds:sdsDestinationName" minOccurs="0"/>

           <xs:element name="ltehss" type="sds:sdsDestinationName" minOccurs="0"/>

            <xs:element name="pcrf" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ocs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ofcs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="aaa" type="sds:sdsDestinationName" minOccurs="0"/>
          <xs:element name="userdef1" type="sds:sdsDestinationName" minOccurs="0"/>

          <xs:element name="userdef2" type="sds:sdsDestinationName" minOccurs="0"/>

        </xs:sequence>
        <xs:attribute name="ent" type="sds:sdsEntity" use="required"/>
        <xs:attribute name="ns" type="sds:sdsNamespace" use="required"/>
        <xs:attribute name="resonly" type="sds:sdsResOnly"/>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
        <xs:attribute name="timeout">
```

```
            <xs:simpleType>
                <xs:restriction base="xs:unsignedInt">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="3600"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="dsrUpdateSubscriberNaiResponse">
        <xs:annotation>
           <xs:documentation>Definition of an updateSubscriberNai response. Element
 name must be
"updateSubscriberNaiResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="updateSubscriberNai"
type="dsrUpdateSubscriberNaiRequest" minOccurs="0"/>
            <xs:element name="res" type="sds:sdsResult"/>
        </xs:sequence>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
    </xs:complexType>
    <xs:complexType name="dsrDeleteSubscriberNaiRequest">
        <xs:annotation>
            <xs:documentation>Definition of a deleteSubscriberNai request. Element
 name must be
"deleteSubscriberNai"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="host" type="sds:sdsNaiHost"/>
            <xs:element name="user" type="sds:sdsNaiUser" maxOccurs="10"/>
        </xs:sequence>
        <xs:attribute name="ent" type="sds:sdsEntity" use="required"/>
        <xs:attribute name="ns" type="sds:sdsNamespace" use="required"/>
        <xs:attribute name="resonly" type="sds:sdsResOnly"/>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
        <xs:attribute name="timeout">
            <xs:simpleType>
                <xs:restriction base="xs:unsignedInt">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="3600"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="dsrDeleteSubscriberNaiResponse">
        <xs:annotation>
           <xs:documentation>Definition of a deleteSubscriberNai response. Element
 name must be
"deleteSubscriberNaiResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="deleteSubscriberNai"
type="dsrDeleteSubscriberNaiRequest" minOccurs="0"/>
            <xs:element name="res" type="sds:sdsResult"/>
        </xs:sequence>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
    </xs:complexType>
    <xs:complexType name="dsrReadSubscriberNaiRequest">
        <xs:annotation>
            <xs:documentation>Definition of a readSubscriberNai request. Element
name must be
"readSubscriberNai"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="host" type="sds:sdsNaiHost"/>
            <xs:element name="user" type="sds:sdsNaiUser" maxOccurs="10"/>
        </xs:sequence>
        <xs:attribute name="ent" type="sds:sdsEntity" use="required"/>
        <xs:attribute name="ns" type="sds:sdsNamespace" use="required"/>
        <xs:attribute name="resonly" type="sds:sdsResOnly"/>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
        <xs:attribute name="timeout">
            <xs:simpleType>
                <xs:restriction base="xs:unsignedInt">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="3600"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="dsrReadSubscriberNaiResponse">
        <xs:annotation>
            <xs:documentation>Definition of a readSubscriberNai response. Element
name must be
"readSubscriberNaiResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
           <xs:element name="readSubscriberNai" type="dsrReadSubscriberNaiRequest"
 minOccurs="0"/>
            <xs:element name="res" type="sds:sdsResult"/>
          <xs:element name="rset" type="dsrResultSetDestinationsNai" minOccurs="0"/>

        </xs:sequence>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
    </xs:complexType>
    <!--Definitions for block transactions-->
    <xs:complexType name="dsrTransactionRequest">
        <xs:annotation>
            <xs:documentation>Definition of a block transaction request. Element
name much be
"tx"</xs:documentation>
        </xs:annotation>
        <xs:sequence maxOccurs="50">
            <xs:choice>
              <xs:element name="updateSubscriber" type="dsrUpdateSubscriberRequest"
 minOccurs="0"/>
              <xs:element name="deleteSubscriber" type="dsrDeleteSubscriberRequest"
 minOccurs="0"/>
                <xs:element name="readSubscriber" type="dsrReadSubscriberRequest"
minOccurs="0"/>
                <xs:element name="updateSubscriberNai"
type="dsrUpdateSubscriberNaiRequest" minOccurs="0"/>
                <xs:element name="deleteSubscriberNai"
type="dsrDeleteSubscriberNaiRequest" minOccurs="0"/>
                <xs:element name="readSubscriberNai"
type="dsrReadSubscriberNaiRequest" minOccurs="0"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="resonly" type="sds:sdsResOnly"/>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
        <xs:attribute name="timeout">
            <xs:simpleType>
                <xs:restriction base="xs:unsignedInt">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="3600"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="dsrTransactionResponse">
```

```
        <xs:annotation>
            <xs:documentation>Definition of a block transaction response. Element
name much be
"txResp"</xs:documentation>
        </xs:annotation>
        <xs:sequence maxOccurs="50">
          <xs:element name="updateSubscriberResp" type="dsrUpdateSubscriberResponse"
 minOccurs="0"/>
          <xs:element name="deleteSubscriberResp" type="dsrDeleteSubscriberResponse"
 minOccurs="0"/>
            <xs:element name="readSubscriberResp" type="dsrReadSubscriberResponse"
 minOccurs="0"/>
            <xs:element name="updateSubscriberNaiResp"
type="dsrUpdateSubscriberNaiResponse" minOccurs="0"/>
            <xs:element name="deleteSubscriberNaiResp"
type="dsrDeleteSubscriberNaiResponse" minOccurs="0"/>
            <xs:element name="readSubscriberNaiResp"
type="dsrReadSubscriberNaiResponse" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="nbreq" type="sds:sdsNumberTxReq" use="required"/>
        <xs:attribute name="id" type="sds:sdsTransactionId"/>
    </xs:complexType>
    <!--Definitions for lists-->
    <xs:complexType name="dsrAddressList">
        <xs:annotation>
            <xs:documentation>Definition of a list of IMSI's and
MSISDN's</xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:element name="imsi" type="sds:sdsImsi" minOccurs="0" maxOccurs="10"/>

            <xs:element name="msisdn" type="sds:sdsMsisdn" minOccurs="0"
maxOccurs="10"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="dsrNaiList">
        <xs:annotation>
            <xs:documentation>Definition of a list of NAI's</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="host" type="sds:sdsNaiHost"/>
            <xs:sequence maxOccurs="10">
                <xs:element name="user" type="sds:sdsNaiUser"/>
            </xs:sequence>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="dsrDestinationList">
        <xs:annotation>
          <xs:documentation>Definition of a list of destinations</xs:documentation>

        </xs:annotation>
        <xs:sequence>
           <xs:element name="imshss" type="sds:sdsDestinationName" minOccurs="0"/>

           <xs:element name="ltehss" type="sds:sdsDestinationName" minOccurs="0"/>

            <xs:element name="pcrf" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ocs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ofcs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="aaa" type="sds:sdsDestinationName" minOccurs="0"/>
          <xs:element name="userdef1" type="sds:sdsDestinationName" minOccurs="0"/>

          <xs:element name="userdef2" type="sds:sdsDestinationName" minOccurs="0"/>

        </xs:sequence>
    </xs:complexType>
    <xs:annotation>
        <xs:documentation>Definitions for result sets</xs:documentation>
    </xs:annotation>
    <xs:complexType name="dsrRowImsi">
        <xs:annotation>
            <xs:documentation>Definition of an output row for an IMSI
destination</xs:documentation>
        </xs:annotation>
        <xs:sequence>
           <xs:element name="imshss" type="sds:sdsDestinationName" minOccurs="0"/>

           <xs:element name="ltehss" type="sds:sdsDestinationName" minOccurs="0"/>

            <xs:element name="pcrf" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ocs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ofcs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="aaa" type="sds:sdsDestinationName" minOccurs="0"/>
          <xs:element name="userdef1" type="sds:sdsDestinationName" minOccurs="0"/>

          <xs:element name="userdef2" type="sds:sdsDestinationName" minOccurs="0"/>

        </xs:sequence>
        <xs:attribute name="imsi" type="sds:sdsImsi"/>
    </xs:complexType>
    <xs:complexType name="dsrRowMsisdn">
        <xs:annotation>
            <xs:documentation>Definition of an output row for an MSISDN
destination</xs:documentation>
        </xs:annotation>
        <xs:sequence>
           <xs:element name="imshss" type="sds:sdsDestinationName" minOccurs="0"/>

           <xs:element name="ltehss" type="sds:sdsDestinationName" minOccurs="0"/>

            <xs:element name="pcrf" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ocs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ofcs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="aaa" type="sds:sdsDestinationName" minOccurs="0"/>
          <xs:element name="userdef1" type="sds:sdsDestinationName" minOccurs="0"/>

          <xs:element name="userdef2" type="sds:sdsDestinationName" minOccurs="0"/>

        </xs:sequence>
        <xs:attribute name="msisdn" type="sds:sdsMsisdn"/>
    </xs:complexType>
    <xs:complexType name="dsrRowNai">
        <xs:annotation>
            <xs:documentation>Definition of an output row for an Nai
destination</xs:documentation>
        </xs:annotation>
        <xs:sequence>
           <xs:element name="imshss" type="sds:sdsDestinationName" minOccurs="0"/>

           <xs:element name="ltehss" type="sds:sdsDestinationName" minOccurs="0"/>

            <xs:element name="pcrf" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ocs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="ofcs" type="sds:sdsDestinationName" minOccurs="0"/>
            <xs:element name="aaa" type="sds:sdsDestinationName" minOccurs="0"/>
          <xs:element name="userdef1" type="sds:sdsDestinationName" minOccurs="0"/>

          <xs:element name="userdef2" type="sds:sdsDestinationName" minOccurs="0"/>

        </xs:sequence>
        <xs:attribute name="host" type="sds:sdsNaiHost"/>
        <xs:attribute name="user" type="sds:sdsNaiUser"/>
    </xs:complexType>
    <xs:complexType name="dsrResultSetDestinationsAddress">
        <xs:annotation>
            <xs:documentation>Definition of an entire result set (i.e. many rows)
for IMSI/MSISDN. Element
name must be "rset"</xs:documentation>
        </xs:annotation>
        <xs:sequence maxOccurs="10">
            <xs:choice>
                <xs:element name="imsi" type="dsrRowImsi"/>
                <xs:element name="msisdn" type="dsrRowMsisdn"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="dsrResultSetDestinationsNai">
        <xs:annotation>
            <xs:documentation>Definition of an entire result set (i.e. many rows)
for Nai's. Element name
must be "rset"</xs:documentation>
        </xs:annotation>
        <xs:sequence maxOccurs="10">
            <xs:element name="nai" type="dsrRowNai"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Appendix D

SOAP WSDL Files

Topics:

This section describes the SOAP requests and responses that are documented in the WSDL files.

SOAP WSDL Files

The definition of the SOAP requests and responses are documented in the WSDL files called “sdsProvisioning.wsdl” and “dsrProvisioning.wsdl”. These are WSDL v1.1.

All SDS related SOAP operations and messages (prefixed with “sds”) are defined in “sdsProvisioning.wsdl” , and all DSR related operations and messages (prefixed with “dsr”) are defined in “dsrProvisioning.wsdl” .

The following table lists the different SOAP operation names, and the SOAP message names.

Table 39: SOAP WSDL Requests

Request	XML Request/Response Entity Name	XDS Request/Response Type	Section
Start Database Transaction	startTransaction	sdsStartTransactionRequest	*Request*
	startTransactionResp	sdsStartTransactionResponse	*Response*
Commit Database Transaction	commit	sdsCommitRequest	*Request*
	commitResp	sdsCommitResponse	*Response*
Abort Database Transaction	rollback	sdsRollbackRequest	*Request*
	rollbackResp	sdsRollbackResponse	*Response*
Create/Update IMSI/MSISDN Routing	updateSubscriber	dsrUpdateSubscriberRequest	*Request*
	updateSubscriberResp	dsrUpdateSubscriberResponse	*Response*
Delete IMSI/MSISDN Routing	deleteSubscriber	dsrDeleteSubscriberRequest	*Request*
	deleteSubscriberResp	dsrDeleteSubscriberResponse	*Response*
Get IMSI/MSISDN Routing	readSubscriber	dsrReadSubscriberRequest	*Request*
	readSubscriberResp	dsrReadSubscriberResponse	*Response*
Create/Update NAI Routing	updateSubscriberNai	dsrUpdateSubscriberNaiRequest	*Request*
	updateSubscriberNaiResp	dsrUpdateSubscriberNaiResponse	*Response*
Delete NAI Routing	deleteSubscriberNai	dsrDeleteSubscriberNaiRequest	*Request*
	deleteSubscriberNaiResp	dsrDeleteSubscriberNaiResponse	*Response*
Get NAI Routing	readSubscriberNai	dsrReadSubscriberNaiRequest	*Request*
	readSubscriberNaiResp	dsrReadSubscriberNaiResponse	*Response*

SDS

The WSDL file " sdsProvisioning.wsdl " is listed in its entirety below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap=
"http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs=
"http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime=
"http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns=
"http://www.tekelec.com/sds/soap" xmlns:ns="http://www.tekelec.com/sds/" xmlns:soap12=
"http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://www.tekelec.com/sds/soap">
    <wsdl:import namespace="http://www.tekelec.com/sds/" location="sdsTypes.xsd"/>
    <wsdl:types>
        <xs:schema targetNamespace="http://new.webservice.namespace"
elementFormDefault="qualified"/>
    </wsdl:types>
    <wsdl:message name="startTransactionResponse">
        <wsdl:part name="startTransactionResponse" type="ns:sdsResult"/>
    </wsdl:message>
    <wsdl:message name="commitResponse">
        <wsdl:part name="commitResponse" type="ns:sdsResult"/>
    </wsdl:message>
    <wsdl:message name="rollbackResponse">
        <wsdl:part name="rollbackResponse" type="ns:sdsResult"/>
    </wsdl:message>
    <wsdl:message name="startTransactionRequest">
       <wsdl:part name="startTransactionRequest" type="ns:sdsTransactionTimeout"/>

    </wsdl:message>
    <wsdl:message name="commitRequest">
        <wsdl:part name="commitRequest" type="xs:string"/>
    </wsdl:message>
    <wsdl:message name="rollbackRequest">
        <wsdl:part name="rollbackRequest" type="xs:string"/>
    </wsdl:message>
    <wsdl:portType name="sdsProvisioning">
        <wsdl:operation name="startTransaction">
            <wsdl:input message="tns:startTransactionRequest"/>
            <wsdl:output message="tns:startTransactionResponse"/>
        </wsdl:operation>
        <wsdl:operation name="commit">
            <wsdl:input message="tns:commitRequest"/>
            <wsdl:output message="tns:commitResponse"/>
        </wsdl:operation>
        <wsdl:operation name="rollback">
            <wsdl:input message="tns:rollbackRequest"/>
            <wsdl:output message="tns:rollbackResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="sdsProvisioningSoap" type="tns:sdsProvisioning">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="startTransaction">
          <soap:operation soapAction="http://www.tekelec.com/sds/startTransaction"/>

            <wsdl:input>
                <soap:body use="literal" namespace="http://www.tekelec.com/sds/"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" namespace="http://www.tekelec.com/sds/"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="commit">
            <soap:operation soapAction="http://www.tekelec.com/sds/commit"/>
            <wsdl:input>
                <soap:body use="literal" namespace="http://www.tekelec.com/sds/"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" namespace="http://www.tekelec.com/sds/"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="rollback">
            <soap:operation soapAction="http://www.tekelec.com/sds/rollback"/>
            <wsdl:input>
                <soap:body use="literal" namespace="http://www.tekelec.com/sds/"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" namespace="http://www.tekelec.com/sds/"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="sdsProvisioning">
        <wsdl:port name="sdsProvisioningSoap" binding="tns:sdsProvisioningSoap">
            <soap:address location="http://www.tekelec.com/sds/"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

DSR

The WSDL file " dsrProvisioning.wsdl " is listed in its entirety below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap=
"http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs=
"http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime=
"http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns=
"http://www.tekelec.com/sds/dsr/soap/" xmlns:ns="http://www.tekelec.com/sds/dsr/"
xmlns:soap12=
"http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:dsr="http://www.tekelec.com/sds/dsr/"
 xmlns:sds=
"http://www.tekelec.com/sds/" xmlns:ws="http://www.tekelec.com/sds/dsr/soap/"
targetNamespace=
"http://www.tekelec.com/sds/dsr/soap/">
    <wsdl:import namespace="http://www.tekelec.com/sds/" location="sdsTypes.xsd"/>
   <wsdl:import namespace="http://www.tekelec.com/sds/dsr/" location="dsrTypes.xsd"/>

    <wsdl:types>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-ENC=
"http://schemas.xmlsoap.org/soap/encoding/" xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi=
"http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tekelec.com/sds/dsr/soap/">
```

```
            <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"
schemaLocation=
"soap-encoding.xsd"/>
            <xsd:import namespace="http://www.tekelec.com/sds/"
schemaLocation="sdsTypes.xsd"/>
            <xsd:import namespace="http://www.tekelec.com/sds/dsr/"
schemaLocation="dsrTypes.xsd"/>
            <!--Request Types-->
            <xsd:complexType name="updateSubscriberRequest">
                <xsd:sequence>
                    <xsd:element name="addressList" type="dsr:dsrAddressList"/>
                    <xsd:element name="destinationList"
type="dsr:dsrDestinationList"/>
                </xsd:sequence>
                <xsd:attribute name="timeout" type="sds:sdsTransactionTimeout"/>
            </xsd:complexType>
            <xsd:complexType name="deleteSubscriberRequest">
                <xsd:sequence>
                    <xsd:element name="addressList" type="dsr:dsrAddressList"/>
                </xsd:sequence>
                <xsd:attribute name="timeout" type="sds:sdsTransactionTimeout"/>
            </xsd:complexType>
            <xsd:complexType name="readSubscriberRequest">
                <xsd:sequence>
                    <xsd:element name="addressList" type="dsr:dsrAddressList"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="updateSubscriberNaiRequest">
                <xsd:sequence>
                    <xsd:element name="naiList" type="dsr:dsrNaiList"/>
                    <xsd:element name="destinationList"
type="dsr:dsrDestinationList"/>
                </xsd:sequence>
                <xsd:attribute name="timeout" type="sds:sdsTransactionTimeout"/>
            </xsd:complexType>
            <xsd:complexType name="deleteSubscriberNaiRequest">
                <xsd:sequence>
                    <xsd:element name="naiList" type="dsr:dsrNaiList"/>
                </xsd:sequence>
                <xsd:attribute name="timeout" type="sds:sdsTransactionTimeout"/>
            </xsd:complexType>
            <xsd:complexType name="readSubscriberNaiRequest">
                <xsd:sequence>
                    <xsd:element name="naiList" type="dsr:dsrNaiList"/>
                </xsd:sequence>
            </xsd:complexType>
            <!--Response Types-->
            <xsd:complexType name="readSubscriberResponse">
                <xsd:sequence>
                    <xsd:element name="result" type="sds:sdsResult"/>
                    <xsd:element name="resultSet"
type="dsr:dsrResultSetDestinationsAddress" minOccurs="0"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="readSubscriberNaiResponse">
                <xsd:sequence>
                    <xsd:element name="result" type="sds:sdsResult"/>
                    <xsd:element name="resultSet"
type="dsr:dsrResultSetDestinationsNai" minOccurs="0"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="updateSubscriberRequest">
       <wsdl:part name="updateSubscriberRequest" type="tns:updateSubscriberRequest"/>
    </wsdl:message>
    <wsdl:message name="updateSubscriberResponse">
        <wsdl:part name="updateSubscriberResponse" type="sds:sdsResult"/>
    </wsdl:message>
    <wsdl:message name="deleteSubscriberRequest">
       <wsdl:part name="deleteSubscriberRequest" type="tns:deleteSubscriberRequest"/>

    </wsdl:message>
    <wsdl:message name="deleteSubscriberResponse">
        <wsdl:part name="deleteSubscriberResponse" type="sds:sdsResult"/>
    </wsdl:message>
    <wsdl:message name="readSubscriberRequest">
        <wsdl:part name="readSubscriberRequest" type="tns:readSubscriberRequest"/>
    </wsdl:message>
    <wsdl:message name="readSubscriberResponse">
       <wsdl:part name="readSubscriberResponse" type="tns:readSubscriberResponse"/>

    </wsdl:message>
    <wsdl:message name="updateSubscriberNaiRequest">
        <wsdl:part name="updateSubscriberNaiRequest"
type="tns:updateSubscriberNaiRequest"/>
    </wsdl:message>
    <wsdl:message name="updateSubscriberNaiResponse">
        <wsdl:part name="updateSubscriberNaiResponse" type="sds:sdsResult"/>
    </wsdl:message>
    <wsdl:message name="deleteSubscriberNaiRequest">
        <wsdl:part name="deleteSubscriberNaiRequest"
type="tns:deleteSubscriberNaiRequest"/>
    </wsdl:message>
    <wsdl:message name="deleteSubscriberNaiResponse">
        <wsdl:part name="deleteSubscriberNaiResponse" type="sds:sdsResult"/>
    </wsdl:message>
    <wsdl:message name="readSubscriberNaiRequest">
        <wsdl:part name="readSubscriberNaiRequest"
type="tns:readSubscriberNaiRequest"/>
    </wsdl:message>
    <wsdl:message name="readSubscriberNaiResponse">
        <wsdl:part name="readSubscriberNaiResponse"
type="tns:readSubscriberNaiResponse"/>
    </wsdl:message>
    <wsdl:portType name="dsrProvisioning">
        <wsdl:operation name="updateSubscriber">
            <wsdl:input message="tns:updateSubscriberRequest"/>
            <wsdl:output message="tns:updateSubscriberResponse"/>
        </wsdl:operation>
        <wsdl:operation name="deleteSubscriber">
            <wsdl:input message="tns:deleteSubscriberRequest"/>
            <wsdl:output message="tns:deleteSubscriberResponse"/>
        </wsdl:operation>
        <wsdl:operation name="readSubscriber">
            <wsdl:input message="tns:readSubscriberRequest"/>
            <wsdl:output message="tns:readSubscriberResponse"/>
        </wsdl:operation>
        <wsdl:operation name="updateSubscriberNai">
            <wsdl:input message="tns:updateSubscriberNaiRequest"/>
            <wsdl:output message="tns:updateSubscriberNaiResponse"/>
        </wsdl:operation>
        <wsdl:operation name="deleteSubscriberNai">
            <wsdl:input message="tns:deleteSubscriberNaiRequest"/>
            <wsdl:output message="tns:deleteSubscriberNaiResponse"/>
        </wsdl:operation>
        <wsdl:operation name="readSubscriberNai">
            <wsdl:input message="tns:readSubscriberNaiRequest"/>
            <wsdl:output message="tns:readSubscriberNaiResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="dsrProvisioningSoap" type="tns:dsrProvisioning">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="updateSubscriber">
            <soap:operation
soapAction="http://www.tekelec.com/sds/dsr/updateSubscriber"/>
            <wsdl:input>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:input>
            <wsdl:output>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="deleteSubscriber">
            <soap:operation
soapAction="http://www.tekelec.com/sds/dsr/deleteSubscriber"/>
            <wsdl:input>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:input>
            <wsdl:output>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="readSubscriber">
            <soap:operation
soapAction="http://www.tekelec.com/sds/dsr/readSubscriber"/>
            <wsdl:input>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:input>
            <wsdl:output>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="updateSubscriberNai">
            <soap:operation
soapAction="http://www.tekelec.com/sds/dsr/updateSubscriberNai"/>
            <wsdl:input>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:input>
            <wsdl:output>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="deleteSubscriberNai">
            <soap:operation
soapAction="http://www.tekelec.com/sds/dsr/deleteSubscriberNai"/>
            <wsdl:input>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:input>
            <wsdl:output>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="readSubscriberNai">
            <soap:operation
soapAction="http://www.tekelec.com/sds/dsr/readSubscriberNai"/>
            <wsdl:input>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:input>
            <wsdl:output>
              <soap:body use="literal" namespace="http://www.tekelec.com/sds/dsr/"/>

            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="dsrProvisioning">
        <wsdl:port name="dsrProvisioningSoap" binding="tns:dsrProvisioningSoap">
            <soap:address location="http://www.tekelec.com/sds/dsr/"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Appendix E

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	Conditioning Action NPP CAs indicate what digit conditioning actions to execute when processing a digit string.
CSV	Comma-separated values The comma-separated value file format is a delimited data format that has fields separated by the comma character and records separated by newlines (a newline is a special character or sequence of characters signifying the end of a line of text).
D	
DSR	Diameter Signaling Router A set of co-located Message Processors which share common Diameter routing tables and are supported by a pair of OAM servers. A DSR Network Element may consist of one or more Diameter nodes.
F	
FQDN	Fully qualified domain name The complete domain name for a specific computer on the Internet (for example, www.tekelec.com).
I	

I	
IMSI	International Mobile Subscriber Identity
K	
KPI	Key Performance Indicators
M	
MSISDN	Mobile Station International Subscriber Directory Number The MSISDN is the network specific subscriber number of a mobile communications subscriber. This is normally the phone number that is used to reach the subscriber.
N	
NAI	Network Access Identifier The user identity submitted by the client during network authentication.
NOC	Network Operations Center
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 Site Operation 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.

S

SNMP	Simple Network Management Protocol. 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.
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer

U

UTC	Coordinated Universal Time

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.

W

WSDL	Web Service Definition Language

X

XML	eXtensible Markup Language A version of the Standard Generalized Markup Language (SGML) that allows Web developers to create customized tags for additional functionality.

X

XSD	XML Schema Definition