Oracle® Communications User Data Repository

SOAP Provisioning Interface Reference

Release 10.0

E53212 Revision 01

November 2014

ORACLE®

Oracle® Communications SOAP Provisioning Interface Reference, Release 10.0

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Topics:

The Introduction chapter explains the purpose and organization of the documentation, defines the document's audience and admonishments, lists related publications and how to location them, and provides information about technical support and training.

Overview

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

Scope and Audience

This documentation is intended for trained and qualified system operators and administrators who are responsible for managing the Enhanced Subscriber Profile Repository (ESPR) application on the User Data Repository (UDR) platform.

Manual Organization

This document is organized into the following chapters:

1. *Introduction* explains the purpose and organization of this documentation, defines its audience and admonishments, lists related publications and where to find them, and provides information about technical support and training.
2. *System Architecture* describes the UDR system architecture.
3. *SOAP Interface Description* describes the SOAP-based provisioning interface.
4. *SOAP Interface Message Definitions* describes the SOAP interface message syntax and parameters.
5. *UDR Data Model* describes the subscriber data and their defined entities and attributes.
6. *Subscriber Provisioning* describes the subscriber provisioning commands and their parameters.
7. *Pool Provisioning* describes the pool provisioning commands and their parameters.
8. *Error Codes* describes the SOAP interface error codes that are returned by the XDS/SOAP Server.
9. *SOAP Interface System Variables* describes the set of system variables (and their parameters) that affect the system's operation as it runs.
10. *Legacy SPR Compatibility Mode* describes the different request/responses to which enabling Legacy SPR Compatibility Mode applies.

Documentation Admonishments

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

Table 1: Admonishments

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

Related Publications

For information about additional publications that are related to this document, refer to the *Related Publications Reference* document, which is published as a separate document on the Oracle Technology Network (OTN) site. See *Locate Product Documentation on the Oracle Technology Network Site* for more information.

Locate Product Documentation on the Oracle Technology Network Site

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

1. Log into the Oracle Technology Network site at *http://docs.oracle.com*.
2. Under **Applications**, click the link for **Communications**.
   The **Oracle Communications Documentation** window opens with Tekelec shown near the top.
3. Click **Oracle Communications Documentation for Tekelec Products**.
4. Navigate to your Product and then the Release Number, and click the **View** link (the **Download** link will retrieve the entire documentation set).
5. To download a file to your location, right-click the PDF link and select **Save Target As**.

Customer Training

Oracle University offers training for service providers and enterprises. Visit our web site to view, and register for, Oracle Communications training:

http://education.oracle.com/communication

To obtain contact phone numbers for countries or regions, visit the Oracle University Education web site:

www.oracle.com/education/contacts

My Oracle Support (MOS)

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

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

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

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

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

Emergency Response

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

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

- A total system failure that results in loss of all transaction processing capability
- Significant reduction in system capacity or traffic handling capability

- Loss of the system's ability to perform automatic system reconfiguration
- Inability to restart a processor or the system
- Corruption of system databases that requires service affecting corrective actions
- Loss of access for maintenance or recovery operations
- Loss of the system ability to provide any required critical or major trouble notification

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

Chapter 2

System Architecture

Topics:

This chapter provides an overview of SOAP system architecture.

System Architecture Overview

Oracle Communications User Data Repository (UDR) performs the function of a subscriber profile repository (SPR), which is a database system that acts as a single logical repository that stores subscriber data. The subscriber data that traditionally has been stored into the HSS /HLR/AuC, Application Servers, etc., is now stored in UDR as specified in 3GPP UDC information model. UDR facilitates the share and the provisioning of user related data throughout services of 3GPP system. Several Applications Front Ends, such as: one or more PCRF/HSS/HLR/AuCFEs can be served by an UDR.

The data stored in UDR can be permanent and temporary data. Permanent data is subscription data and relates to the required information the system needs to know to perform the service. User identities (e.g. MSISDN, IMSI, NAI and AccountId), service data (e.g. service profile) and authentication data are examples of the subscription data. This kind of user data has a lifetime as long as the user is permitted to use the service and may be modified by administration means. Temporary subscriber data is dynamic data which may be changed as a result of normal operation of the system or traffic conditions (e.g. transparent data stored by Application Servers for service execution, user status, usage, etc.).

UDR is a database system providing the storage and management of subscriber policy control data for PCRF nodes with future upgradability to support additional types of nodes. Subscriber/Pool data is created/retrieved/modified or deleted through the provisioning or by the Sh interface peers (PCRF). The following subscriber/pool data is stored in UDR:

- Subscriber
  - Profile
  - Quota
  - State
  - Dynamic Quota
- Pool
  - Pool Profile
  - Pool Quota
  - Pool State
  - Pool Dynamic Quota

Figure 1: UDR High Level Architecture illustrates a high level UDR Architecture.

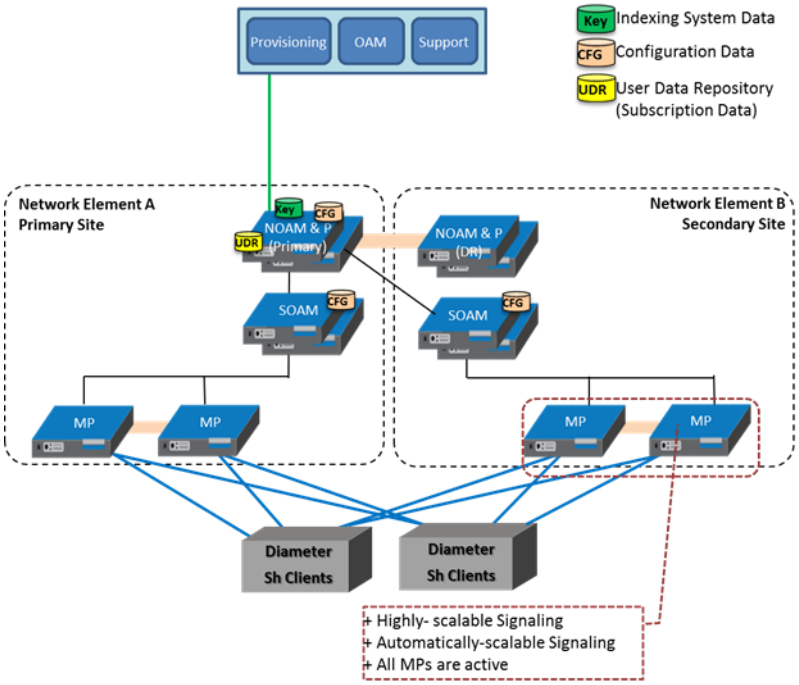

Figure 1: UDR High Level Architecture

As shown in the figure, UDR consists of several functional blocks. The Message Processors (MP) provide support for a variety of protocols that entail the front end signaling to peer network nodes. The back end User Data Repository (UDR) database will reside on the N-OAMP servers. The initial release will focus on the development of the Sh messaging interface for use with UDR.

As the product evolves forward, the subscriber profiles in UDR can be expanded to support data associated with additional applications. Along with that, the MPs can be expanded to support additional Diameter interfaces associated with these applications. The IPFE can be integrated with the product to facilitate signaling distribution across multiple MP nodes.

The Network level OAMP server (NOAM&P) shown in the architecture provides the provisioning, configuration and maintenance functions for all the network elements under it.

System level OAM server (SOAM) is a required functional block for each network element which gets data replicated from NOAM&P and in turn replicates the data to the message processors.

MP functions as the client-side of the network application, provide the network connectivity and hosts network stack such as Diameter, SOAP, LDAP, SIP, and SS7.

The SOAP provisioning interface provides following data manipulation commands:

- Subscriber:
  - Subscriber Profile create/delete
  - Subscriber Profile field create/retrieve/modify/delete
  - Subscriber opaque data create/retrieve/modify/delete
    - Quota, State and Dynamic Quota
  - Reset of Subscriber Quota opaque data
- Pool:

- Pool Profile create/delete
- Pool Profile field create/retrieve/modify/delete
- Pool opaque data create/retrieve/modify/delete
  - Pool Quota, Pool State and Pool Dynamic Quota
- Pool subscriber membership operations
  - Add/remove from pool
  - Get pool subscriber membership
  - Get pool for subscriber

XML SOAP Application Server (XSAS)

The process interfacing to provisioning clients runs on every active NOAM&P server. The XSAS is responsible for:

- Accepting and authorizing SOAP provisioning client connections
- Processing and responding to SOAP requests received from provisioning clients
- Converting SOAP requests into internal XML format and converting responses from internal XML format to SOAP format
- Communicating (via internal XML) with the Provisioning Backend Application
- Updating the provisioning command log with requests sent and responses received

Provisioning Clients

Figure 2: Provisioning Interface Overview shows an overview of the provisioning interface.

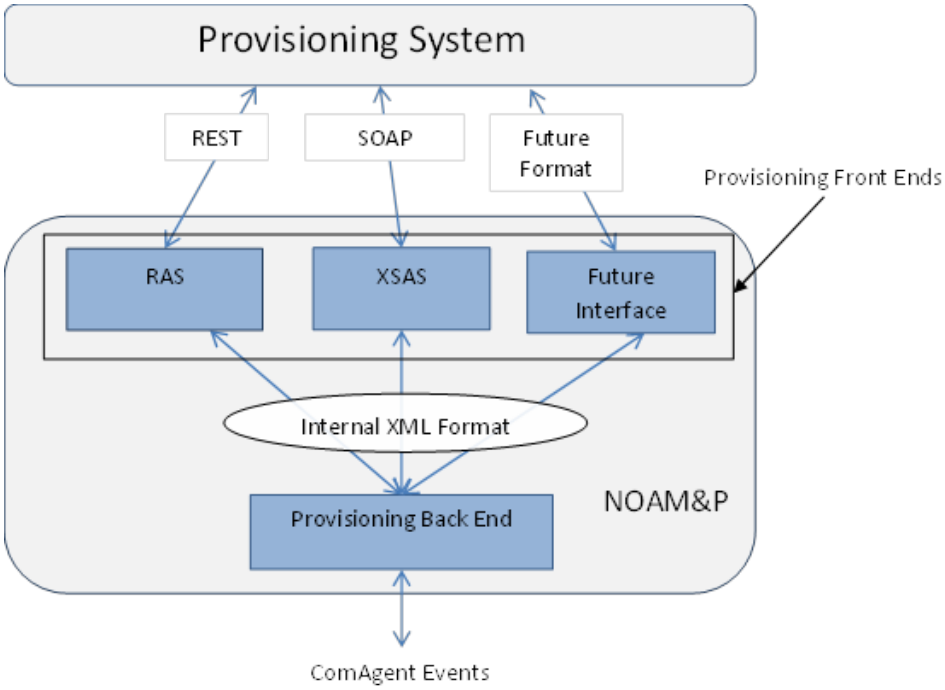

Figure 2: Provisioning Interface Overview

The XSAS provides connections to the customer provisioning systems (CPS). These are independent information systems supplied and maintained by the network operator to be used for provisioning the UDR system. Through the XSAS process, the CPS may add, delete, change or retrieve information about any subscriber or pool.

CPS's use SOAP to send requests to manipulate and query data in the Provisioning Database. Provisioning Clients establish TCP/IP connections to the XSAS running on the Active NOAM&P using the Primary NOAM&P's VIP.

Provisioning clients need to re-establish connections with the XSAS using the Primary UDR's VIP upon switchover from the Primary's Active to its Standby UDR server. Provisioning clients also need to redirect connections to the Secondary's VIP upon switchover from the Primary UDR site to the Disaster Recovery UDR 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 should drop the connection and re-establish it before trying again.

Provisioning clients are expected to re-send requests that resulted in a temporary error, or for which no response was received. The list of permanent/temporary error codes is described in *Error Codes*.

Security

For securing connections between the SOAP Interface and provisioning clients in an unsecure/untrusted network, a list of authorized IP addresses is provided.

The system configuration process maintains a white list of server IP addresses from which clients are authorized to establish a TCP/IP connection from.

The XSAS process verifies provisioning connections by utilizing the authorized IP address list. Any connect request coming from an IP address that is not on the list is denied (connection is immediately closed). All currently active connections established from an IP address which is removed from the Authorized IP list are immediately closed.

Multiple Connections

The XSAS process supports multiple connections, and each connection is considered persistent unless declared otherwise. The HTTP persistent connections do not use separate keep-alive messages, they just allow multiple requests to use a same TCP/IP connection. However, connections are closed after being idle for a time limit configured in idle timeout (see *Idle Timeout*).

The provisioning client establishes a new TCP/IP connection to XSAS process before sending the first SOAP command. After the execution of the request, the XSAS process sends a response message back, and keeps the connection alive as long as the next request comes before idle timeout.

Note: In order to achieve the maximum provisioning TPS rate that the UDR SOAP interface is certified for, multiple simultaneous provisioning connections are required.

- For example, if the certified maximum provisioning TPS rate is 200 TPS, and the `Maximum SOAP Connections` (see *SOAP Interface System Variables*) is set to 100, then up to 100 connections may be required in order to achieve 200 TPS. It is ***not*** possible to achieve the maximum provisioning TPS rate
- When calculating the provisioning TPS rate, if any `<tx>` transactions are sent (see section *Block Transaction Mode*), then the TPS rate is calculated using the number of requests contained in the `<tx>`. A `<tx>` request does ***not*** count as 1 TPS.

Request Queue Management

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

Incoming requests, whether multiple requests from a single client or requests from multiple clients, are not prioritized. Multiple requests on multiple connections 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) on a different connection, this would be responded to immediately, potentially before the any/some/all of the previous requests have been responded to.

Note: All requests from a client sent on a single connection are processed by UDR serially. Multiple requests can be sent without receiving a response, but each request is queued and not processed until the previous request has completed. A client can send multiple requests across multiple connections, and these may execute in parallel (but requests on *each connection* are still processed serially).

Database Transactions

Each create/update/delete request coming from SOAP interface triggers a unique database transaction, i.e. a database transaction started by a request is committed before sending a response.

Block Transaction Mode

The block database transaction mode requires explicit <tx> XML 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 the sequence supplied 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, all requests apart from the one that failed would have error code=1 (NOT PROCESSED) which indicates that the request was not performed or has been rolled back.

All block transactions must also satisfy limits indicated by the *Maximum Requests in SOAP <tx> XML*, *Maximum Provisioning Backend Response Timeout* and *Transaction Durability Timeout* system variables, which are defined in *SOAP Interface System Variables*. If any of those limits are exceeded, the transaction will be aborted and automatically rolled back.

If a block transaction is sent which contains more than *Maximum Requests in SOAP <txt> XML* requests, then the request will fail with a SOAP error <message error="20"> (see *Provisioning Interface Overview*).

Note: The relevant transaction related measurement(s) will be incremented once per <tx> request (i.e., by +1). The relevant request based measurements will be incremented once per request contained in the <tx>. All requests share the same outcome (e.g., if a <tx> request contained 5 requests, and the transaction was successful, then measurement *RxXsasProvMsgsSuccessful* will be incremented by 5). If the first 3 requests in the transaction were successful, and the 4th request failed, then the transaction will fail, get rolled back, and *RxXsasProvMsgsFailed* will be incremented by 5.

Request

```
<tx [resonly="resonly"]>
  <req ... >
    request1
  </req>
    <expr><attr name="keyName1"/><value val="keyValue1"/></expr>
[
  <req ...>
    request2
  </req>
    :
  <req ... >
    requestN
  </req>
]
</tx>
```

Table 2: Request Definitions: Block Transaction Mode

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response **Note:** Any *resonly* value supplied in the `<tx>` will take precedence on any *resonly* value supplied in a contained request within `<req>`. If no *resonly* value is supplied in the `<tx>`, then the value supplied in a contained request within `<req>` takes precedence. The default value for *resonly* when not supplied is "n".	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
requestX	SOAP XML Request contained in the transaction **Note:** The maximum number of requests that can be included in a `<tx>` transaction is defined in the *Maximum Requests in SOAP <tx> XML* system variable, which is defined in *SOAP Interface System Variables*	

Response

```
<tx nbreq="nbreq" [resonly="resonly"]>
  <req ...>
[    originalXMLRequest1 ]
    <res error="error" affected="affected"/>
  </req>
[
  <req ...>
[    originalXMLRequest2 ]
    <res error="error" affected="affected"/>
  </req>
  :
  <req ...>
[    originalXMLRequest3 ]
    <res error="error" affected="affected"/>
  </req>
]
</tx>
```

Table 3: Response Definitions: Block Transaction Mode

Variable	Definition	Value
nbreq	The number of requests contained in the original XML `<tx>` request	

Variable	Definition	Value
resonly	(Optional) The *resonly* value from the original XML `<tx>` request, if supplied	
originalXMLRequestN	(Optional) The text of the original XML request that was contained in the `<tx>` request, if necessary (see the request definition for *resonly*)	A string with 1 to 4096 characters
error	Error code indicating outcome of request. "0" means success. A value of "1" (NOT PROCESSED) indicates that the request was not performed or had been rolled back. Other values are dependent on the request being executed, and are listed in the description for that specific request.	
affected	The number of subscribers affected by the request. A value of "1" or more is expected for success **Note:** This value may be non-zero for requests that were valid within a transaction, but where a subsequent request failed, and the transaction was rolled back. The *affected* value is given to indicate the request would have been successful if committed	

Note: For a transaction to be considered successful, all *error* values in ***all*** request responses must be "0".

Note: Results for a "select" request may be returned even if the transaction failed. Based on the *error* values for all request responses, it is up to the provisioning client sending the request to use the returned information for the "select" if the transaction itself was not successful.

Examples

Request #1

This request creates 2 subscribers, and gets a 3rd subscriber.

```
<tx resonly="y">
  <req name="insert">
    <ent name="Subscriber"/>
```

```
      <set>
        <expr>
          <attr name="MSISDN"/>
          <value val="19195551234"/>
        </expr>
        <expr>
          <attr name="BillingDay"/>
          <value val="1"/>
        </expr>
        <expr>
          <attr name="Entitlement"/>
          <value val="DayPass,DayPassPlus"/>
        </expr>
      </set>
  </req>
    <req name="insert">
      <ent name="Subscriber"/>
      <set>
        <expr>
          <attr name="MSISDN"/>
          <value val="15141234567"/>
        </expr>
        <expr>
          <attr name="BillingDay"/>
          <value val="1"/>
        </expr>
        <expr>
          <attr name="Entitlement"/>
          <value val="DayPass,DayPassPlus"/>
        </expr>
      </set>
    </req>
    <req name="select">
      <ent name="Subscriber"/>
      <select>
        <expr>
          <attr name="IMSI"/>
        </expr>
        <expr>
          <attr name="MSISDN"/>
        </expr>
        <expr>
          <attr name="NAI"/>
        </expr>
      </select>
      <where>
        <expr>
          <attr name="IMSI"/>
          <op value="="/>
          <value val="302370123456789"/>
        </expr>
      </where>
   </req>
</tx>
```

Response #1

In this example, all requests were successful, and the transaction was committed.

```
<tx nbreq="3" resonly="y">
  <req name="insert">
    <res error="0" affected="1"/>
```

```
  </req>
  <req name="insert">
    <res error="0" affected="1"/>
  </req>
  <req name="select">
    <res error="0" affected="1"/>
    <rset>
      <row>
        <rv>302370123456789</rv>
        <rv>15145551234</rv>
        <rv>person@operator.com</rv>
      </row>
    </rset>
  </req>
</tx>
```

Response #2

In this example, the first request was successful, but the second request failed. The transaction was rolled back. The second request failed due to error FIELD NOT FOUND. The third command was not attempted.

```
<tx nbreq="3" resonly="y">
  <req name="insert">
    <res error="1" affected="1"/>
  </req>
  <req name="insert">
    <res error="70012" affected="0"/>
  </req>
  <req name="select">
    <res error="1" affected="0"/>
  </req>
</tx>
```

Response #3

In this example, the second request is invalid due to an unknown entity. The transaction was not attempted.

```
<tx nbreq="3" resonly="y">
  <req name="insert" resonly="y">
    <res error="1" affected="0"/>
  </req>
  <req name="insert" resonly="y">
    <res error="70000" affected="0"/>
  </req>
  <req name="select" resonly="y">
    <res error="1" affected="0"/>
  </req>
</tx>
```

Response #4

In this example, all requests are valid, but the commit of the transaction failed. The transaction was rolled back.

```
<tx nbreq="3" resonly="y">
```

```
  <req name="insert">
    <res error="70038" affected="1"/>
  </req>
  <req name="insert">
    <res error="70038" affected="1"/>
  </req>
  <req name="select">
    <res error="70038" affected="1"/>
    <rset>
      <row>
        <rv>302370123456789</rv>
        <rv>15145551234</rv>
        <rv>person@operator.com</rv>
      </row>
    </rset>
  </req>
</tx>
```

ACID-Compliant Transactions

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

Atomicity

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

Consistency

Data across all requests performed inside a transaction is consistent.

Isolation

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

Durability

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

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

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

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

Connection Management

It is possible to enable/disable/limit the SOAP provisioning interface in a number of different ways.

Connections Allowed

The configuration variable `Allow SOAP Provisioning Connections` (see *SOAP Interface System Variables*) controls whether SOAP interface connections are allowed to the configured port. If this variable is set to *NOT_ALLOWED*, then all existing connections are immediately dropped. Alarm 13026 (as described in section 0) is raised. Any attempts to connect are rejected.

When `Allow SOAP Provisioning Connections` is set back to *ALLOWED*, the alarm is cleared, and connections are accepted again.

Disable Provisioning

When the UDR 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 a SERVICE_UNAVAILABLE error indicating the service is unavailable.

For an example of a provisioning request/response when provisioning is disabled, see the last example in *Create Profile*.

Idle Timeout

The connection between a Provisioning client and the XSAS process is handled in a persistent fashion. The configuration variable `SOAP Interface Idle Timeout` (see *SOAP Interface System Variables*) indicates the time to wait before closing the connection due to inactivity (i.e. no requests are received).

Maximum Simultaneous Connections

The configuration variable `Maximum SOAP Connections` (see *SOAP Interface System Variables*) defines the maximum number of simultaneous SOAP Interface client connections.

TCP Port Number

The configuration variable `SOAP Interface Port` (see *SOAP Interface System Variables*) defines the SOAP Interface TCP Listening Port.

Behavior During Low Free System Memory

If the amount of free system memory available to the database falls below a critical limit, then requests that create or update data may fail with the error MEMORY_FULL. Before this happens, memory threshold alarms will be raised indicating the impending behavior if the critical level is reached.

The error returned by the SOAP interface when the critical level has been reached is :

```
<res error="70042" affected="0"/>
```

Chapter 3

SOAP Interface Description

Topics:

This chapter provides an overview of the SOAP-based provisioning interface.

Provisioning Interface Overview

The UDR supports a SOAP-based provisioning interface for management of subscriber data. This interface supports querying, creation, modification, and deletion of subscriber and pool data. The SOAP Messages and SOAP Replies are transported over the HTTP protocol.

Each SOAP Message/Reply contains a UDR format XML request/response. The following XML request types are supported:

- Update
- Insert
- Delete
- Select
- Operation

In SOAP messages, the authentication is part of the SOAP Envelope Header. But for UDR authentication information is only provided for backward compatibility, and any optional SOAP Envelope Header information provided within a legacy UDR format request (see *Legacy SPR Compatibility Mode*) for *UserName* and *Passwd* is ignored. The client's TCP/IP must be present in the white list of server addresses (see *Security*).

A SOAP provisioning client application is responsible for:

- Establishing a TCP/IP connection with the SOAP Server using the Primary UDR's VIP and the SOAP XSAS listening port (configurable by the UDR GUI, see *SOAP Interface System Variables*).
- Creating and sending SOAP request messages (see *Basic XML Message Format*) to the SOAP Server.
- Receiving and processing SOAP response messages (see *Basic XML Message Format*) received from the SOAP Server.
- Detecting and handling connection errors. It is recommended that the clients 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 *SOAP Interface System Variables*) configurable by the UDR GUI. If a connection attempt is made while connections are not allowed, the connection is rejected by XSAS. 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 *SOAP Interface System Variables*) configurable by the UDR GUI. If an attempt is made to connect more than the number of SOAP connections allowed, the connection is rejected by the SOAP Server.

The SOAP interface uses SOAP as wrapper of XML requests and responses. The detailed format of the request and response formats are shown below.

The format of a SOAP Request is:

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

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">

  <SOAP-ENV:Body>

    <ns1:processTransaction>

      <![CDATA[REQUEST]]>

    </ns1:processTransaction>

  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Example SOAP Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">

  <SOAP-ENV:Body>
    <ns1:processTransaction>
      <![CDATA[
        <req name="insert">
          <ent name="Subscriber"/>
          <set>
            <expr><attr name="MSISDN"/>
                  <value val="33628323201"/></expr>
            <expr><attr name="BillingDay"/>
                  <value val="12"/></expr>
          </set>
        </req>
      ]]>
    </ns1:processTransaction>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP interface uses the following wrapper for the XML response and error codes:

Note: Either the <ns1:message> or the <SOAP-ENV:Fault> element is present, but not both. The contents of the <SOAP-ENV:Fault> are dependent on the SOAP error that occurs and can vary, and thus are not shown here.

Example SOAP Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">

<
  <SOAP-ENV:Body>

    <ns1:message error="ErrorCode">
```

```
      <![CDATA[RESPONSE]]>

    </ns1:message>

  </SOAP-ENV:Body>
|
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <SOAP-ENV:Fault>
    ...
    </SOAP-ENV:Fault>

  </SOAP-ENV:Body>
>

</SOAP-ENV:Envelope>
```

Status Codes and Error Messages

If an error occurred in processing the request or with the format of the message, an error result code will be sent as shown below:

1. `<message error="0">`: normal, request transaction was sent and processed
2. `<message error ="0">` but the message content has " `<res error=error code number ... >` ". This implies there is a problem with the content of the request message (e.g., a problem with format or value out of range). The error code numbers are generated by UDR.
3. `<message error="10">`: Communication problem, unable to process the request transaction. The response does not contain any other response/error content.
4. `<message error="20">`: Unable to parse the request transaction. The response does not contain any other response/error content.

See *Error Codes* for a list of all supported error codes.

Example of a Response message indicating success:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="0">
      <![CDATA[<req name="insert" resonly="y">
                 <res error="0" affected="1"/>
               </req>]]>
    </ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of a Response message with an error code returned:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="0">
      <![CDATA[<req name="insert" resonly="y">
                 <res error="70019" affected="0"/>
               </req>]]>
    </ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of a Response message when a communications error occurred:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="10"></ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of a Response message when a request parsing failure occurred:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="20"></ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of a Response message when a SOAP Fault occurred:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>Method 'processTransaction' not implemented: method name or
namespace not recognized</faultstring>
      <detail></detail>
```

```
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Error Codes

The list of error codes is described in section *Error Codes*.

Legacy SPR Format SOAP Request/Response

UDR can be configured to operate in a compatibility mode for legacy SPR customers, which affects the SOAP request/response format. See *Legacy SPR Compatibility Mode* for more details.

Chapter 4

SOAP Interface Message Definitions

Topics:

This chapter describes the SOAP interface message syntax and parameters.

Message Conventions

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

Table 4: Message Conventions

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

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

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

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

Basic XML Message Format

Request

The following describes the basic layout of an XML request, with all different options and parameters included. UDR requests are made up of different combinations of the parameters. All are shown below for illustrative purposes. Proper examples of which parameters are relevant for each request are described in the section that follows.

```
<req name="reqname" [resonly="resonly"] [id="id"] [odk="odk"]>
  <ent name="entityname"/>

  <select>

    <expr><attr name="fieldName"/>

  </select>

  <set>

    <expr><attr name="fieldName"/><value val="fieldValue"/></expr>

   <expr><attr name="fieldName"/><op value="="/><value val="" isnull="y"/></expr>


    <oper name="AddToSet">
      <expr><attr name="setFieldName"/><value val="setFieldValue"/></expr>
    </oper>

    <oper name="RemoveFromSet">
      <expr><attr name="setFieldName"/><value val="setFieldValue"/></expr>
    </oper>

    <expr><attr name="cdataFieldName"/><op value="="/><cdata>
<![CDATA[
cdataFieldValue
]]></cdata></expr>

  </set>

  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
    <expr><attr name="rowKeyName"/><op value="="/><value
```

```
val="rowKeyValue"/></expr>
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
  </where>

  <oper name="operName">
    <expr><attr name="fieldName"/><value val="fieldValue"/></expr>
  </oper>

</req>
```

The *reqname* attribute indicates what type of request is being sent. Values are either *insert*, *update*, *delete*, *select*, or *operation*, depending on the request.

The *resonly* attribute controls whether or not the original request is included along with the response. The *resonly* attribute is optional, and if set to *y*, then the original request is NOT included in the response (i.e. **res**ult **only**). If *resonly* is set to *n*, then the original request IS included in the response. The default value of the flag (when the *resonly* attribute is not supplied) is *n* - i.e. return the request in the response.

The *id* attribute is used by the XSAS 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 XSAS in the corresponding response, so a unique *id* value must be sent in each request message to differentiate responses.

The odk attribute allows an insert request to convert the insert request to an update request if the target entity already exists. The odk attribute is optional, and if set to *yes*, then if the entity being inserted already exists, the entity will be updated instead of the request failing. The default value of the flag (for when the attribute is not supplied) is to not convert the insert request to an update request. Hence, if the target entity already exists, the request will fail.

The *entityname* attribute identifies the provisioning entity type on which the request is being performed on. Values are either *subscriber*, *pool*, *QuotaEntity*, or *PoolQuotaEntity* depending on the request, which should match the configured Entity values in the SEC.

The *namespace* attribute identifies the database namespace in which the data relating to the request is stored. This is not used in UDR, but is retained for backwards compatability. Value is always set to *policy*. This attribute is optional, and can be supplied for backwards compatibility with the legacy SPR. Any value supplied is not validated, and ignored.

When a field value is included to be set (for example in an insert/update request), a `<set>` element is present. Within this, zero, one, or many `<expr><attr name=" fieldName "/><value val=" fieldValue "/></expr >` element(s) are present. The *fieldName* indicates the name of the field being set, and the *fieldValue* is the value to set it to. When the value of a field should be deleted, this is performed by setting the *fieldValue* as empty (i.e. `""`), and additionally specifying the attribute is `null="y"` .

Note: When specifying fields in a `<set>` element, field order is not important. The fields defined for an entity do not have to be specified in the order they are defined in the SEC.

When a field value is included to be retrieved (for example in a select request), a `<select>` element is present. Within this, one, or many `<expr><attr name=" fieldName "/></expr>` element(s) are present. The *fieldName* indicates the name of the field being retrieved. For a select request, at least one field value must be requested. Only the fields requested will be returned in the response.

When a field is a list type (such as Entitlement in Profile), an embedded operation request is used to add/remove values from the list. This is performed by including the element `<oper name="`

operName"> where *operName* is either *AddToSet* (to add a value(s) to a list) or *RemoveFromSet* (to remove a value(s) from a list). The name of the field being modified is specified in *setFieldName*, and the value(s) being added/removed are specified in *setFieldValue*. Multiple values can be specified in *setFieldValue*, with each individual value in a separate <expr><attr name=" setFieldName "/><value val=" setFieldValue "/></expr> element.

Note: The *ns* attribute is optional, and can be supplied for backwards compatibility with the legacy SPR. Any value supplied is not validated, and ignored.

When a field is to be set as an XML data "blob", this is done by indicating the field *cdataFieldValue* contains a <cdata> element, and then including the data within the constructs of an XML CDATA section. The CDATA section starts with "<![CDATA[", then the *cdataFieldValue* containing the XML data "blob", and the CDATA section ends with "]]>".

Most commands identify the subscriber for which the provisoning request is being made by specifying the subscriber address in the <where> element. When present, a key type/value must be provided. Depending on the command, *keyType* can be *IMSI*, *MSISDN*, *NAI*, *AccountId*, or *PoolID*. The value of the key (of the indicated key type) is set in *keyValue*.

Depending on the *keyType*, the *keyValue*, is validated as below:

keyType	***keyValue* Validation**
IMSI	10-15 numeric digits
MSISDN	8-15 numeric digits. **Note:** A preceding '+' symbol is NOT supported, and will be rejected.
NAI	String in "user@domain" format
AccountId	1-255 characters
PoolID	1-22 numeric digits, minimum value 1

When a request is performing an action on a specific row in an entity (such as updating a field value in a specific quota instance), the row key field name used to select the row is specified in *rowKeyName*. The value of key is specified in *rowKeyValue*. If a field within the row can indicate uniqueness, in the case of more than one row having the same *rowKeyName/rowKeyValue*, then this field is specified in *instanceFieldName/instanceFieldValue*.

When the *reqname* is set to *operation*, the <oper> element is present. This defines the operation name in *operName*.

Response

The following describes the basic layout of an XML response, with all different options and parameters included. UDR responses are made up of different combinations of the parameters. All are shown below for illustrative purposes. Proper examples of which parameters are relevant for each response are described in the section that follows.

```
<req name="reqname" [resonly="resonly"] [id="id"]>
  originalXMLRequest
  <res error="error" affected="affected"/>
    <rset>
    <row>
```

```
        <rv>rowValue</rv>
        <rv>
<![CDATA[
cdataRowValue
]]>
        <rv></rv>
        <rv null="y"/>
    </row>
  </rset>
</req>
```

The `reqname` attribute contains the same value as supplied in the request. Values are either *insert*, *update*, *delete*, *select*, or *operation*, depending on the request.

If the `resonly` attribute was included in the request, the same value is returned in the response.

If the `id` attribute was included in the request, the same value is returned in the response.

The `originalXMLRequest` element is the text of the original XML request that was sent. This is only present if the `resonly="n"` attribute is set in the original request (or the *resonly* attribute was not supplied, as the default value is n).

The `error` attribute indicates the outcome of the request. A value of "*0*" indicates success. Any other value indicates failure. The possible errors for each request are detailed in the following section for each request. The list of error codes is described in *Error Codes*.

The `affected` attribute indicates the number of affected subscribers. A value of "*1*" (or more) is expected (for success) and "`0`" for failure.

If a select request has been performed (or with some operation requests), result data returned is contained within a `<rset>` ("rowset") element. Within an `<rset>` can be zero (if no matching data was found) or one `<row>` element (UDR does not currently support returning multiple `<row>` elements). Within a `<row>` are one or more `<rv>` (row value) elements containing a *rowValue* detailing the requested field value. One `<rv>` element corresponds for every fieldValue requested in the select request. The `<rv>` elements are given in the same order as the *fieldValues* are specified.

Note: An `<rv>` element can contain an entire XML CDATA section, starting with "`<![CDATA[ `", then the *cdataRowValue* containing the XML data "blob", and the CDATA section ends with "`]]>`". If the `<rv>` element represents a valid field that is not present in the XML data "blob", then this is indicated with `<rv null= "y">`. If the field is present in the XML blob, but has an empty value, this is indicated with `<rv></rv>`.

Note: Whenever XML blob data is returned, fields may not be returned in the order they are defined in the SEC. The fields may be returned in any order.

Encoding of Multiple Embedded CDATA Sections

Requests and responses may contain multiple embedded CDATA sections - i.e. one CDATA section that completely contains another CDATA section, because the SOAP envelope begins with a CDATA section to contain the XML requests/responses. These requests/responses require special formatting.

Subscriber Provisioning and *Subscriber Profile Commands* describe CDATA sections within the UDR commands without any reference on how these should be represented once they embedded in the SOAP envelope.

The following sections here give examples of the complete SOAP HTTP requests/response to indicate how to format requests when requests are sent to UDR by a provisioning client, and how responses returned by UDR will be returned to the UDR client.

Request

When physically encoding the XML data to be sent, all embedded CDATA start and end sequences must be changed (the opening and closing sequences for the initial CDATA within the `<message>` element does not need to be changed).

- Replace all embedded occurances of '`<![CDATA[` ' with "`&lt;![CDATA[` "
- Replace all embedded occurances of '`]]` ' with "`]]&gt;` "

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:processTransaction>
      <![CDATA[
        <req name="insert" resonly="y">
          <ent name="Subscriber"/>
          <set>
            <expr><attr name="QuotaEntity"/>
              <op value="="/><cdata>&lt;![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9999</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
              ]]&gt;</cdata>
            </expr>
          </set>
          <where>
            <expr><attr name="MSISDN"/><op value="="/><value
val="13123654862"/></expr>
          </where>
        </req>
      ]]>
    </ns1:processTransaction>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response

When a response is received, every '<' and '>' character within the <message> element will have been replaced with "<" and ">" respectively (including for the initial CDATA).

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="0">
      &lt;![CDATA[
        &lt;?xml version="1.0" encoding="UTF-8"?&gt;
        &lt;req name="select" resonly="y"&gt;
        &lt;res affected="1" error="0"/&gt;
          &lt;rset&gt;
            &lt;row&gt;
              &lt;rv&gt;
&lt;![CDATA[
&lt;?xml version="1.0"?&gt;
  &lt;usage&gt;
    &lt;version&gt;1&lt;/version&gt;
    &lt;quota name="AggregateLimit"&gt;
      &lt;cid&gt;9999&lt;/cid&gt;
      &lt;time&gt;3422&lt;/time&gt;
      &lt;totalVolume&gt;1000&lt;/totalVolume&gt;
      &lt;inputVolume&gt;980&lt;/inputVolume&gt;
      &lt;outputVolume&gt;20&lt;/outputVolume&gt;
      &lt;serviceSpecific&gt;12&lt;/serviceSpecific&gt;
      &lt;nextResetTime&gt;2010-05-22T00:00:00-05:00&lt;/nextResetTime&gt;
    &lt;/quota&gt;
  &lt;/usage&gt;
]]&gt;
              &lt;/rv&gt;
            &lt;/row&gt;
          &lt;/rset&gt;
        &lt;/req&gt;
      ]]&gt;
    </ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Case Sensitivity

The constructs that XML requests are made up of (such as <req>, <ent>, <set>, <where>, etc.) are case-sensitive. Exact case must be followed for all the commands described in this document, or the request will fail.

For example, the following is valid:

```
<req name="delete">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
```

```
  </where>
</req>
```

But the following is NOT:

```
<req name="delete">
  <Ent name="Subscriber"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Entity names ***are*** case-sensitive, for example "Subscriber", "QuotaEntity", and "Pool".

Entity field names and key names ***are not*** case-sensitive, for example *fieldName*, *setFieldName, keyName* and *instanceFieldName*.

Entity field values, key values, and row identifiers ***are*** case-sensitive, for example *fieldValue*, *setFieldValue*, *keyValue*, *rowIdName*, *rowIdValue*,and *instanceFieldValue*.

Examples:

- When accessing a *fieldName* defined as "inputVolume" in the SEC, then "inputvolume", "INPUTVOLUME" or "inputVolume" ***are*** valid field names. Field names do not have to be specified in a request as they are defined in the SEC
- When a field is returned in a response, it is returned ***as defined in the SEC*** . For example, if the above field is created using the name "INPUTVOLUME", then it will be returned in a response as "inputVolume"
- When a *fieldValue* is used to find a field (such as when using the "Delete Field Value" command), the field value ***is*** case-sensitive. If a multi-value field contained the values "DayPass,Weekend,Evening" and the Delete Field Value command was used to delete the value "WEEKEND", then this would fail
- When an attribute in the XML blob contains the row identifier name - aka *rowIdName* (for example for Quota, the element <quota name="AggregateLimit"> contains the attribute called "name") the row identifier name ***is*** case-sensitive, and must be specified as defined
- When a *rowIdValue* is used to find a row (such as when using the "Get Row" command), the row identifier value ***is*** case-sensitive. If an entity contained a row called "DayPass", and the Get Row command was used to get the row "DAYPASS", then this would fail
- When a *instanceFieldName* is used to find a row (such as when using the "Get Row" command), the row instance identifier field name is ***not*** case-sensitive
- When a *instanceFieldValue* is used to find a row (such as when using the "Get Row" command), the row instance identifier field value ***is*** case-sensitive. If an entity contained a row called with a field with the value "Data", and the Get Row command was used to get the row with the field value "DATA", then this would fail
- When a *keyName* is specified in a <where> or <set> element (such as MSISDN), the key name is ***not*** case-sensitive
- When a *keyValue* is specified in the <where> element (such as for an NAI), the value ***is*** case-sensitive. For example, for a subscriber with an NAI of "mum@foo.com", then "Mum@foo.com" or "MUM@FOO.COM" will ***not*** find the subscriber

List of Messages

The following table provides a list of operations/messages for subscriber data. Each row of the table represents a command.

Table 5: Summary of Supported Subscriber Commands

Operation Data/Type	Command	SOAP
Subscriber Profile	Create Profile	insert
	Get Profile	select
	Delete Profile	delete
Subscriber Field	Add Field Value	update (using "AddToSet" operation)
	Get Field	select
	Update Field	update
	Delete Field	update, null
	Delete Field Value	update (using "RemoveFromSet" operation)
Subscriber Opaque Data	Create Opaque Data	insert
	Get Opaque Data	select
	Update Opaque Data	update
	Delete Opaque Data	update (using "isnull" attribute)
Subscriber Data Row	Create Row	insert
	Get Row	select
	Delete Row	delete
Subscriber Data Row Field	Get Row Field	select
	Update Row Field	update
	Delete Row Field	update (using "isnull" attribute)
Subscriber Special Operation Commands	Reset Quota	operation

The following table provides a list of operations/messages for pool data. Similar to the previous table, each row of the table represents a command.

Table 6: Summary of Supported Pool Commands

Operation Data/Type	Command	SOAP
Pool Profile	Create Pool	insert
	Get Pool	select
	Delete Pool	delete
Pool Field	Add Field Value	update (using "AddToSet" operation)
	Get Field	select
	Update Field	update
	Delete Field	update (using "isnull" attribute)
	Delete Field Value	update (using "RemoveFromSet" operation)
Pool Opaque Data	Create Opaque Data	insert
	Get Opaque Data	select
	Update Opaque Data	update
	Delete Opaque Data	update (using "isnull" attribute)
Pool Data Row	Create Row	insert
	Get Row	select
	Delete Row	delete
Pool Data Row Field	Get Row Field	select
	Update Row Field	update
	Delete Row Field	update (using "isnull" attribute)
Additional Pool Commands	Add Member to Pool	operation
	Remove Member from Pool	operation
	Get Pool Members	operation
	Get Pool by Member (key)	Operation

Chapter 5

UDR Data Model

Topics:

This chapter describes the UDR data model.

Data Model Overview

The UDR is a system used for the storage and management of subscriber policy control data. The UDR functions as a centralized repository of subscriber data for the PCRF.

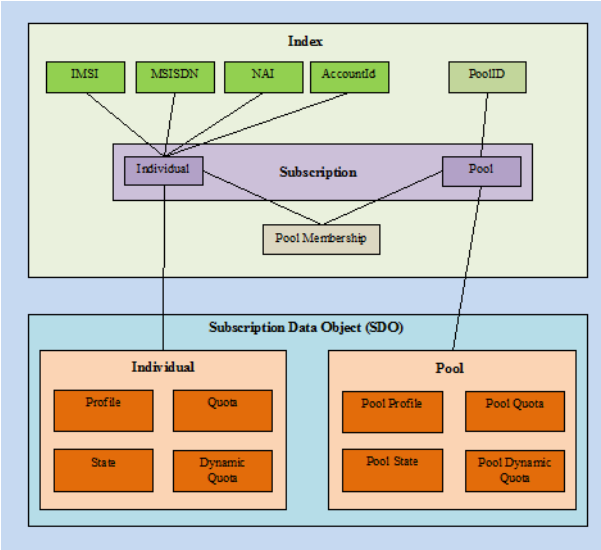

Figure 3: UDR Data Model

The subscriber-related data includes:

- Profile/Subscriber Data: pre-provisioned information that describes the capabilities of each subscriber. This data is typically written by the customer's OSS system (via a provisioning interface) and referenced by the PCRF (via the Sh interface).
- Quota: information that represents the subscriber's use of managed resources quota, pass, top-up, roll-over). Although the UDR provisioning interfaces allow quota data to be manipulated, this data is typically written by the PCRF and only referenced using the provisioning interfaces.
- State: subscriber-specific properties. Like quota, this data is typically written by the PCRF, and referenced using the provisioning interfaces.
- Dynamic Quota: dynamically configured information related to managed resources (pass, top-up, roll-over). This data may be created or updated by either the provisioning interface or the Sh interface.
- Pool Membership: The pool to which the subscriber is associated. The current implementation allows a subscriber to be associated with a single pool, although the intention is to extend this to multiple pools in the future.

The UDR can also be used to group subscribers using Pools. This feature allows wireless carriers to offer pooled or family plans that allow multiple subscriber devices with different subscriber account IDs, such as MSISDN, IMSI, or NAI to share one quota.

The pool-related data includes:

- Pool Profile: pre-provisioned information that describes a pool
- Pool Quota: information that represents the pool's use of managed resources (quota, pass, top-up, roll-over)

- Pool State: pool-specific properties
- Pool Dynamic Quota: dynamically configured information related to managed resources (pass, top-up, roll-over)
- Pool Membership: list of subscribers that are associated with a pool

The data architecture supports multiple Network Applications. This flexibility is achieved though implementation of a number of registers in a Subscriber Data Object (SDO) and storing the content as Binary Large Objects (BLOB). An SDO exists for each individual subscriber, and an SDO exists for each pool.

The Index contains information on the following:

- Subscription:
  - A subscription exists for every individual subscriber, and for every pool
  - Maps a subscription to the user identities through which it can be accessed
  - Maps an individual subscription to the pool of which they are a member
- User Identities:
  - Use to map a specific user identity to a subscription
    - IMSI, MSISDN, NAI and AccountId map to an individual subscription
    - PoolID maps to a pool subscription
- Pool membership
  - Maps a pool to the list of the individual subscriber members
- The Subscription Data Object (SDO):

  An SDO record contains a list of registers, holding a different type of entity data in each register. An SDO record exists for:
  - Each individual subscriber
    - Defined entities stored in the registers are:
      - Profile
      - Quota
      - State
      - Dynamic Quota
  - Each pool
    - Defined entities stored in the registers are:
      - Pool Profile
      - Pool Quota
      - Pool State
      - Pool Dynamic Quota

Provisioning applications can create, retrieve, modify, and delete subscriber/pool data. The indexing system allows to access Subscriber SDO via IMSI,MSISDN,NAI or AccountId. The pool SDO can be accessed via PoolD.

A field within an entity can be defined as mandatory, or optional. A mandatory field must exist, and cannot be deleted.

A field within an entity can have a default value. If an entity is created, and the field is not specified, it will be created with the default value.

A field within an entity can be defined so that once created, it cannot be modified. Any attempt to update the field once created will fail.

A field within an entity can have a reset value. If a reset command is used on the entity, those fields with a defined reset value will be set to the defined value. This is currently only applicable to field values within a row for the Quota entity.

Subscriber Data

Subscriber Profile

The Subscriber profile represents the identifying attributes associated with the user. In addition to the base fields indicated their level of service, it also includes a set of custom fields that the customer's provisioning system can use to store information associated with the subscriber. The values in custom fields are generally set by the customer's OSS and are read by the PCRF for use in policies.

The Subscriber profile shall support the following sequence of attributes. Each record must have at least one of the following key values: MSISDN, IMSI, NAI, AccountId.

BillingDay must be defined with a default value if another value is not specified. The remaining fields are optional, based on the description provided for each.

Note: UDR only supports an MSISDN with 8-15 numeric digits. A preceeding '+' symbol is NOT supported, and will be rejected.

Table 7: Subscriber Profile Fields

Name (xml tag)	Type	Description
subscriber	---	Sequence (multiplicty=1)
MSISDN	String	Subscriber's MSISDN (8-15 numeric digits)
IMSI	String	Subscriber's IMSI (10-15 numeric digits)
NAI	String	Subscriber's NAI (in format "user@domain"
AccountId	String	Any string that can be used to identify the account for the subscriber.

Name (xml tag)	Type	Description
BillingDay	String	Allowed values are [0-31]. The day of the month [1-31] on which the subscriber's associated quota should be reset. [0] indicates that the default value configured at the PCRF level should be used. This is automatically set in any record where BillingDay is not specified.
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the subscriber's profile.
Tier	String	Subscriber's tier.
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.
Custom3	String	Fields used to store customer-specific data.
Custom4	String	Fields used to store customer-specific data.
Custom5	String	Fields used to store customer-specific data.
Custom6	String	Fields used to store customer-specific data.
Custom7	String	Fields used to store customer-specific data.
Custom8	String	Fields used to store customer-specific data.
Custom9	String	Fields used to store customer-specific data.
Custom10	String	Fields used to store customer-specific data.
Custom11	String	Fields used to store customer-specific data.
Custom12	String	Fields used to store customer-specific data.

Name (xml tag)	Type	Description
Custom13	String	Fields used to store customer-specific data.
Custom14	String	Fields used to store customer-specific data.
Custom15	String	Fields used to store customer-specific data.
Custom16	String	Fields used to store customer-specific data.
Custom17	String	Fields used to store customer-specific data.
Custom18	String	Fields used to store customer-specific data.
Custom19	String	Fields used to store customer-specific data.
Custom20	String	Fields used to store customer-specific data.

Quota

The Quota entity is used by the PCRF to record the current resource usage associated with a subscriber. A quota entity may contain multiple quota elements, each one tracking a different resource.

The Quota entity shall be associated with a subscriber record and supports the following sequence of attributes:

Note: The Quota entity contains a version number. Different attributes maybe be present based on the version number value of the entity being accessed. In this product, only v3 of Quota is supported.

Note: The default value given in the table is used either:

- When a Quota instance is created, and no value is supplied for the field. In this case, the field is created with the value indicated.
- When a Quota instance is reset using the "Reset Quota" command. Each field listed below is set to the value indicated. If the field does not currently exist in the Quota, it is created.

Table 8: Quota Instance Default Values

Name (xml tag)	Type	Default Value	Description	Quota Versions
usage	---	---	Sequence (multiplicity = 1)	
version	String	---	Version of the schema	
quota	---	---	Sequence (multiplicity = N)	

Name (xml tag)	Type	Default Value	Description	Quota Versions
name	String	---	Quota name (identifier)	1/2/3
cid	String	---	Internal identifier used to identity a quota within a subscriber profile.	1/2/3
time	String	Empty string ""	Tracks the time-based resource consumption for a Quota.	1/2/3
totalVolume	String	"0"	Tracks the bandwidth volume-based resource consumption for a Quota.	1/2/3
inputVolume	String	"0"	Tracks the upstream bandwidth volume-based resource consumption for a Quota.	1/2/3
outputVolume	String	"0"	Tracks the downstream bandwidth volume-based resource consumption for a Quota.	1/2/3
serviceSpecific	String	Empty string ""	Tracks service-specific resource consumption for a Quota.	1/2/3
nextResetTime	String	Empty string ""	Indicates the time after which the usage counters need to be reset. See below for date/time format	1/2/3
Type	String	Empty string ""	Type of the resource in use.	2/3
grantedTotalVolume	String	"0"	Represents the granted total volume of all the subscribers in the pool, in case of pool quota. In case of individual quota, it will represent the granted volume to all the PDN connections for that subscriber	2/3

Name (xml tag)	Type	Default Value	Description	Quota Versions
grantedInputVolume	String	"0"	Granted Input Volume	2/3
grantedOutputVolume	String	"0"	Granted Output Volume	2/3
grantedTime	String	Empty string ""	Granted Total Time	2/3
grantedServiceSpecific	String	Empty string ""	Granted Service Specific Units	2/3
QuotaState	String	Empty string ""	State of the resource in use.	3
RefInstanceId	String	Empty string ""	Instance-id of the associated provisioned pass, top-up or roll-over.	3

Note: Date/Timestamp format is:

CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm]

where:

- - = years before 0001
- CC = century
- YY = year
- MM = month
- DD = day
- T = Date/Time separator
- hh = hour
- mm = minutes
- ss = seconds
- Z = UTC (Coordinated Universal Time)
- +|- = time offset from UTC

The format has a regular expression along the lines of (excluding time zone part):

[0-9][0-9][0-9][0-9]\-[0-9][0-9]\-[0-9][0-9]T[0-9][0-9]:[0-9][0-9]:[0-9][0-9]

State

The State entity is written by the PCRF to store the state of various properties managed as a part of the subscriber's policy. Each subscriber may have a state entity. Each state entity may contain multiple properties.

The State entity contains a version number. Different attributes maybe be present based on the version number value of the entity being accessed. In UDR, there is only one version number of 1.

The State entity shall support the following sequence of attributes:

Table 9: Supported Attribute Sequences

Name (xml tag)	Type	Description
state	---	Sequence (multiplicity=1)
version	String	Version of the schema
property	---	Sequence (multiplicity = N)
name	String	The property name.
value	String	Value associated with the given property.

Dynamic Quota

The DynamicQuota entity records usage associated with passes, top-ups, and roll-overs. The DynamicQuota entity is associated with the Subscriber profile and may be created or updated by either the PCRF or the customer's OSS system.

The DynamicQuota entity contains a version number. Different attributes maybe be present based on the version number value of the entity being accessed. In UDR, there is only one version number of 1.

The DynamicQuota entity shall support the following sequence of attributes:

Table 10: Dynamic Quota Sequence of Attributes

Name (xml tag)	Type	Description
definition	---	Sequence (multiplicity = 1)
version	String	Version of the schema
DynamicQuota	---	Sequence (multiplicity = N)
Type	String	Identifies the dynamic quota type.
name	String	The class identifier for a pass or top-up. This name will be used to match top-ups to quota definitions on the PCRF. This name will be used in policy conditions and actions on the PCRF.
InstanceId	String	A unique identifier to identify this instance of a dynamic quota object.
Priority	String	An integer represented as a string. This number allows service providers to specify when one pass or top-up should

Name (xml tag)	Type	Description
		be used before another pass or top-up.
InitialTime	String	An integer represented as a string. The number of seconds initially granted for the pass/top-up.
InitialTotalVolume	String	An integer represented as a string. The number of bytes of total volume initially granted for the pass/top-up.
InitialInputVolume	String	An integer represented as a string. The number of bytes of input volume initially granted for the pass/top-up.
InitialOutputVolume	String	An integer represented as a string. The number of bytes of output volume initially granted for the pass/top-up.
InitialServiceSpecific	String	An integer represented as a string. The number of service specific units initially granted for the pass/top-up.
activationdatetime	String	The date/time after which the pass or top-up may be active. See below for date/time format
expirationdatetime	String	The date/time after which the pass or top-up is considered to be exhausted See below for date/time format
purchasedatetime	String	The date/time when a pass was purchased See below for date/time format
Duration	String	The number of seconds after first use in which the pass must be used or expired. If both Duration and expirationdatetime are present, the closest expiration time is used
InterimReportingInterval	String	The number of seconds after which the GGSN/DPI/Gateway should revalidate quota grants with the PCRF

Note: Date/Timestamp format is:

CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm]

where:

- - = years before 0001
- CC = century
- YY = year
- MM = month
- DD = day
- T = Date/Time separator
- hh = hour
- mm = minutes
- ss = seconds
- Z = UTC (Coordinated Universal Time)
- +|- = time offset from UTC

The format has a regular expression along the lines of (excluding time zone part):

[0-9][0-9][0-9][0-9]\-[0-9][0-9]\-[0-9][0-9]T[0-9][0-9]:[0-9][0-9]:[0-9][0-9]

Pool Data

Pool Profile

The Pool profile includes a set of custom fields that the customer's provisioning system can use to store information associated with the pool. The values in custom fields are generally set by the customer's OSS and are read by the PCRF for use in policies.

Each pool profile must have a unique key value called *PoolID*.

BillingDay must be defined with a default value if another value is not specified. The remaining fields are optional, based on the description provided for each.

The Pool profile record consists of the following sequence of attributes.

Table 11: Pool Profile Fields

Name (xml tag)	Type	Description
pool	---	Sequence (multiplicty=1)
PoolID	String	Pool identifier (1-22 numeric digits, minimum value 1)

Name (xml tag)	Type	Description
BillingDay	Uint8	The day of the month [1-31] on which the pool's associated quota should be reset. [0] indicates that the default value configured at the PCRF level should be used.
BillingType	String	The billing frequency, monthly, weekly, daily
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the pool's profile.
Tier	String	Subscriber's tier.
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.
Custom3	String	Fields used to store customer-specific data.
Custom4	String	Fields used to store customer-specific data.
Custom5	String	Fields used to store customer-specific data.
Custom6	String	Fields used to store customer-specific data.
Custom7	String	Fields used to store customer-specific data.
Custom8	String	Fields used to store customer-specific data.
Custom9	String	Fields used to store customer-specific data.
Custom10	String	Fields used to store customer-specific data.
Custom11	String	Fields used to store customer-specific data.
Custom12	String	Fields used to store customer-specific data.
Custom13	String	Fields used to store customer-specific data.

Name (xml tag)	Type	Description
Custom14	String	Fields used to store customer-specific data.
Custom15	String	Fields used to store customer-specific data.
Custom16	String	Fields used to store customer-specific data.
Custom17	String	Fields used to store customer-specific data.
Custom18	String	Fields used to store customer-specific data.
Custom19	String	Fields used to store customer-specific data.
Custom20	String	Fields used to store customer-specific data.

Pool Quota

The PoolQuota entity records usage associated with quotas, passes, top-ups, and roll-overs associated with the pool. The PoolQuota entity is associated with the Pool Profile and may be created or updated by either the PCRF or the customer's OSS system.

The PoolQuota entity contains a version number. Different attributes maybe be present based on the version number value of the entity being accessed. In UDR, there is only one version number of 3.

The PoolQuota entity consists of the following sequence of attributes:

Note: The default value given in the table is used when a PoolQuota instance is created, and no value is supplied for the field. In this case, the field is created with the value indicated.

Table 12: Pool Quota Fields

Name (xml tag)	Type	Size	Default Value	Description
usage	---	---	---	Sequence (multiplicity = 1)
version	String	8	---	Version of the schema
quota	---	---	---	Sequence (multiplicity = N)
name	String	255		Quota name (identifier)
cid	String	255		Internal identifier used to identity a quota within a subscriber profile.
time	String	255	Empty string “”	Tracks the time-based resource consumption for a Quota.

Name (xml tag)	Type	Size	Default Value	Description
totalVolume	String	255	"0"	Tracks the bandwidth volume-based resource consumption for a Quota.
inputVolume	String	255	"0"	Tracks the upstream bandwidth volume-based resource consumption for a Quota.
outputVolume	String	255	"0"	Tracks the downstream bandwidth volume-based resource consumption for a Quota.
serviceSpecific	String	255	Empty string ""	Tracks service-specific resource consumption for a Quota.
nextResetTime	String	255	Empty string ""	When set, it indicates the time after which the usage counters need to be reset. See below for date/time format
Type	String	255	Empty string ""	Type of the resource in use.
grantedTotalVolume	String	255	"0"	Represents the granted total volume of all the subscribers in the pool, in case of pool quota. In case of individual quota, it will represent the granted volume to all the PDN connections for that subscriber.
grantedInputVolume	String	255	"0"	Granted Input Volume
grantedOutputVolume	String	255	"0"	Granted Output Volume
grantedTime	String	255	Empty string ""	Granted Total Time
grantedServiceSpecific	String	255	Empty string ""	Granted Service Specific Units
QuotaState	String	255	Empty string ""	State of the resource in use.
RefInstanceId	String	255	Empty string ""	Instance-id of the associated provisioned pass, top-up or roll-over.

Note: Date/Timestamp format is:

CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm]

where:

- - = years before 0001
- CC = century
- YY = year
- MM = month
- DD = day
- T = Date/Time separator
- hh = hour
- mm = minutes
- ss = seconds
- Z = UTC (Coordinated Universal Time)
- +|- = time offset from UTC

The format has a regular expression along the lines of (excluding time zone part):

[0-9][0-9][0-9][0-9]\-[0-9][0-9]\-[0-9][0-9]T[0-9][0-9]:[0-9][0-9]:[0-9][0-9]

Pool State

The PoolState entity is written by the PCRF to store the state of various properties managed as a part of the pool's policy. Each pool profile may have a PoolState entity. Each PoolState entity may contain multiple properties.

The PoolState entity contains a version number. Different attributes maybe be present based on the version number value of the entity being accessed. In UDR, there is only one version number of 1.

The PoolState entity consists of the following sequence of attributes:

Table 13: Supported Pool State Attribute Sequences

Name (xml tag)	Type	Description
state	---	Sequence (multiplicity=1)
version	String	Version of the schema
property	---	Sequence (multiplicity = N)
name	String	The property name.
value	String	Value associated with the given property.

Pool Dynamic Quota

The PoolDynamicQuota entity records usage associated with passes, top-ups, and roll-overs associated with the pool. The PoolDynamicQuota entity is associated with the Pool Profile and may be created or updated by either the PCRF or the customer's OSS system.

The PoolDynamicQuota entity contains a version number. Different attributes maybe be present based on the version number value of the entity being accessed. In UDR, there is only one version number of 1.

The PoolDynamicQuota entity shall support the following sequence of attributes:

Table 14: Pool Dynamic Quota Sequence of Attributes

Name (xml tag)	Type	Description
definition	---	Sequence (multiplicity = 1)
version	String	Version of the schema
DynamicQuota	---	Sequence (multiplicity = N)
Type	String	Identifies the dynamic quota type.
name	String	The class identifier for a pass or top-up. This name will be used to match top-ups to quota definitions on the PCRF. This name will be used in policy conditions and actions on the PCRF.
InstanceId	String	A unique identifier to identify this instance of a dynamic quota object.
Priority	String	An integer represented as a string. This number allows service providers to specify when one pass or top-up should be used before another pass or top-up.
InitialTime	String	An integer represented as a string. The number of seconds initially granted for the pass/top-up.
InitialTotalVolume	String	An integer represented as a string. The number of bytes of total volume initially granted for the pass/top-up.
InitialInputVolume	String	An integer represented as a string. The number of bytes of input volume initially granted for the pass/top-up.
InitialOutputVolume	String	An integer represented as a string. The number of bytes of output volume initially granted for the pass/top-up.
InitialServiceSpecific	String	An integer represented as a string. The number of service specific units initially granted for the pass/top-up.
activationdatetime	String	The date/time after which the pass or top-up may be active. See below for date/time format
expirationdatetime	String	The date/time after which the pass or top-up is considered to be exhausted See below for date/time format
purchasedatetime	String	The date/time when a pass was purchased See below for date/time format

Name (xml tag)	Type	Description
Duration	String	The number of seconds after first use in which the pass must be used or expired. If both Duration and expirationdatetime are present, the closest expiration time is used
InterimReportingInterval	String	The number of seconds after which the GGSN/DPI/Gateway should revalidate quota grants with the PCRF.

Note: Date/Timestamp format is:

CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm]

where:

- - = years before 0001
- CC = century
- YY = year
- MM = month
- DD = day
- T = Date/Time separator
- hh = hour
- mm = minutes
- ss = seconds
- Z = UTC (Coordinated Universal Time)
- +|- = time offset from UTC

The format has a regular expression along the lines of (excluding time zone part):

[0-9][0-9][0-9][0-9]\-[0-9][0-9]\-[0-9][0-9]T[0-9][0-9]:[0-9][0-9]:[0-9][0-9]

Chapter 6

Subscriber Provisioning

Topics:

This chapter describes subscriber provisioning commands.

Subscriber Profile Commands

Note: For command responses, the error code values described are listed in *Error Codes*.

Table 15: Summary of Subscriber Profile Commands

Command	Description	Key(s)	Command Syntax
Create Profile	Create a new subscriber/ subscriber Profile	-	`<req name="insert"> <ent name="Subscriber"/>`
Get Profile	Get subscriber Profile data	MSISDN, IMSI, NAI or AccountId	`<req name="select"> <ent name="Subscriber"/>`
Delete Profile	Delete all subscriber Profile data and all opaque data associated with the subscriber	MSISDN, IMSI, NAI or AccountId	`<req name="delete"> <ent name="Subscriber"/>`

Create Profile

This operation creates a new subscriber profile using the field-value pairs that are specified in the request content.

Unlike other subscriber commands, *keyName* and *KeyValue* are not specified in the request as part of the "where" tag. Request content includes at least one key value (and up to 4 different key types), and field-value pairs, all as specified in the Subscriber Entity Configuration.

Note: All key values (IMSI/MSISDN/NAI/AccountId) should be specified identically in both the <key> section and in the Profile XML blob. The values specified in the <key> section are used to create the subscriber and define what values are used in the <key> section for subsequent requests. The values in the Profile XML blob are simply stored and returned if requested.

Note: The subscriber profile data provided is fully validated against the definition in the SEC. If the validation check fails, then the request is rejected.

Note: Multi-value fields can be specified by a single *fieldNameX* value with a delimited list of values, or multiple *fieldNameX* fields, each containing a single value.

Prerequisites

A subscriber with any of the keys supplied in the Profile must not exist.

Request

```
<req name="insert" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="keyName1"/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyName4"/><value val="keyValueN"/></expr>
]
[
    <expr>
<
      <attr name="fieldName1"/><value val="fieldValue1"/>
|
      <attr name="cdataFieldName1"/><op value="="/>
        <cdata><![CDATA[cdataFieldValue1]]></cdata>
>
    </expr>

    <expr>
<
      <attr name="fieldName2"/><value val="fieldValue2"/>
|
      <attr name="cdataFieldName2"/><op value="="/>
        <cdata><![CDATA[cdataFieldValue2]]></cdata>
>
    </expr>
    :
    <expr>
<
      <attr name="fieldNameN"/><value val="fieldValueN"/>
|
      <attr name="cdataFieldNameN"/><op value="="/>
        <cdata><![CDATA[cdataFieldValueN]]></cdata>
>
    </expr>
]
  </set>
</req>
```

Table 16: Request Variable Definitions: Create Profile

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
keyNameX	A key field within the subscriber Profile	• IMSI • MSISDN

Variable	Definition	Value
		• NAI • AccountId
keyValueX	Corresponding key field value assigned to *keyNameX*	
fieldNameX	A user defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to *fieldNameX*	
cdataFieldNameX	A user defined field within the subscriber Profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC	• Quota • State • DynamicQuota
cdataFieldValueX	Contents of the XML data "blob" for *cdataFieldNameX*	

Note: One key is mandatory. Any combination of key types are allowed. More than one occurrence of each key type (i.e. IMSI/MSISDN/NAI/AccountId) is supported, up to an engineering configured limit.

Note: Key/field order in the request is not important.

Response

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 17: Response Variable Definitions: Create Profile

Variable	Definition	Values
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	

Variable	Definition	Values
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Create Profile* for other values.	
affected	The number of subscriber Profile rows created. A value of "1" is expected for success.	

Error Codes

Table 18: Error Codes: Create Profile

Error Code	Description
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
INVALID _SOAP_XML	Invalid SOAP XML
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_EXISTS	Key Already Exists. A subscriber/pool already exists with the given key
MULT_VER_TAGS_FOUND	Multiple Version Tags Found
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
INVALID_XML	Invalid Input XML

Examples

Request #1

A subscriber is created, with an AccountId, MSISDN and IMSI keys. The BillingDay and Entitlement fields are set. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber" />
  <set>
    <expr><attr name="AccountId"/><value val="10404723525"/></expr>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
```

```
    <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
  </set>
</req>
```

Response #1

The request is successful, and the subscriber was created.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A subscriber is created, with an AccountId, MSISDN and IMSI keys. Another subscriber already exists with the given IMSI.

```
<req name="insert" resonly="y">
  <ent name="Subscriber" />
  <set>
   <expr><attr name="AccountId"/><value val="10404723525"/></expr>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPassPlus"/></expr>
  </set>
</req>
```

Response #2

The request fails. The `error` value indicates a subscriber already exists with the given IMSI, and the `affected` rows are `0`.

```
<req name="insert" resonly="y">
  <res error="70020" affected="0"/>
</req>
```

Request #3

A subscriber is created, with an AccountId, MSISDN and IMSI keys. The BillingDay and Entitlement fields are set. The request is not required in the response. Provisioning has been disabled.

```
<req name="insert" resonly="y">
  <ent name="Subscriber" />
  <set>
    <expr><attr name="AccountId"/><value val="10404723525"/></expr>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
  </set>
</req>
```

Response #3

The request fails. The `error` value indicates that provisioning has been disabled.

```
<req name="insert" resonly="y">
  <res error="70031" affected="0/">
</req>
```

Request #4

A subscriber is created, with an AccountId, MSISDN and IMSI keys. The BillingDay and Entitlement fields are set. The Quota and State entities are also created. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber" />
  <set>
    <expr><attr name="AccountId"/><value val="178322212122"/></expr>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="IMSI"/><value val="302370123456789"/></expr>
    <expr><attr name="BillingDay"/><value val="6"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
    <expr><attr name="Quota"/><op value="="/>
      <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
                     <usage>
                       <version>3</version>
                       <quota name="Weekend">
                         <totalVolume>100</totalVolume>
                         <Type>quota</Type>
                         <QuotaState>active</QuotaState>
                         <nextResetTime>2014-01-10T02:00:00</nextResetTime>
                       </quota>
                       <quota name="Evenings">
                         <totalVolume>100</totalVolume>
                         <Type>quota</Type>
                         <QuotaState>active</QuotaState>
                       <nextResetTime>2014-02-01T00:00:00</nextResetTime>
                       </quota>
                     </usage>]]>
      </cdata>
    </expr>
    <expr><attr name="State"/><op value="="/>
      <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <state>
          <version>1</version>
          <property>
            <name>mcc</name>
            <value>302</value>
          </property>
        </state>]]>
      </cdata>
    </expr>
  </set>
</req>
```

Response #4

The request is successful, and the subscriber was created.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Get Profile

This operation retrieves all field-value pairs created for a subscriber that is identified by the *keyName* and *keyValue*.

A *keyName* and *keyValue* are required in the request in order to identify the subscriber. The response content includes only valid field-value pairs which have been previously provisioned or created by default.

Prerequisites

A subscriber with a key of the *keyName*/*keyValue* supplied must exist.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 19: Request Definitions: Get Profile

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	Transaction id value provided in request, and will be passed back in the response	1-4294967295
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
```

```
  <rset>
    <row>
      <rv>
          <![CDATA[cdataRowValue]]>
      </rv>
    </row>
  </rset>
]
</req>
```

Table 20: Response Definitions: Get Profile

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Profile* for other values.	
affected	The number of data rows returned. A value of "1" is expected for success	
cdataRowValue	Contents of the subscriber Profile XML data "blob"	

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful. Only a single `<row>` element is returned, with a single `<rv>` (row value) element containing an XML CDATA construct containing the requested pool profile data (i.e. XML blob).

Error Codes

Table 21: Error Codes: Get Profile

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to get Profile data for a subscriber. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the subscriber Profile data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
    <rset>
      <row>
        <rv>
          <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
            <subscriber>
              <field name="AccountId">10404723525</field>
              <field name="MSISDN">33123654862</field>
              <field name="IMSI">184569547984229</field>
              <field name="BillingDay">1</field>
              <field name="Tier"></field>
              <field name="Entitlement">WeekPass</field>
              <field name="Entitlement">DayPass</field>
            </subscriber>]]>
        </rv>
    </row>
  </rset>
</req>
```

Delete Profile

This operation deletes all profile data (field-value pairs) and opaque data for the subscriber identified by the *keyName* and *keyValue*.

Prerequisites

A subscriber with a key of the *keyName*/*keyValue* supplied must exist.

The subscriber must not be a member of a pool, or the request will fail.

Request

```
<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 22: Request Variable Definitions: Delete Profile

Variable Name	Definition	Values
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • *n* — include the original request in the response (default)
id	Transaction id value provided in request, and will be passed back in the response	1-4294967295
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	

Response

```
<req name="delete" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 23: Response Variable Definitions: Delete Profile

Variable	Definition	Values
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see	

Variable	Definition	Values
	Error Codes: Delete Profile for other values	
affected	The number of subscribers deleted. A value of "1" is expected for success.	

Error Codes

Table 24: Error Codes: Delete Profile

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found.
SUB_IN_POOL	Subscriber is Pool Member. The subscriber is a member of a pool. A subscriber cannot be deleted if they are a pool member.

Examples

Request #1

The subscriber with the given MSISDN is deleted. The subscriber exists. The request (by default) should be included in the response.

```
<req name="delete">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the original request is included in the response.

```
<req name="delete">
  <req name="delete">
    <ent name="Subscriber"/>
    <where>
      <expr><attr name="MSISDN"/><op value="="/>
            <value val="33123654862"/></expr>
    </where>
  </req>
  <res error="0" affected="1"/>
</req>
```

Request #2

The subscriber with the given MSISDN is deleted. The subscriber does NOT exist. The request should not be included in the response.

```
<req name="delete">
  <ent name="Subscriber" resonly="y"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123655555/></expr>
  </where>
</req>
```

Response #2

The request fails. The `error` value indicates a subscriber with the given MSISDN does not exist, and the `affected` rows are `0`. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70019" affected="0"/>
</req>
```

Subscriber Profile Field Commands

Table 25: Summary of Subscriber Profile Field Commands

Command	Description	Key(s)	Command Syntax
Add Field Value	Add a value to the specified field. This operation does not affect any pre-existing values for the field	MSISDN, IMSI, NAI or AccountId	`<req name="update"> <ent name="Subscriber"/> <oper name="AddToSet">`
Get Field	Retrieve the value(s) for the specified field		`<req name="select"> <ent name="Subscriber"/>`
Update Field	Update field(s) to the specified value(s)		`<req name="update"> <ent name="Subscriber"/>`
Delete Field	Delete all the values for the specified field(s)		`<req name="update"> <ent name="Subscriber"/> … <value val="" isnull="y"/>…`
Delete Field Value	Delete a value for the specified field		`<req name="update"> <ent name="Subscriber"/> <oper name="RemoveFromSet">`

Add Field Value

This operation adds one or more value(s) to the specified multi-value field for the subscriber identified by the *keyName* and *keyValue*.

This operation can only be executed for the fields defined as multi-value field in the Subscriber Entity Configuration. Any pre-existing values for the field are not effected.

All existing values are retained, and the new values(s) specified are inserted. For example, if the current value of a field was "a,b,c", and this command was used with value "d", after the update the field would have the value "a,b,c,d".

Note: If a value being added already exists, the request will fail.

Note: The *fieldValue* is case-sensitive. An attempt to add the value "a" to current field value of "a,b,c" would fail, but an attempt to add the value "A" would be successful and result in the field value being "a,b,c,A"

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The field *fieldName* must be a valid field in the subscriber Profile, and must be a multi-value field.

Each *fieldValueX* being added must NOT already be present in the field.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="fieldName"/><value val="fieldValue1"/></expr>
[
      <expr><attr name="fieldName"/><value val="fieldValue2"/></expr>
      :
      <expr><attr name="fieldName"/><value val="fieldValueX"/></expr>]
]
    </oper>
  </set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 26: Request Variable Definitions: Add Field Value

Variable	Definition	Values
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
fieldName	A user defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to *fieldName*	
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValueX	Corresponding key field value assigned to *keyName* **Note:** Each *keyValueX* must contain a single value for the	

Variable	Definition	Values
	multi-value field. To remove more than one value, include multiple `<expr>` elements for the field, each containing a single value.	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 27: Response Variable Definitions: Add Field Value

Variable	Definition	Values
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Add Field Value* for other values	
affected	The number of subscriber Profiles updated. A value of "1" is expected for success.	

Error Codes

Table 28: Error Codes: Add Field Value

Error Code	Description
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FIELD_NOT_UPDATABLE	Field Cannot be Updated. The field is defined in the SEC as not be updatable.
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
VALUE_EXISTS	List value already exists.
FLD_NOT_MULTI	Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value.

Examples

Request #1

A request is made to add the value *DayPass* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* value is not already present in the *Entitlement* field. The request is not required in the response. An *id* value is supplied, which is required in the response.

```
<req name="update" resonly="y" id="13579">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the value was added to the *Entitlement* field. The original request is not included. The *id* value was included.

```
<req name="update" resonly="y" id="13579">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to add the values *HighSpeed* and *Unlimited* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. Neither value is already present in the *Entitlement* field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="HighSpeed"/></expr>
      <expr><attr name="Entitlement"/><value val="Unlimited"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the values were added to the *Entitlement* field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #3

A request is made to add the value *DayPass* to the Entitlement field. The *Entitlement* field is a valid multi-value field. The *DayPass* value is already present in the *Entitlement* field. The request is not required in the response. An *id* value is supplied, which is required in the response.

```
<req name="update" resonly="y" id="13579">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #3

The request fails. The `error` value indicates the given value is already present, and the `affected` rows are `0`. The original request is not included. The `id` value was included.

```
<req name="update" resonly="y" id="13579">
  <res error="70033" affected="0"/>
</req>
```

Request #4

A request is made to add the value *Gold* to the *Tier* field. The *Tier* field is not a valid multi-value field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Tier"/><value val="Gold"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #4

The request fails. The `error` value indicates the field is not a multi-value field, and the `affected` rows are `0`. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70034" affected="0"/>
</req>
```

Get Field

This operation retrieves the values for the specified field(s) for the subscriber identified by the specified *keyName* and *keyValue*.

Note: An entire entity for the subscriber can be retrieved by specifying an *opaqueDataType* corresponding to the interface entity name in the SEC. The *opaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail

Prerequisites

A subscriber with a key of the *keyName/keyValue* supplied must exist.

Each requested field *fieldNameX* must be a valid field in the subscriber Profile.

Each requested *opaqueDataTypeX* must reference a valid Entity in the Interface Entity Map table in the SEC.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="fieldName1"/></expr>
[
    <expr><attr name="fieldName2"/></expr>
    :
    <expr><attr name="fieldNameN"/></expr>
]
[
    <expr><attr name="opaqueDataType1"/></expr>
    <expr><attr name="opaqueDataType2"/></expr>
    :
```

```
    <expr><attr name="opaqueDataTypeN"/></expr>
]

  </select>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 29: Request Variable Definitions: Get Field

Variable	Definition	Values
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	Transaction id value provided in request, and will be passed back in the response	1-4294967295
fieldNameX	A user defined field within the subscriber Profile	
opaqueDataTypeX	A user defined field within the subscriber Profile, that represents a transparent or opaque data entity	• Quota • State • DynamicQuota
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyNameX*	

Note: At least one *fieldNameX*/*opaqueDataTypeX* field must be requested.

Note: The order in which *fieldNameX*/*opaqueDataTypeX* are specified in the request is not important.

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
[
```

```
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv>
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv>
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv>
]
[
      <rv>cdataRowValue1</rv> | <rv null="y">
      <rv>cdataRowValue2</rv> | <rv null="y">
      :
      <rv>cdataRowValueN</rv> | <rv null="y">
]
    </row>
  </rset>
]
</req>
```

Table 30: Response Variable Definitions: Get Field

Variable	Definition	Values
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Field* for other values.	
affected	The number of subscriber Profiles from which data is returned. A value of "1" is expected for success.	
rowValueX	The value of the requested field (for normal fields, not for opaque/transparent entities). **Note:** For multi-value fields, the value will contain a comma separated list of values on a single line. E.g. "a,b,c".	

Variable	Definition	Values
cdataRowValueX	Contents of the XML data "blob" (for requested fields that are opaque/transparent entities)	

Note: The <rset> (row set) element is optional. It is only present if the request was successful. Only a single <row> element is returned. One <rv> (row value) element exists for every *fieldNameX* or *opaqueDataTypeX* supplied in the original request. The <rv> elements are ordered the same as the *fieldNameX / opaqueDataTypeX* fields were specified in the original request. If the field is valid, but not present in the entity, this is indicated with <rv null="y">. If the field is present, but has an empty value, this is indicated with <rv></rv>.

Error Codes

Table 31: Error Codes: Get Field

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC.
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to get the *MSISDN*, *Entitlement*, *Tier*, and *BillingDay* fields. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="MSISDN"/></expr>
    <expr><attr name="Entitlement"/></expr>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="BillingDay"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the 4 requested values are returned (the *Entitlement* is a multi-value field). The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>33123654862</rv>
      <rv>DayPass,WeekPass,Weekend</rv>
```

```
      <rv>InfinityPrepaid</rv>
      <rv>23</rv>
    </row>
  </rset>
</req>
```

Request #2

A request is made to get the *IMSI*, *Entitlement*, *Tier*, and *Custom20* fields. The *Entitlement* and *Tier* fields are set in the XML blob, the *IMSI* field is not set, and the *Custom20* field is set, but has an empty value. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="IMSI"/></expr>
    <expr><attr name="Entitlement"/></expr>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="Custom20"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the 4 requested values are returned (the *Entitlement* is a multi-value field). The *IMSI* field is indicated as unset, and the *Custom20* field is indicated as empty. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
      <rv>1,14,2,8</rv>
      <rv>InfinityPrepaid</rv>
      <rv></rv>
    </row>
  </rset>
</req>
```

Request #3

A request is made to get the *Tier*, *Rating*, and *BillingDay* fields. The *Rating* field is not a valid field in a subscriber Profile. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="Rating"/></expr>
    <expr><attr name="BillingDay"/></expr>
  </select>
  <where>
    <expr><attr name="NAI"/><op value="="/>
          <value val="john.smith@operator.com"/></expr>
  </where>
</req>
```

Response #3

The request fails. The `error` value indicates that the *Rating* field is undefined, and the `affected` rows are `0`. The original request is not included.

```
<req name="select" resonly="y">
  <res error="70015" affected="0"/>
</req>
```

Request #4

A request is made to get the *MSISDN*, and *BillingDay* fields, as well as the *Quota* and *State* entity data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="MSISDN"/></expr>
    <expr><attr name="BillingDay"/></expr>
    <expr><attr name="Quota"/></expr>
    <expr><attr name="State"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #4

The request is successful, and the 4 requested values are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>33123654862</rv>
      <rv>23</rv>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="AggregateLimit">
              <cid>9223372036854775807</cid>
              <time>3422</time>
              <totalVolume>1000</totalVolume>
              <inputVolume>980</inputVolume>
              <outputVolume>20</outputVolume>
              <serviceSpecific>12</serviceSpecific>
              <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
            </quota>
          </usage>>
        </rv>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <state>
            <version>1</version>
            <property>
              <name>mcc</name>
              <value>315</value>
            </property>
            <property>
```

```
              <name>expire</name>
              <value>2010-02-09T11:20:32</value>
            </property>
            <property>
              <name>approved</name>
              <value>yes</value>
            </property>
          </state>]]>
        </rv>
    </row>
  </rset>
</req>
```

Request #5

A request is made to get the *MSISDN* field, as well as the *DynamicQuota* and *State* entity data. The subscriber does not have any DynamicQuota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="MSISDN"/></expr>
    <expr><attr name="DynamicQuota"/></expr>
    <expr><attr name="State"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #5

The request is successful, and the 3 requested values are returned. The DynamicQuota is indicated as being not set. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>33123654862</rv>
      <rv null="y"/>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <state>
            <version>1</version>
            <property>
              <name>mcc</name>
              <value>315</value>
            </property>
            <property>
              <name>expire</name>
              <value>2010-02-09T11:20:32</value>
            </property>
            <property>
              <name>approved</name>
              <value>yes</value>
            </property>
          </state>]]>
        </rv>
    </row>
```

```
  </rset>
</req>
```

Update Field

This operation updates a field(s) to the specified values for the subscriber identified by the specified key name and key value. This operation replaces ("sets") the value(s) of the field(s), which means that any existing value(s) for the field(s) are deleted first.

For multi-value fields, all existing values are removed, and only the new values(s) specified are inserted. Adding values to a current set is accomplished using Add Field Value. For example, if the current value of a field was "a,b,c", and this command was used with value "d", after the update the field would have the value "d" (it would NOT be "a,b,c,d").

All fields are updated at once in the DB. All fields and all values must be valid for the update to be successful. I.e., as soon as one error is detected during processing, the request is abandoned (and an error returned). For example, if the third specified field fails validation, then none of the fields are updated.

Note: If the requested field(s) are valid, but not currently present, they will be created.

Note: An entire entity for the subscriber can be replaced by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the entire XML blob value in *cdataFieldValue*. The *cdataFieldName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: Multi-value fields can be specified by a single *fieldNameX* value with a delimited list of values, or multiple *fieldNameX* fields, each containing a single value.

Note: A request to update a field(s) cannot be mixed with a request to delete a field, otherwise the request will fail. A request to update a field(s) must only contain field(s) being updated.

Note: A request to update a key field(s) (such as IMSI/MSISDN/NAI/AccountId) cannot be mixed with a request that updates non-key field(s). In this case, separate requests must be sent: one containing key field(s), and another containing non-key fields (ideally, within a transaction).

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

Each requested *fieldName* must be a valid field in the subscriber Profile.

Each requested *cdataFieldName* must be a valid non pooled transparent/opaque interface entity name for a subscriber.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
<ent name="Subscriber"/>
  <set>
    <expr>
<
      <attr name="fieldName1"/><value val="fieldValue1"/></expr>
|
      <attr name="cdataFieldName1"/><op value="="/>
        <cdata><![CDATA[cdataFieldValue1]]></cdata>
```

```
>
    </expr>
[
    <expr>
<
      <attr name="fieldName2"/><value val="fieldValue2"/>
|
      <attr name="cdataFieldName2"/><op value="="/>
        <cdata><![CDATA[cdataFieldValue2]]></cdata>
>
    </expr>
    :
    <expr>
<
      <attr name="fieldNameN"/><value val="fieldValueN"/>
|
      <attr name="cdataFieldNameN"/><op value="="/>
        <cdata><![CDATA[cdataFieldValueN]]></cdata>
>
    </expr>
]
</set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 32: Request Variable Definitions: Update Field

Variable	Definition	Values
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
fieldNameX	A user defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to *fieldNameX* **Note:** For multi-value fields, the value can contain a comma separated list of values on a single line. E.g. "a,b,c"	
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId

Variable	Definition	Values
keyValue	Corresponding key field value assigned to *keyName*	
cdataFieldNameX	A user defined field within the subscriber Profile, that represents a transparent or opaque data entity, as per the the defined interface entity name in the SEC	• Quota • State • DynamicQuota
cdataFieldValueX	Contents of the XML data "blob" for *cdataFieldNameX*	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 33: Response Variable Definitions: Update Field

Variable	Definition	Values
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Update Field* for other values.	
affected	The number of subscriber Profiles updated. A value of "1" is expected for success.	

Error Codes

Table 34: Error Codes: Update Field

Error Code	Description
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FIELD_NOT_UPDATABLE	Field Cannot be Updated. The field is defined in the SEC as not be updatable
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INVALID_XML	Invalid Input XML

Examples

Request #1

A request is made to update the value of the *BillingDay* field to *23*, and the *Tier* field to *Gold*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><value val="23"/></expr>
    <expr><attr name="Tier"/><value val="Gold"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="305801234567890"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the *BillingDay* value was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to update the value of the *BillingDay* field to *55*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><value val="55"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="305801234567890"/></expr>
  </where>
</req>
```

Response #2

The request fails. The `error` value indicates the value of *BillingDay* was invalid, and the `affected` rows are `0`. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70006" affected="0"/>
</req>
```

Request #3

A request is made to update the value of the *BillingDay* field to *23* and the entire *State* entity. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><value val="23"/></expr>
    <expr><attr name="State"/><op value="="/>
      <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <state>
          <version>1</version>
          <property>
            <name>mcc</name>
            <value>302</value>
          </property>
          <property>
            <name>expire</name>
            <value>2014-02-09T11:20:32</value>
          </property>
        </state>]]>
      </cdata>
    </expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="305801234567890"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the *BillingDay* and *State* values were updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #4

A request is made to update the value of the *Entitlement* field using a single field with multiple values. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Entitlement"/><value val="Weekend,Evening"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="305801234567890"/></expr>
  </where>
</req>
```

Response #4

The request is successful, and the *Entitlement* value was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #5

A request is made to update the value of the *Entitlement* field using multiple fields, each containing a single value. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Entitlement"/><value val="Weekend"/></expr>
    <expr><attr name="Entitlement"/><value val="Evening"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="305801234567890"/></expr>
  </where>
</req>
```

Response #5

The request is successful, and the *Entitlement* value was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #6

A request is made to update the value of the *MSISDN* and *BillingDay* fields. The request is not required in the response

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="BillingDay"/><value val="55"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="305801234567890"/></expr>
  </where>
</req>
```

Response #6

The request fails. The *error* value indicates that the request is not valid because it contained key and non-key fields, and the *affected* rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70030" affected="0"/>
</req>
```

Delete Field

This operation deletes the specified fields for the subscriber identified by *keyName* and *keyValue* in the request.

If the field is multi-value field then all values are deleted. Deletion of a field results removal of the entire field from the subscriber Profile. I.e. the field is not present, not just the value is empty.

Note: The field being deleted does NOT need to have a current value. It can be empty (i.e. deleted) already, and the request will succeed.

Note: A request to delete a field(s) cannot be mixed with a request to update a field, otherwise the request will fail. A request to delete a field(s) must only contain field(s) being deleted.

Prerequisites

A subscriber with the key of the *keyName*/*keyValue* supplied must exist.

Each requested field *fieldName* must be a valid field in the subscriber Profile.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="fieldName1"/><op value="="/>
          <value val="" isnull="y"/></expr>
[
    <expr><attr name="fieldName2"/><op value="="/>
          <value val="" isnull="y"/></expr>
    :
    <expr><attr name="fieldNameN"/><op value="="/>
          <value val="" isnull="y"/></expr>
]
```

```
  </set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 35: Request Variable Definitions: Delete Field

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	
fieldNameX	A user defined field within the subscriber Profile	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 36: Response Variable Definitions: Delete Field

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters

Variable	Definition	Value
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The id value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Field* for other values.	
affected	The number of subscriber Profiles updated. A value of "1" is expected for success.	

Error Codes

Table 37: Error Codes: Delete Field

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to delete the *BillingDay* and *Tier* fields. Both fields are valid subscriber Profile fields. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><op value="="/>
          <value val="" isnull="y"/></expr>
    <expr><attr name="Tier"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the two fields were deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to delete the *Message* field. *Message* is not a valid subscriber Profile field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Message"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #2

The request fails. The `error` value indicates the given field was not found, and the `affected` rows are `0`. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70015" affected="0"/>
</req>
```

Delete Field Value

This operation deletes one or more value(s) from the specified field for the subscriber identified by the *keyName* and *keyValue* in the request.

This operation can only be executed for the fields defined as multi-value field in the Subscriber Entity Configuration.

Each individual value is removed from the subscriber Profile. If a supplied value does not exist, then it is ignored. For example, if a profile contains values "A,B,C" and a request to delete "A,B" is made, this succeeds and the profile is left with "C" as the value. If the profile contains "A,B,C" and a request is made to delete "C,D" the request succeeds and the profile is left with "A,B" as the value.

If all values are removed, the entire field is removed from the subscriber Profile (i.e. there is no XML element present).

Note: The *fieldValue* is case-sensitive. An attempt to remove the value "A" from a current field value of "A,B,C" would be successful, but an attempt to remove the value "a" would fail.

Note: A request to delete a field(s) cannot be mixed with a request to update a field, otherwise the request will fail. A request to delete a field(s) must only contain field(s) being deleted.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The field *fieldName* must be a valid field in the subscriber Profile, and must be a multi-value field.

Each *fieldValueX* being added must be present in the field.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="fieldName"/><value val="fieldValue1"/></expr>
[
      <expr><attr name="fieldName"/><value val="fieldValue2"/></expr>
      :
      <expr><attr name="fieldName"/><value val="fieldValueX"/></expr>]
]
  </set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 38: Request Variable Definitions: Delete Field Value

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	
fieldName	A user defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to *fieldName*	

Note: Each *keyValueX* must contain a single value for the multi-value field. To remove more than one value, include multiple `<expr>` elements for the field, each containing a single value.

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 39: Response Variable Definitions: Delete Field Value

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The id value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Field Value* for other values.	
affected	The number of subscriber Profiles updated. A value of "1" is expected for success.	

Error Codes

Table 40: Error Codes: Delete Field Value

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
FLD_NOT_MULTI	Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value.

Examples

Request #1

A request is made to remove the value *DayPass* from the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* value is present in the *Entitlement* field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the value was removed from the *Entitlement* field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to remove the values *WeekendPass* and *Unlimited* from the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *WeekendPass* value is present in the *Entitlement* field, but the *Unlimited* value is not. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="Entitlement"/><value val="WeekendPass"/></expr>
      <expr><attr name="Entitlement"/><value val="Unlimited"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the *WeekendPass* value was removed from the *Entitlement* field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Subscriber Opaque Data Commands

Table 41: Summary of Subscriber Opaque Data Commands

Command	Description	Key(s)	Command Syntax
Create Opaque Data	Create opaque data of the specified type	MSISDN, IMSI, NAI or AccountId	`<req name="insert"> <ent name="Subscriber"/>`
Get Opaque Data	Retrieve opaque data of the specified type		`<req name="select"> <ent name="Subscriber"/>`
Update Opaque Data	Update opaque data of the specified type		`<req name="update"> <ent name="Subscriber"/>`
Delete Opaque Data	Delete opaque data of the specified type		`<req name="update"> <ent name="Subscriber"/> … <value val="" isnull="y"/>…`

Create Opaque Data

This operation creates the opaque data of the specified *opaqueDataType* for the subscriber identified by the *keyName* and *keyValue* in the request.

The opaque data is provided in the request within a `<cdata>` construct.

Note: The opaque data for creating an entity/row is provided in the request within a CDATA construct.

Note: The *opaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *opaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.

No opaque data of the *opaqueDataType* must already exist for the subscriber (unless the odk attribute is specified).

Request

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="Subscriber"/>
```

```
  <set>
    <expr><attr name="opaqueDataType"/><op value="="/><cdata>
<![CDATA[
cdataFieldValue
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 42: Request Variable Definitions: Create Opaque Data

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
odk	(Optional) Indicates that the insert request should be converted to an update if the opaque data type specified already exists	
opaqueDataType	A user defined type/name for the opaque data	• Quota • State • DynamicQuota
cdataFieldValue	Contents of the XML data "blob"	
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	

Response

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 43: Response Variable Definitions: Create Opaque Data

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Create Opaque Data* for other values.	
affected	The number of opaque data rows inserted/updated for the subscriber. A value of "1" is expected for success.	

Error Codes

Table 44: Error Codes: Create Opaque Data

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
MULT_VER_TAGS_FOUND	Multiple Version Tags Found
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INVALID _XML	Invalid Input XML
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Error Code	Description
REG_EXISTS	Register Already Exists

Examples

Request #1

A request is made to create the *Quota* opaque data. The Quota XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
   <expr><attr name="Quota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the Quota opaque data was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to create the *State* opaque data. The State XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="State"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
```

```
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the State opaque data was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #3

A request is made to create the *DynamicQuota* opaque data. The Quota XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="DynamicQuota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="AggregateLimit">
    <Type>Roll-Over</Type>
    <InstanceId>15678</InstanceId>
    <Priority>4</Priority>
    <InitialTime>135</InitialTime >
    <InitialTotalVolume>2000</InitialTotalVolume>
    <InitialInputVolume>1500</InitialInputVolume>
    <InitialOutputVolume>500</InitialOutputVolume>
    <InitialServiceSpecific>4</InitialServiceSpecific>
    <activationdatetime>32</activationdatetime>
    <expirationdatetime>28</expirationdatetime>
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
</definition>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the DynamicQuota opaque data was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #4

A request is made to create the *Location* opaque data. The Location XML blob is supplied whole. *Location* is NOT a valid opaque data type. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Location"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<location>
  <town>Montreal</town>
  <province>Quebec</province>
  <country>Canada</country>
</location>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #4

The request fails. The `error` value indicates the opaque data type is invalid, and the `affected` rows are `0`. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="70015" affected="0"/>
</req>
```

Get Opaque Data

This operation retrieves the opaque data of the specified *opaqueDataType* for the subscriber identified by the *keyName* and *keyValue* in the request.

The response contains the entire XML blob for the requested opaque data.

Note: The *opaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *opaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.

The opaque data of the *opaqueDataType* must exist for the subscriber.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="opaqueDataType"/></expr>
  </select>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 45: Request Variable Definitions: Get Opaque Data

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
opaqueDataType	A user defined type/name for the opaque data	• Quota • State • DynamicQuota
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>
          <![CDATA[cdataRowValue]]>
      </rv>
    </row>
  </rset>
]
</req>
```

Table 46: Response Variable Definition: Get Opaque Data

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The id value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Opaque Data* for other values.	
affected	The number of subscriber opaque data rows returned. A value of "1" is expected for success.	
cdataRowValue	Contents of the XML data "blob"	

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful. Only a single `<row>` element is returned, with a single `<rv>` (row value) element containing an XML CDATA construct containing the requested pool opaque data (i.e. XML blob).

Error Codes

Table 47: Error Codes: Get Opaque Data

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC

Examples

Request #1

A request is made to get the *Quota* opaque data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="Quota"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the Quota opaque data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
    <rset>
      <row>
        <rv>
          <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
            <usage>
              <version>3</version>
              <quota name="AggregateLimit">
                <cid>9223372036854775807</cid>
                <time>3422</time>
                <totalVolume>1000</totalVolume>
                <inputVolume>980</inputVolume>
                <outputVolume>20</outputVolume>
                <serviceSpecific>12</serviceSpecific>
                <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
              </quota>
            </usage>]]>
        </rv>
    </row>
  </rset>
</req>
```

Request #2

A request is made to get the *State* opaque data. *State* is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="State"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #2

The request fails. The `error` value indicates the opaque data type is not found, and the `affected` rows are `0`.The original request is not included.

```
<req name="select" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

Request #3

A request is made to get the *Location* opaque data. *Location* is NOT a valid opaque data type. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="Location"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #3

The request fails. The `error` value indicates the opaque data type is invalid, and the `affected` rows are `0`. The original request is not included.

```
<req name="select" resonly="y">
  <res error="70015" affected="0"/>
</req>
```

Update Opaque Data

This operation updates the opaque data of the specified *opaqueDataType* for the subscriber identified by the *keyName* and *keyValue* in the request.

The opaque data is provided in the request within a `<cdata>` construct. The existing opaque data is completely replaced by the data supplied in the request.

Note: The opaque data provided in an XML blob is always checked to be valid XML. If the entity is defined as transparent in the SEC, then the XML blob is fully validated against the definition in the SEC. If either validation check fails, then the request is rejected.

Note: The *opaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *opaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
```

```
  <set>
    <expr><attr name="opaqueDataType"/><op value="="/><cdata>
<![CDATA[
cdataFieldValue
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 48: Response Variable Definition: Update Opaque Data

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
opaqueDataType	A user defined type/name for the opaque data	• Quota • State • DynamicQuota
cdataFieldValue	Contents of the XML data "blob"	
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 49: Response Variable Definitions: Update Opaque Data

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Update Opaque Data* for other values.	
affected	The number of subscriber opaque data rows updated. A value of "1" is expected for success.	

Error Codes

Table 50: Error Codes: Update Opaque Data

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element.
INVALID _XML	Invalid Input XML
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to update the *State* opaque data. The State XML blob is supplied whole. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="State"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>3</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the State opaque data is updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to update the *Quota* opaque data. *Opaque* is a valid opaque data type, but the subscriber does not have this opaque data type. The Quota XML blob is supplied whole. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
   <expr><attr name="Quota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>92233720368547758O7</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
```

```
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the Quota opaque data was created. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="0"/>
</req>
```

Delete Opaque Data

This operation deletes the opaque data of the specified *opaqueDataType* for the subscriber identified by the *keyName* and *keyValue* in the request.

Only one opaque data type can be deleted per request.

Note: The deletion of a non-existent opaque data type (but that is defined in the SEC) is not considered as an error.

Note: The *opaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *opaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="opaqueDataType"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
</req>
```

Table 51: Request Variable Definitions: Delete Opaque Data

Variable	Definition	Value
keyName	A user defined field identified as key for the subscriber	
keyValue	A key value identifying the subscriber	
opaqueDataType	A user defined type/name for the opaque data **Note:** The data is deleted by setting an empty field value, and also specifying the attribute `isnull="y"`.	• Quota • State • DynamicQuota

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 52: Response Variable Definitions: Delete Opaque Data

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Opaque Data* for other values.	

Variable	Definition	Value
affected	The number of subscriber opaque data entries deleted. A value of "1" is expected for success.	

Error Codes

Table 53: Error Codes: Delete Opaque Data

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to delete the *State* opaque data. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="State"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the State opaque data is deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to delete the *Quota* opaque data. *Quota* is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Quota"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
```

```
  </where>
</req>
```

Response #2

The request is successful, because no error is returned if the subscriber does not have the opaque data type.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Subscriber Data Row Commands

A transparent data entity may contain data that is organized in "rows". An example of a row is a specific quota within the Quota entity.

The row commands allow operations (create/retrieve/update/delete) at the row level. The required row is identified in the request by the *RowIdName/RowIdValue*.

Table 54: Summary of Subscriber Data Row Commands

Command	Description	Key(s)	Command Syntax
Create Row	Insert data row into transparent data of the specified type.	MSISDN, IMSI, NAI or AccountId	`<req name="insert"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`
Get Row	Retrieve data row from transparent data of the specified type.		`<req name="select"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`
Delete Row	Delete data row within transparent data of the specified type.		`<req name="delete"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`

Create Row

This operation creates a data row for the subscriber identified by the *keyName* and *keyValue*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. All *fieldNameX* fields specified are set within the row.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for Quota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row already existed called "DayPass", then an attempt to update an existing row called "DAYPASS" would be successful, and two rows called "DayPass" and "DAYPASS" would be present.

Note: If the transparent entity specified in *entityName* does not exist for the subscriber, it will be created.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

Request

Note: This command allows 2 different formats. One with the *keyName/keyValue* within the `<set>` element, and another with the *keyName/keyValue* within a `<select>` element.

Format #1

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="keyName"/><value val="keyValue"/></expr>
    <expr><attr name="rowIdName"/><value val="rowIdValue"/></expr>
[
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]
  </set>
</req>
```

Format #2

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="rowIdName"/><value val="rowIdValue"/></expr>
[
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
  </where>
  </set>
</req>
```

Table 55: Request Variable Definition: Create Row

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295

Variable	Definition	Value
odk	(Optional) Indicates that the insert request should be converted to an update if the data row for the specified entry already exists	
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the Quota transparent data
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyNameX*	
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
fieldNameX	A user defined field within the data row	
fieldValueX	Corresponding field value assigned to *fieldNameX* **Note:** For multi-value fields, the value can contain a comma separated list of values on a single line. E.g. "a,b,c".	

Note: Rows that have the same *rowIdName/rowIdValue* are permitted. Where duplicate rows occur, and an additional field is set to define uniqueness (such as `<cid>` in the Quota entity) no validation is performed by UDR to ensure uniqueness. Unique values must be supplied by the provisioning client otherwise operations (such as updating an existing row) may fail if more than one matching row is found.

Note: If the `odk="yes"` attribute is set (implying that an update be made if the row exists), then if **multiple** rows exist for the specified *rowIdName/rowIdValue*, then the request will fail because it is not known which of the multiple rows to update.

Response

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
```

```
  <res error="error" affected="affected"/>
</req>
```

Table 56: Response Variable Definition: Create Row

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Create Row* for other values.	
affected	The number of data entities updated. A value of "1" is expected for success.	

Error Codes

Table 57: Error Codes: Create Row

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INVALID_SOAP_XML	Invalid SOAP XML
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
MULTIPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria. **Note:** Only returned when the `odk`="yes" attribute is supplied, and duplicate candidate rows to update are found

Examples

Request #1

A request is made to create a data row in the *QuotaEntity* (Quota) data. The data row identifier field is *name*, and the value is *Q1*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="name"/><value val="Q1"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
          <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>
```

Response #1

The request is successful, and the data row was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to create a data row in the *QuotaEntity* (Quota) data, using the alternate request format. The data row identifier field is *name*, and the value is *Q2*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="name"/><value val="Q2"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
```

```
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
          <value val="1961-12-15T09:04:03"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/><value val="15141234567"/></expr>

  </where>
</req>
```

Response #2

The request is successful, and the data row was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #3

A request is made to create a data row in the *QuotaEntity* (Quota) data. Quota is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="name"/><value val="Q1"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/><value val="1961-12-15T09:04:03"/></expr>

  </set>
</req>
```

Response #3

The request is successful, and the data row as well as the Quota entity are created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #4

A request is made to create a data row in the *QuotaEntity* (Quota) data. The data row identifier field is *name*, and the value is *Q2*. The "odk" attribute is included requesting the data row be updated if it already exists. A **single** row with the *name* of *Q2* exists in the Quota data. The request is not required in the response.

```
<req name="insert" resonly="y" odk="yes">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
```

```
    <expr><attr name="name"/><value val="Q2"/></expr>
    <expr><attr name="cid"/><value val="9223372036854888888"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/><value val="1961-12-15T09:04:03"/></expr>

  </set>
</req>
```

Response #4

The request is successful, and the existing *Q2* data row was updated. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #5

A request is made to create a data row in the *QuotaEntity* (Quota) data. The data row identifier field is *name*, and the value is *Q3*. The "odk" attribute is included requesting the data row be updated if it already exists. **Two** rows with the *name* of *Q3* already exist in the Quota data. The request is not required in the response.

```
<req name="insert" resonly="y" odk="yes">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="name"/><value val="Q3"/></expr>
    <expr><attr name="cid"/><value val="9223372036854777777"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/><value val="1961-12-15T09:04:03"/></expr>

  </set>
</req>
```

Response #5

The request fails. The error value indicates that multiple existing rows are found (i.e., more than one row has a *name* of *Q3*, and hence it is not possible to know which of the two rows to update), and the affected rows are 0. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

Get Row

This operation retrieves a data row(s) for the subscriber identified by the *keyName* and *keyValue*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName/instanceFieldValue*.

All data rows that match the requested *rowName/rowValue* and *instanceFieldName/instanceFieldValue* (if specified) are returned.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for Quota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row already existed called "DayPass", then an attempt to retrieve a row called "DayPass" would successful, but an attempt to retrieve a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to retrieve a row with a field with the value "Data" would be successful, but an attempt to retrieve a row with a field with the value "DATA" would fail.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

The transparent entity must exist for the subscriber.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 58: Request Variable Definitions: Get Row

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)

Variable	Definition	Value
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the Quota transparent data
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyNameX*	
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the Quota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
<
      <rv>
          <![CDATA[cdataRowValue1]]>
      </rv>
|
      <rv null="y"/>
>
    </row>
[
    <row>
      <rv>
          <![CDATA[cdataRowValue2]]>
      </rv>
    </row>
    :
    <row>
```

```
      <rv>
          <![CDATA[cdataRowValueN]]>
      </rv>
    </row>
]
  </rset>
]
</req>
```

Table 59: Response Variable Definitions: Get Row

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Row* for other values.	
affected	The number of subscriber data rows returned. A value of "1" is expected for success.	
cdataRowValueN	Contents of the XML data "blob" containing one requested/matching data row	

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful. One `<row>` element is returned per matching row, with a single `<rv>` (row value) element containing an XML CDATA construct containing a single requested data row instance.

Note: If the transparent entity exists, but the row value was not found, then the <*rv*> (row value) will indicate the row does not exist by containing the value <*rv null="y"*/>.

Error Codes

Table 60: Error Codes: Get Row

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found

Examples

Request #1

A request is made to get the *Q1* data row from the Quota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the Quota data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Q1">
            <cid>9223372036854775807</cid>
            <time>3422</time>
            <totalVolume>1000</totalVolume>
            <inputVolume>980</inputVolume>
            <outputVolume>20</outputVolume>
            <serviceSpecific>12</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

Request #2

A request is made to get the *Weekend* data row from the Quota data. The Quota data contains two rows called *Weekend*. One with `<cid>` of *11223344*, the other with a `<cid>` of *99887766*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and 2 Quota data rows are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Weekend">
            <cid>11223344</cid>
            <time>3422</time>
            <totalVolume>1000</totalVolume>
            <inputVolume>980</inputVolume>
            <outputVolume>20</outputVolume>
            <serviceSpecific>12</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Weekend">
            <cid>99887766</cid>
            <time>1232</time>
            <totalVolume>2000</totalVolume>
            <inputVolume>440</inputVolume>
            <outputVolume>8220</outputVolume>
            <serviceSpecific>99</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

Request #3

A request is made to get the *Weekend* data row from the Quota data, with the <cid> value of *11223344*. The Quota data contains two rows called *Weekend*. One with <cid> of *11223344*, the other with a <cid> of *99887766*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the Quota data with a <cid> of *11223344* is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Weekend">
            <cid>11223344</cid>
            <time>3422</time>
            <totalVolume>1000</totalVolume>
            <inputVolume>980</inputVolume>
            <outputVolume>20</outputVolume>
            <serviceSpecific>12</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

Request #4

A request is made to get the *LateNight* data row from the Quota data, with the <cid> value of *11223344*. The Quota data contains five rows called LateNight. Two with <cid> of *11223344*, one with a <cid> of *99887766*, and one with a <cid> of *55556666*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="LateNight"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #4

The request is successful, and the 2 Quota data rows with a <cid> of *11223344* are returned. The original request is not included.

```
<req name="select">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="LateNight">
            <cid>11223344</cid>
            <time>3422</time>
            <totalVolume>1000</totalVolume>
            <inputVolume>980</inputVolume>
            <outputVolume>20</outputVolume>
            <serviceSpecific>12</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="LateNight">
            <cid>11223344</cid>
            <time>1232</time>
            <totalVolume>2000</totalVolume>
            <inputVolume>440</inputVolume>
            <outputVolume>8220</outputVolume>
            <serviceSpecific>99</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

Request #5

A request is made to get the *Weekday* data row in the Quota data. The *Weekday* data row does NOT exist in the Quota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>
```

Response #5

The request is successful, and indicates that the requested row does not exist. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

Request #6

A request is made to get the *Weekday* data row in the Quota data. *Quota* is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>
```

Response #6

The request fails. The *error* value indicates the opaque data type is not found, and the *affected* rows are 0. The original request is not included.

```
<req name="select" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

Delete Row

This operation deletes a data row for the subscriber identified by the *keyName* and *keyValue*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*.

If more than one row matches the requested *rowName*/*rowValue* and *instanceFieldName*/*instanceFieldValue* (if specified), then all matching rows are deleted.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for Quota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row existed called "DayPass", then an attempt to delete a row called "DayPass" would be successful, but an attempt to delete a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to delete a row with a field with the value "Data" would be successful, but an attempt to delete a row with a field with the value "DATA" would fail.

Note: The deletion of a non-existent data row is not considered as an error.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

The transparent entity must exist for the subscriber.

Request

```
<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 61: Request Variable Definitions: Delete Row

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the Quota transparent data
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyNameX*	

Variable	Definition	Value
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the Quota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="delete" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 62: Response Variable Definitions: Delete Row

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Row* for other values.	
affected	The number of subscriber data rows deleted. A value of "1" indicates the row(s) existed and	

Variable	Definition	Value
	were deleted. A value of "0" indicates the row did not exist.	

Error Codes

Table 63: Error Codes: Delete Row

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found

Examples

Request #1

A request is made to delete the *Q1* data row from the Quota data. The *Q1* data row exists in the Quota data, and is there is only one row called *Q1*. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and data row in the Quota data is deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to update the *Weekend* data row in the Quota data. The *Weekend* data row does NOT exist in the Quota data. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

Response #2

The request is successful, because no error is returned if the data row is not present. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #3

A request is made to delete the *Q3* data row in the Quota data. The Quota data contains two rows called *Q3*. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the data row in the Quota data was deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #4

A request is made to delete the *Q4* data row in the Quota data with the `<cid>` value of *11223344*. The Quota data contains two rows called *Q4*. One with `<cid>` *11223344* the other with `<cid>` of *99887766*. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q4"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #4

The request is successful, and the data row in the Quota data is deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #5

A request is made to delete the *Bonus* data row in the Quota data. *QuotaEntity* is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Bonus"/></expr>
  </where>
</req>
```

Response #5

The request fails. The *error* value indicates the opaque data type is not found, and the *affected* rows are 0. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

Subscriber Data Row Field Commands

A transparent data entity may contain data that is organized in "rows". An example of a row is a specific quota within the Quota entity.

The row/field commands allow operations (retrieve/update/delete) at the row/field level. The required row is identified in the request by the *rowIdName/rowIdValue*, and the field is identified by the *fieldName*.

Table 64: Summary of Subscriber Data Row Field Commands

Command	Description	Key(s)	Command Syntax
Get Row Field	Retrieve value(s) for the specified field(s).	(MSISDN, IMSI, NAI or AccountId) and Row Identifier	`<req name="select"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value`

Command	Description	Key(s)	Command Syntax
		and Field name	val="rowIdValue"> </expr> …
Update Row Field	Update field(s) to the specified value(s).		<req name="update"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …
Delete Row Field	Delete all value(s) for the specified field(s).		<req name="delete"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …

Get Row Field

This operation retrieves a fields(s) within a data row for the subscriber identified by the *keyName* and *keyValue*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName/instanceFieldValue*. The field name(s) are specified in *fieldNameX*.

All data rows that match the requested *rowName/rowValue* and *instanceFieldName/instanceFieldValue* (if specified) are returned.

Note: If the specified row does not exist, the request will fail. If the specified row exists, but the field does not exist, this is not treated as an error, and no row/field data is deleted.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for Quota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row already existed called "DayPass", then an attempt to get a field in a row called "DayPass" would be successful, but an attempt to get a field in a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to delete a row with a field with the value "Data" would be successful, but an attempt to delete a row with a field with the value "DATA" would fail.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

A data row with the given identifier/instance within the transparent data should exist for the subscriber.

The field name(s) specified must be valid fields for the Entity as defined in the SEC.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <select>
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
[
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]
  </select>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 65: Request Variable Definitions: Get Row Field

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the Quota transparent data
fieldNameX	A user defined field within the data row	

Variable	Definition	Value
fieldValueX	Corresponding field value assigned to *fieldNameX* **Note:** For multi-value fields, the value can contain a comma separated list of values.	
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyNameX*	
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the Quota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv>
[
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv>
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv>
]
    </row>
[
    <row>
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv>
[
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv>
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv>
]
    </row>
```

```
    :
    <row>
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv>
[
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv>
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv>
]
    </row>
]
  </rset>
]
</req>
```

Table 66: Response Variable Definitions: Get Row Field

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Row Field* for other values.	
affected	The number of subscriber data rows from which data is returned. A value of "1" is expected for success.	
rowValue	The value of the requested field. **Note:** For multi-value fields, the value will contain a comma separated list of values on a single line. E.g. "a,b,c".	

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful. One <row> element is returned per matching row. One `<rv>` (row value) element exists for every *fieldNameX* supplied in the original request. The `<rv>` elements are ordered the same as the *fieldNameX* fields were specified in the original request. If the field is valid, but not present in the entity, this is indicated with `<rv null="y">`. If the field is present , but has an empty value, this is indicated with `<rv></rv>`.

Error Codes

Table 67: Error Codes: Get Row Field

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found
ROW_NOT_FOUND	Data row specified is not found

Examples

Request #1

A request is made to get the *inputVolume* field in the *Q1* data row of the Quota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="inputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the requested field value *980* is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>980</rv>
    </row>
  </rset>
</req>
```

Request #2

A request is made to get the *outputVolume* and *cid* fields in the *Q2* data row of the Quota data. The Quota data contains two rows called *Q2*. One with `<cid>` of *11223344*, the other with a `<cid>` of *99887766*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
```

```
  <select>
    <expr><attr name="outputVolume"/></expr>
    <expr><attr name="cid"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q2"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the requested field values are returned from each row. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>220</rv>
      <rv>11223344</rv>
    </row>
    <row>
      <rv>1050</rv>
      <rv>99887766</rv>
    </row>
  </rset>
</req>
```

Request #3

A request is made to get the *outputVolume* field in the *Q3* data row of the Quota data, with the `<cid>` value of *11223344*. The Quota data contains two rows called *Q3*. One with `<cid>` of *11223344*, the other with a `<cid>` of *99887766*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the requested field value *4000* is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>4000</rv>
    </row>
  </rset>
</req>
```

Request #4

A request is made to get the *inputVolume*, *outputVolume*, and *totalVolume* fields in the *Q1* data row of the Quota data. The *inputVolume* field is present in the *Q1* quota, the *outputVolume* field is present in the Q1 quota, but has an empty value, and the *totalVolume* field is not present in the Q1 quota (but is a valid field). The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="inputVolume"/></expr>
    <expr><attr name="outputVolume"/></expr>
    <expr><attr name="totalVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #4

The request is successful, and the requested field values are returned. The *inputVolume* field is set to 980, the *outputVolume* field is indicated as being present, but empty, and the *totalVolume* field is indicated as not being present. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>980</rv>
      <rv></rv>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

Update Row Field

This operation updates field(s) within a data row for the subscriber identified by the *keyName* and *keyValue*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field name(s) are specified in *fieldNameX*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. If the specified field(s) are valid but do not currently exist, they will be created.

If more than one row matches the requested *rowName*/*rowValue* and *instanceFieldName*/*instanceFieldValue* (if specified), then the update request will fail.

Note: If the specified row does not exist, the request will fail.

Note: If the requested field(s) exists, are valid, but not currently present, they will be created.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for Quota, the element<quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row already existed called "DayPass", then an attempt to update a field in a row called "DayPass" would be successful, but an attempt to update a field in a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to delete a row with a field with the value "Data" would be successful, but an attempt to delete a row with a field with the value "DATA" would fail.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

A data row with the given identifier/instance within the transparent data should exist for the subscriber.

The field name(s) specified must be valid fields for the Entity as defined in the SEC.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
[
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]
  </set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 68: Request Variable Definitions: Update Row Field

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)

Variable	Definition	Value
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the Quota transparent data
fieldNameX	A user defined field within the data row	
fieldValueX	Corresponding key field value assigned to *fieldNameX* **Note:** For multi-value fields, the value can contain a comma separated list of values on a single line. E.g. "a,b,c"	
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the Quota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 69: Response Variable Definitions: Update Row Field

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Update Row Field* for other values.	
affected	The number of data rows updated. A value of "1" is expected for success.	

Error Codes

Table 70: Error Codes: Update Row Field

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FIELD_NOT_UPDATABLE	Field Cannot be Updated. The field is defined in the SEC as not be updatable
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found
ROW_NOT_FOUND	Data row specified is not found

Error Code	Description
MULITPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria

Examples

Request #1

A request is made to update the *inputVolume* field in the *Q1* data row of the Quota data. The *Q1* data row exists in the Quota data, and is there is only one row called *Q1*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req
```

Response #1

The request is successful, and the field in the data row in the Quota data was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to update the *cid* field in the *Q1* data row in the Quota data. The *Q1* data row exists in the Quota data, and is there is only one row called *Q1*. The *cid* field is not allowed to be updated. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="cid"/><value val="11223344"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

Response #2

The request fails. The `error` value indicates the `cid` field cannot be updated, and the affected rows are `0`. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70016" affected="0"/>
</req>
```

Request #3

A request is made to update the *outputVolume* field in the *Q6* data row of the Quota data. The *Q6* data row exists in the Quota data, but there are two rows called *Q3*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q6"/></expr>
  </where>
</req>
```

Response #3

The request fails because there was more than one row called *Q6*. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

Delete Row Field

This operation deletes a field(s) within a data row for the subscriber identified by the *keyName* and *keyValue*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field name(s) are specified in *fieldNameX*.

If more than one row matches the requested *rowName*/*rowValue* and *instanceFieldName*/*instanceFieldValue* (if specified), then the update request will fail.

Note: If the specified row does not exist, the request will fail. If the specified row exists, but the field does not exist, this is not treated as an error, and no row/field data is deleted.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for Quota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row existed called "DayPass", then an attempt to delete a field in a row called "DayPass" would be successful, but an attempt to delete a field in a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to delete a field in a row with a field with the value "Data" would be successful, but an attempt to delete a field in a row with a field with the value "DATA" would fail.

Note: A request to delete a field(s) cannot be mixed with a request to update a field, otherwise the request will fail. A request to delete a field(s) must only contain field(s) being deleted.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

At least one data row with the given identifier/instance within the transparent data should exist for the subscriber.

The field name(s) specified must be valid fields for the Entity as defined in the SEC.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><op value="="/><value val=""
isnull="y"/></expr>
[
    <expr><attr name="fieldName2"/><op value="="/><value val=""
isnull="y"/></expr>
              :
    <expr><attr name="fieldNameN"/><op value="="/><value val=""
isnull="y"/></expr>
]
  </set>
  <where>
    <expr><attr name="keyName"/><op value="="/><value val="keyValue"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 71: Request Variable Definitions: Delete Row Field

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request *n* — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295

Variable	Definition	Value
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the Quota transparent data
fieldNameX	A user defined field within the data row	
fieldValueX	Corresponding key field value assigned to *fieldNameX* **Note:** For multi-value fields, the value can contain a comma separated list of values on a single line. E.g. "a,b,c"	
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the Quota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 72: Response Variable Definitions: Delete Row Field

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The id value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Row Field* for other values.	
affected	The number of data entities updated. A value of "1" indicates the row existed and the field was deleted. A value of "0" indicates the field did not exist.	

Error Codes

Table 73: Error Codes: Delete Row Field

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found
ROW_NOT_FOUND	Data row specified is not found
MULITPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria

Examples

Request #1

A request is made to delete the *inputVolume* field in the *Q1* data row of the Quota data. The *Q1* data row exists in the Quota data, and is there is only one row called *Q1*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the field in the data row in the Quota data was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to delete the *outputVolume* field in the *Q3* data row of the Quota data. The Quota data contains two rows called *Q3*. The request is not required in the response.

```
<req name="update">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
  </where>
</req>
```

Response #2

The request fails because there are two Quota rows called *Q3*. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

Request #3

A request is made to update delete the *outputVolume* field in the *Q4* data row of the Quota data with the <cid> 11223344. The Q4 data row exists in the Quota data, and is there are two rows called *Q4*, one with <cid>*11223344* and one with <cid>*99887766*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q4"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the *outputVolume* field in the *Q4* data row in the Quota data was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Special Operations

A transparent data entity may contain data that is organized in "rows". An example of a row is a specific quota within the Quota entity.

The required row is identified in the request by the *RowIdName*/*RowIdValue*.

A specific instance of a quota (i.e. a specified row) within the Quota transparent data entity can have its fields reset to pre-defined values using a provisioning command.

Table 74: Summary of Subscriber Special Operation Commands

Command	Description	Key(s)	Command Syntax
Reset Quota	Reset the fields within the specified Quota	MSISDN, IMSI, NAI or AccountId	`<req name="operation"> <oper name="ResetQuota">`

Reset Quota

This operation resets a particular quota row within the Quota data associated with a subscriber.

If the subscriber has Quota data, then the configured values within the specified quota row are reset to the configured default values.

Note: The *quotaName* is case-sensitive. If a row existed called "DayPass", then an attempt to reset a quota row called "DayPass" would be successful, but an attempt to reset a quota row called "DAYPASS" would fail.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The Quota transparent data must exist for the subscriber.

The specified Quota row must exist in the Quota transparent data.

Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="ResetQuota">
    <expr><param name="keyName"/><op value="="/>
          <value val="keyValue"/></expr>
    <expr><param name="name"/><op value="="/>
          <value val="quotaName"/></expr>
  </oper>
</req>
```

Table 75: Request Variable Definitions: Reset Quota

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	
quotaName	The name that identifies the required quota row within the Quota data blob	

Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
```

```
]
  <res error="error" affected="affected"/>
</req>
```

Table 76: Response Variable Definitions: Reset Quota

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Reset Quota* for other values.	
affected	The number of quota rows reset. A value of "1" is expected for success.	

Error Codes

Table 77: Error Codes: Reset Quota

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found
ROW_NOT_FOUND	Data row specified is not found
MULITPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria

Examples

Request #1

A request is made to reset the *Q1* Quota row for a subscriber. The subscriber has Quota data, and the Quota data contains a row called *Q1*. The request is not required in the response.

```
<req name="operation">
  <oper name="ResetQuota">
    <expr><param name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><param name="name"/><op value="="/><value val="Q1"/></expr>
  </oper>
</req>
```

Response #1

The request is successful, and the specified Quota row was reset. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to reset the *Monthly* Quota row. The subscriber does not have Quota data. The request is not required in the response.

```
<req name="operation">
  <oper name="ResetQuota">
    <expr><param name="MSISDN"/><op value="="/>
          <value val="15141234567"/></expr>
    <expr><param name="name"/><op value="="/><value val="Monthly"/></expr>
  </oper>
</req>
```

Response #2

The request fails. The `error` value indicates the subscriber does not have Quota data, and the `affected` rows are `0`. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

Request #3

A request is made to reset the *Q6* Quota row. The subscriber has Quota data, but the Quota data does NOT contain a Quota row called *Q6*. The request is not required in the response.

```
<req name="operation">
  <oper name="ResetQuota">
    <expr><param name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><param name="name"/><op value="="/><value val="Q6"/></expr>
  </oper>
</req>
```

Response #3

The request fails because the *Q6* data row was not present. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70032" affected="0"/>
</req>
```

Chapter 7

Pool Provisioning

Topics:

This chapter describes how to provision various pool commands.

Pools are used to group subscribers that share common data. Subscribers in a pool share all the entities of that pool.

Provisioning clients can create, retrieve, modify, and delete pool data. Pool data is accessed via the *PoolID* value associated with the pool.

Note: For command responses, the error code values described are listed in *Error Codes*.

Pool Profile Commands

Table 78: Summary of Pool Profile Commands

Command	Description	Key(s)	Command Syntax
Create Pool	Creates a new pool profile using the field-value pairs that are specified in the request content.	-	`<req name="insert">` `<ent name="Pool"/>`
Get Pool	Get pool profile dat	PoolID	`<req name="select">` `<ent name="Pool"/>`
Delete Pool	Delete pool profile data and all opaque data associated with the pool	PoolID	`<req name="delete">` `<ent name="Pool"/>`

Create Pool

This operation creates a new pool profile using the field-value pairs that are specified in the request content.

Unlike other pool commands, the key value (PoolID) is not specified in the request as part of the "where" tagg. Request content includes, *poolId*, and field value pairs, all as specified in the Subscriber Entity.

Note: The pool profile data provided is fully validated against the definition in the SEC. If the validation check fails, then the request is rejected.

Note: An entire entity for the pool can be created by specifying a *cdataFieldName* corresponding to the interface entity in the SEC, and supplying the entire XML blob value in *cdataFieldValue*. The *cdataFieldName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: Multi-value fields can be specified by a single*fieldNameX* value with a delimited list of values, or multiple *fieldNameX* fields, each containing a single value.

Prerequisites

A pol with the supplied PoolID must not exist.

Request

```
<req name="insert" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolID"/><value val="poolId"/></expr>
[
    <expr>
<
      <attr name="fieldName1"/><value val="fieldValue1"/>
```

```
|
      <attr name="cdataFieldName1"/><op value="="/>
        <cdata><![CDATA[cdataFieldValue1]]></cdata>
>
    </expr>

    <expr>
<
      <attr name="fieldName2"/><value val="fieldValue2">
|
      <attr name="cdataFieldName2"/><op value="="/>
        <cdata><![CDATA[cdataFieldValue2]]></cdata>
>
    </expr>
    :
    <expr>
<
      <attr name="fieldNameN"/><value val="fieldValueN"/>
|
      <attr name="cdataFieldNameN"/><op value="="/>
        <cdata><![CDATA[cdataFieldValueN]><</cdata>
>
    </expr>
]
  </set>
</req>
```

Table 79: Request Variable Definitions: Create Pool

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
poolId	PoolID value of the pool being created Numeric value, 1-22 digits in length	1-999999999999999999999
fieldNameX	A user defined field within the pool Profile	
fieldValueX	Corresponding field value assigned to *fieldNameX* **Note:** For multi-value fields, the value will contain a comma separated list of values on a single line. E.g. "a,b,c"	

Variable	Definition	Value
cdataFieldNameX	A user defined field within the pool Profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC	• PoolQuota • PoolState • PoolDynamicQuota
cdataFieldValueX	Contents of the XML data "blob" for *cdataFieldNameX*	

Note: PoolID/field order in the request is not important.

Response

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 80: Response Variable Definitions: Create Pool

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Create Pool* for other values.	
affected	The number of PoolProfile rows created. A value of "1" is expected for success.	

Error Codes

Table 81: Error Codes: Create Pool

Error Code	Description
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
INVALID_SOAP _XML	Invalid SOAP XML
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_EXISTS	Key Already Exists. A subscriber/pool already exists with the given key
INVALID_KEY_VALUE	The key value supplied is invalid, due to invalid characters/format etc.
MULT_VER_TAGS_FOUND	Multiple Version Tags Found
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
INVALID_XML	Invalid Input XML

Examples

Request #1

A pool is created, with PoolID. The *BillingDay* and *Entitlement* fields are set. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPassPlus"/></expr>
  </set>
</req>
```

Response #1

The request is successful, and the pool was created.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A pool is created, with PoolID. The *BillingDay* and *Entitlement* fields are set. The *PoolQuota* and *PoolState* entities are also created. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
    <expr><attr name="PoolQuota"/><op value="="/>
      <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
                     <usage>
                       <version>3</version>
                       <quota name="Weekend">
                         <totalVolume>500</totalVolume>
                         <Type>quota</Type>
                         <QuotaState>active</QuotaState>
                         <nextResetTime>2014-01-10T02:00:00</nextResetTime>
                       </quota>
                       <quota name=Evenings">
                         <totalVolume>300</totalVolume>
                         <Type>quota</Type>
                         <QuotaState>active</QuotaState>
                         <nextResetTime>2014-01-10T02:00:00</nextResetTime>
                       </quota>
                     </usage>]]>
      </cdata>
    </expr>
    <expr><attr name="PoolState"/><op value="="/>
      <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <state>
          <version>1</version>
          <property>
            <name>shared</name>
            <value>yes</value>
          </property>
          <property>
            <name>expire</name>
            <value>2014-02-09T11:20:32</value>
          </property>
        </state>]]>
      </cdata>
    </expr>
  </set>
</req>
```

Response #2

The request is successful, and the pool was created.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Get Pool

This operation retrieves all field-value pairs created for a pool that is identified by the *poolId*.

A *poolId* is required in the request in order to identify the pool. The response content includes only valid field-value pairs which have been previously provisioned or created by default.

Prerequisites

A pool with a key of the *poolId* supplied must exist.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 82: Request Variable Definitions: Get Pool

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	Transaction id value provided in request, and will be passed back in the response	1-4294967295
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999

Response

```
req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>
          <![CDATA[cdataRowValue]]>
      </rv>
    </row>
  </rset>
]
</req>
```

Table 83: Response Variable Definitions: Get Pool

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request	
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Text Error Codes: Get Pool* for other values.	
affected	The number of data rows returned. A value of "1" is expected for success.	
cdataRowValue	Contents of the pool Profile XML data "blob"	

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful. Only a single `<row>` element is returned, with a single `<rv>` (row value) element containing an XML CDATA construct containing the requested pool profile data (i.e. XML blob).

Error Codes

Table 84: Error Codes: Get Pool

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
INTF_ENTY_NOT_FOUND	Interface Entity Not Found

Examples

Request #1

A request is made to get pool Profile data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Pool"/>
```

```
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the pool Profile data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
    <rset>
      <row>
        <rv>
          <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
            <pool>
              <field name="PoolID">100000</field>
              <field name="BillingDay">5</field>
              <field name="Tier">12</field>
              <field name="Entitlement">Weekpass</field>
              <field name="Entitlement">Daypass</field>
              <field name="Custom15">allo</field>
            </pool>]]>
        </rv>
    </row>
  </rset>
</req>
```

Delete Pool

This operation deletes all profile data (field-value pairs) and opaque data for the pool that is identified by the *poolId*.

Prerequisites

A pool with a key of the *poolId* supplied must exist.

The pool must have no subscriber members, or the request will fail.

Request

```
<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 85: Request Variable Definitions: Delete Pool

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including	• y — only provide the result, do NOT include the original request

Variable	Definition	Value
	the original request in the response	• n — include the original request in the response (default)
id	Transaction id value provided in request, and will be passed back in the response	1-4294967295
poolId	PoolID value of the pool being created. Numeric value, 1-22 digits in length	1-999999999999999999999

Response

```
<req name="delete" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 86: Response Variable Definitions: Delete Pool

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Pool* for other values.	
affected	The number of pools deleted. A value of "1" is expected for success.	

Error Codes

Table 87: Error Codes: Delete Pool

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found.
HAS_POOL_MEMBERS	Has Pool Members. A pool cannot be deleted when it has member subscribers.
INTF_ENTY_NOT_FOUND	Interface Entity Not Found

Examples

Request #1

The pool with the given PoolID is deleted. The pool exists. The request should not be included in the response.

```
<req name="delete" resonly="y">
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the pool was deleted.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

The pool with the given PoolID is deleted. The pool does NOT exist. The request should not be included in the response.

```
<req name="delete" resonly="y">
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

Response #2

The request fails. The `error` value indicates a pool with the given PoolID does not exist, and the `affected` rows are `0`. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70019" affected="0"/>
</req>
```

Request #3

The pool with the given PoolID is deleted. The pool exists, but has member subscribers. The request should not be included in the response.

```
<req name="delete" resonly="y">
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

Response #3

The request fails. The `error` value indicates the pool has member subscribers , and the `affected` rows are `0`. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70022" affected="0"/>
</req>
```

Pool Field Commands

Table 88: Pool Field Commands

Command	Description	Key(s)	Command Syntax
Add Field Value	Add a value to the specified field. This operation does not affect any pre-existing values for the field	PoolID	`<req name="update"> <ent name="Pool"/> <oper name="AddToSet">`
Get Field	Retrieve the value(s) for the specified field(s)		`<req name="select"> <ent name="Pool"/>`
Update Field	Update field(s) to the specified value(s)		`<req name="update"> <ent name="Pool"/>`
Delete Field	Delete all the values for the specified field(s)		`<req name="update"> <ent name="Pool"/> … <value val="" isnull="y"/>…`
Delete Field Value	Delete field(s) with the specified value(s)		`<req name="update"> <ent name="Pool"/> <oper name="RemoveFromSet>`

Add Field Value

This operation adds one or more value(s) to the specified multi-value field for the pool identified by the *poolId*.

This operation can only be executed for the fields defined as multi-value field in the Subscriber Entity Configuration. Any pre-existing values for the field are not effected.

All existing values are retained, and the new values(s) specified are inserted. For example, if the current value of a field was "a,b,c", and this command was used with value "d", after the update the field would have the value "a,b,c,d".

Note: If a value being added already exists, the request will fail.

Note: The *fieldValue* is case-sensitive. An attempt to add the value "a" to current field value of "a,b,c" would fail, but an attempt to add the value "A" would be successful and result in the field value being "a,b,c,A".

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The field *fieldName* must be a valid field in the pool Profile, and must be a multi-value field.

Each *fieldValueX* being added must NOT already be present in the field.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="fieldName"/><value val="fieldValue1"/></expr>
[
      <expr><attr name="fieldName"/><value val="fieldValue2"/></expr>
      :
      <expr><attr name="fieldName"/><value val="fieldValueX"/></expr>]
]
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 89: Request Variable Definitions: Add Field Value

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
fieldName	A user defined field within the pool Profile	
fieldValueX	Corresponding field value assigned to *fieldName*	
PoolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999

Note: Each *keyValueX* must contain a single value for the multi-value field. To add more than one value, include multiple `<expr>` elements for the field, each containing a single value.

Response

```
<req name="update" [resonly="resonly"][id="id"]>
[
  originalXMLRequest
```

```
]
  <res error="error" affected="affected"/>
</req>
```

Table 90: Response Variable Definitions: Add Field Value

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Add Field Value (Pool)* for other values.	
affected	The number of pool Profiles updated. A value of "1" is expected for success.	

Error Codes

Table 91: Error Codes: Add Field Value (Pool)

Error Code	Description
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FIELD_NOT_UPDATABLE	Field Cannot be Updated. The field is defined in the SEC as not be updatable.
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
VALUE_EXISTS	List value already exists.

Error Code	Description
FLD_NOT_MULTI	Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value.

Examples

Request #1

A request is made to add the value *DayPass* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* value is not already present in the *Entitlement* field. The request is not required in the response. An id value is supplied, which is required in the response.

```
<req name="update">
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the value was added to the *Entitlement* field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to add the values *HighSpeed* and *Unlimited* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. Neither value is already present in the *Entitlement* field. The request is not required in the response.

```
req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="HighSpeed"/></expr>
      <expr><attr name="Entitlement"/><value val="Unlimited"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the values were added to the *Entitlement* field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #3

A request is made to add the value *Gold* to the *Tier* field. The *Tier* field is NOT a valid multi-value field. The request is not required in the response.

```
<req name="update">
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Tier"/><value val="Gold"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #3

The request fails. The *error* value indicates the *Tier* field is not a multi-value field, and the *affected* rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70034" affected="0"/>
</req>
```

Get Field

This operation retrieves the values for the specified field(s) for the pool identified by the specified *poolId*.

Note: An entire entity for the pool can be retrieved by specifying an *opaqueDataType* corresponding to the interface entity name in the SEC. The *opaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail.

Prerequisites

A pool with a key of the *poolId* supplied must exist.

Each requested field *fieldNameX* must be a valid field in the subscriber Profile.

Each requested *opaqueDataTypeX* must reference a valid Entity in the Interface Entity Map table in the SEC.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <select>
[
```

```
    <expr><attr name="fieldName1"/></expr>
    <expr><attr name="fieldName2"/></expr>
    :
    <expr><attr name="fieldNameN"/></expr>
]
[
    <expr><attr name="opaqueDataType1"/></expr>
    <expr><attr name="opaqueDataType2"/></expr>
    :
    <expr><attr name="opaqueDataTypeN"/></expr>
]

  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 92: Request Variable Definitions: Get Field (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	Transaction id value provided in request, and will be passed back in the response	1-4294967295
fieldNameX	A user defined field within the pool Profile	
fieldValueX	Corresponding field value assigned to *fieldNameX*	
opaqueDataTypeX	A user defined field within the subscriber Profile, that represents a transparent or opaque data entity	• PoolQuota • PoolState • PoolDynamicQuota
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999

Note: At least one *fieldNameX*/*opaqueDataTypeX* field must be requested.

Note: The order in which *fieldNameX*/*opaqueDataTypeX* are specified in the request is not important.

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
```

```
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
[
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv>
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv>
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv>
]
[
      <rv>cdataRowValue1</rv> | <rv null="y">
      <rv>cdataRowValue2</rv> | <rv null="y">
      :
      <rv>cdataRowValueN</rv> | <rv null="y">
]
    </row>
  </rset>
]
</req>
```

Table 93: Response Variable Definitions: Get Field (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The*resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Field (Pool)* for other values.	
affected	The number of pool Profiles from which data is returned. A value of "1" is expected for success .	

Variable	Definition	Value
rowValueX	The value of the requested field (for normal fields, not for opaque/transparent entities). **Note:** For multi-value fields, the value will contain a comma separated list of values on a single line. E.g. "a,b,c"	
cdataRowValueX	Contents of the XML data "blob" (for requested fields that are opaque/transparent entities)	

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful. Only a single `<row>` element is returned. One `<rv>` (row value) element exists for every *fieldNameX* or *opaqueDataTypeX* supplied in the original request. The `<rv>` elements are ordered the same as the *fieldNameX / opaqueDataTypeX* fields were specified in the original request. If the field is valid, but not present in the entity, this is indicated with `<rv null= "y" >`. If the field is present, but has an empty value, this is indicated with `<rv></rv>` .

Error Codes

Table 94: Error Codes: Get Field (Pool)

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC.
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to get the *PoolID*, *Entitlement*, *Tier*, and *BillingDay* fields. The request is not required in the response.

```
<req name="select" >
  <ent name="Pool"/>
  <select>
    <expr><attr name="PoolID"/></expr>
    <expr><attr name="Entitlement"/></expr>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="BillingDay"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the 4 requested values are returned (the *Entitlement* is a multi-value field). The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>1000</rv>
      <rv>DayPass,WeekPass,Weekend</rv>
      <rv>InfinityPrepaid</rv>
      <rv>23</rv>
    </row>
  </rset>
</req>
```

Request #2

A request is made to get the *MSISDN* and *BillingDay* fields, as well as the *PoolQuota* and *PoolState* entity data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Pool"/>
  <select>
    <expr><attr name="PoolID"/></expr>
    <expr><attr name="BillingDay"/></expr>
    <expr><attr name="PoolQuota"/></expr>
    <expr><attr name="PoolState"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the 4 requested values are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>2000</rv>
      <rv>11</rv>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="AggregateLimit">
              <cid>9223372036854775807</cid>
              <time>3422</time>
              <totalVolume>1000</totalVolume>
              <inputVolume>980</inputVolume>
              <outputVolume>20</outputVolume>
              <serviceSpecific>12</serviceSpecific>
              <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
            </quota>
          </usage>]]>
        </rv>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <state>
            <version>1</version>
```

```
            <property>
              <name>mcc</name>
              <value>315</value>
            </property>
            <property>
              <name>expire</name>
              <value>2010-02-09T11:20:32</value>
            </property>
            <property>
              <name>approved</name>
              <value>yes</value>
            </property>
          </state>]]>
        </rv>
    </row>
  </rset>
</req>
```

Request #3

A request is made to get the *Custom10*, *Entitlement*, *Tier*, and *Custom20* fields. The *Entitlement* and *Tier* fields are set in the XML blob, the *Custom10* field is not set, and the *Custom20* field is set, but has an empty value. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Pool"/>
  <select>
    <expr><attr name="Custom10"/></expr>
    <expr><attr name="Entitlement"/></expr>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="Custom20"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the 4 requested values are returned (the *Entitlement* is a multi-value field). The *Custom10* field is indicated as unset, and the *Custom20* field is indicated as empty. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
      <rv>1,14,2,8</rv>
      <rv>InfinityPrepaid</rv>
      <rv></rv>
    </row>
  </rset>
</req>
```

Update Field

This operation updates a field(s) to the specified values for the pool identified by the specified *poolId*. This operation replaces ("sets") the value(s) of the field(s), which means that any existing value(s) for the field(s) are deleted first.

For multi-value fields, all existing values are removed, and only the new values(s) specified are inserted. Adding values to a current set is accomplished using Add Field Value. For example, if the current value of a field was "a,b,c", and this command was used with value "d", after the update the field would have the value "d" (it would NOT be "a,b,c,d").

All fields are updated at once in the DB. All fields and all values must be valid for the update to be successful. I.e., as soon as one error is detected during processing, the request is abandoned (and an error returned). For example, if the third specified field fails validation, then none of the fields are updated.

Note: If the requested field(s) are valid, but not currently present, they will be created.

Note: An entire entity for the pool can be replaced by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the entire XML blob value in *cdataFieldValue.* The *cdataFieldName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: Multi-value fields can be specified by a single *fieldNameX* value with a delimited list of values, or multiple *fieldNameX* fields, each containing a single value.

Note: A request to update a field(s) cannot be mixed with a request to delete a field, otherwise the request will fail. A request to update a field(s) must only contain field(s) being updated.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The field *fieldName* must be a valid field in the pool Profile.

Each requested *cdatafFieldName* must be a valid pooled transparent/opaque interface entity name for a pool.

Request

```
<req name="update" [resonly="resonly"] [id="id]>
  <ent name="Pool"/>
  <set>
    <expr>
<
      <attr name="fieldName1"/><value val="fieldValue1"/>
|
      <attr name="cdataFieldName1"/><op value="="/>
        <cdata><![CDATA[cdataFieldValue1]]></cdata>
>
    </expr>
[
    <expr>
<
      <attr name="fieldName2"/><value val="fieldValue2"/>
|
      <attr name="cdataFieldName2"/><op value="="/>
        <cdata><![CDATA[cdataFieldValue2]]></cdata>
```

```
>
    </expr>
    :
    <expr>
<
      <attr name="fieldNameN"/><value val="fieldValueN"/>
|
      <attr name="cdataFieldNameN"/><op value="="/>
        <cdata><![CDATA[cdataFieldValueN]]?</cdata>
>
    </expr>
]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 95: Request Variable Definitions: Update Field (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
fieldNameX	A user defined field within the pool Profile	
fieldValueX	Corresponding field value assigned to *fieldNameX*	
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999
cdataFieldNameX	A user defined field within the pool Profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC	• PoolQuota • PoolState • PoolDynamicQuota
cdataFieldValueX	Contents of the XML data "blob" for *cdataFieldNameX*	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
```

```
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 96: Response Variable Definitions: Update Field (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Update Field (Pool)* for other values.	
affected	The number of pool Profiles updated. A value of "1" is expected for success.	

Error Codes

Table 97: Error Codes: Update Field (Pool)

Error Code	Description
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FIELD_NOT_UPDATABLE	Field Cannot be Updated. The field is defined in the SEC as not be updatable
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INVALID_XML	Invalid Input XML

Examples

Request #1

A request is made to update the value of the *BillingDay* field to *23*, and the *Tier* field to *Gold*. The request is not required in the response.

```
<req name="update">
  <ent name="Pool"/>
  <set>
    <expr><attr name="BillingDay"/><value val="23"/></expr>
    <expr><attr name="Tier"/><value val="Gold"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the *BillingDay* and *Tier* values are updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to update the value of the *BillingDay* field to *18*, and the entire *PoolState* entity. The request is not required in the response.

```
<req name="update">
  <ent name="Pool"/>
  <set>
    <expr><attr name="BillingDay"/><value val="18"/></expr>
    <expr><attr name="PoolState"/><op value="="/>
      <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <state>
          <version>1</version>
          <property>
            <name>shared</name>
            <value>yes</value>
          </property>
          <property>
            <name>expire</name>
            <value>2014-02-09T11:20:32</value>
          </property>
        </state>]]>
      </cdata>
    </expr>
  </set>
```

```
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the *BillingDay* and *PoolState* values are updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Delete Field

This operation deletes the specified fields for the pool identified by *poolId* in the request.

If the field is multi-value field then all values are deleted. Deletion of a field results removal of the entire field from the pool Profile. I.e. the field is not present, not just the value is empty.

Note: The field being deleted does NOT need to have a current value. It can be empty (i.e. deleted) already, and the request will succeed.

Note: A request to delete a field(s) cannot be mixed with a request to update a field, otherwise the request will fail. A request to delete a field(s) must only contain field(s) being deleted.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

Each requested field *fieldNameX* must be a valid field in the pool Profile.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="fieldName1"/><op value="="/>
          <value val="" isnull="y"/></expr>
[
    <expr><attr name="fieldName2"/><op value="="/>
          <value val="" isnull="y"/></expr>
    :
    <expr><attr name="fieldNameN"/><op value="="/>
          <value val="" isnull="y"/></expr>
]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 98: Request Variable Definitions: Delete Field (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
fieldNameX	A user defined field within the pool Profile	
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 99: Response Variable Definitions: Delete Field (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Field (Pool)* for other values.	

Variable	Definition	Value
affected	The number of pool Profiles updated. A value of "1" is expected for success.	

Error Codes

Table 100: Error Codes: Delete Field (Pool)

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to delete the *BillingDay* and *Tier* fields. Both fields are valid pool Profile fields. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="BillingDay"/><op value="="/>
          <value val="" isnull="y"/></expr>
    <expr><attr name="Tier"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the two fields were deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Delete Field Value

This operation deletes one or more value(s) from the specified field for the pool identified by the *poolId* in the request.

This operation can only be executed for the fields defined as multi-value field in the Subscriber Entity Configuration.

Each individual value is removed from the pool Profile. If a supplied value does not exist, then it is ignored. For example, if a profile contains values "A,B,C" and a request to delete "A,B" is made, this

succeeds and the profile is left with "C" as the value. If the profile contains "A,B,C" and a request is made to delete "C,D" the request succeeds and the profile is left with "A,B" as the value.

If all values are removed, the entire field is removed from the pool Profile (i.e. there is no XML element present).

Note: The *fieldValue* is case-sensitive. An attempt to remove the value "A" from a current field value of "A,B,C" would be successful, but an attempt to remove the value "a" would fail.

Note: A request to delete a field(s) cannot be mixed with a request to update a field, otherwise the request will fail. A request to delete a field(s) must only contain field(s) being deleted.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The field *fieldName* must be a valid field in the pool Profile, and must be a multi-value field.

Each *fieldValueX* being added must be present in the field.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="fieldName"/><value val="fieldValue1"/></expr>
[
      <expr><attr name="fieldName"/><value val="fieldValue2"/></expr>
      :
      <expr><attr name="fieldName"/><value val="fieldValueX"/></expr>]
]
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 101: Response Variable Definitions: Delete Field Value (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
fieldName	A user defined field within the subscriber Profile	

Variable	Definition	Value
fieldValueX	Corresponding field value assigned to *fieldName*	
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999

Note: Each *keyValueX* must contain a single value for the multi-value field. To remove more than one value, include multiple `<expr>` elements for the field, each containing a single value.

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 102: Response Variable Definitions: Delete Field Value (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Field Value (Pool)* for other values.	
affected	The number of pool Profiles updated. A value of "1" is expected for success.	

Error Codes

Table 103: Error Codes: Delete Field Value (Pool)

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
FLD_NOT_MULTI	Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value.

Examples

Request #1

A request is made to remove the value *DayPass* from the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* value is present in the *Entitlement* field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the value was removed from the *Entitlement* field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to remove the values *WeekendPass* and *Unlimited* from the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *WeekendPass* value is present in the *Entitlement* field, but the *Unlimited* value is not. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <oper name="RemoveFromSet"/>
      <expr><attr name="Entitlement"/><value val="WeekendPass"/></expr>
      <expr><attr name="Entitlement"/><value val="Unlimited"/></expr>
```

```
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the *WeekendPass* value was removed from the *Entitlement* field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Pool Opaque Data Commands

Table 104: Summary of Pool Opaque Data Commands

Command	Description	Key(s)	Command Syntax
Create Opaque Data	Insert pool opaque data of the specified type	PoolID	`<req name="insert"> <ent name="Pool"/>`
Get Opaque Data	Retrieve pool opaque data of the specified type		`<req name="select"> <ent name="Pool"/>`
Update Opaque Data	Update pool opaque data of the specified type		`<req name="update"> <ent name="Pool"/>`
Delete Opaque Data	Delete pool opaque data of the specified type		`<req name="update"> <ent name="Pool"/> … <value val="" isnull="y"/>…`

Create Opaque Data

This operation creates the opaque data of the specified *opaqueDataType* for the pool identified by the *poolId* in the request.

The pool opaque data is provided in the request within a `<cdata>` construct.

Note: The opaque data for creating an entity/row is provided in the request within a CDATA construct.

Note: The opaque data provided is always checked to be valid XML. If the entity is defined as transparent in the SEC, then the XML blob is fully validated against the definition in the SEC. If either validation check fails, then the request is rejected.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *poolOpaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.

No pool opaque data of the *poolOpaqueDataType* must already exist for the pool (unless the odk attribute is specified).

Request

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="poolOpaqueDataType"/><op value="="/><cdata>
<![CDATA[
cdataFieldValue
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 105: Request Variable Definitions: Create Opaque Data (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
odk	(Optional) Indicates that the insert request should be converted to an update if the opaque data type specified already exists	
poolOpaqueDataType	A user defined type/name for the pool opaque data	• PoolQuota • PoolState • PoolDynamicQuota
cdataFieldValue	Contents of the XML data "blob"	
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999

Response

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 106: Response Variable Definitions: Create Opaque Data (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Create Opaque Data (Pool)* for other values.	
affected	The number of opaque data rows inserted/updated for the pool. A value of "1" is expected for success.	

Error Codes

Table 107: Error Codes: Create Opaque Data (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
MULT_VER_TAGS_FOUND	Multiple Version Tags Found
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field

Error Code	Description
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INVALID _XML	Invalid Input XML
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_EXISTS	Register Already Exists

Examples

Request #1

A request is made to create the *PoolQuota* opaque data. The PoolQuota XML blob is supplied whole. The request is not required in the response.

```
req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolQuota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
  <cid>9223372036854775807</cid>
  <time>3422</time>
  <totalVolume>1000</totalVolume>
  <inputVolume>980</inputVolume>
  <outputVolume>20</outputVolume>
  <serviceSpecific>12</serviceSpecific>
  <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage >
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
 </where>
</req>
```

Response #1

The request is successful, and the *PoolQuota* opaque data was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to create the *PoolState* opaque data. The *PoolState* XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolState"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the PoolState opaque data was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #3

A request is made to create the *PoolDynamicQuota* opaque data. The *PoolDynamicQuota* XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolDynamicQuota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQquota name="AggregateLimit">
    <Type>Roll-Over</Type>
    <InstanceId>15678</InstanceId>
    <Priority>4</Priority>
    <InitialTime>135</InitialTime >
    <InitialTotalVolume>2000</InitialTotalVolume>
    <InitialInputVolume>1500</InitialInputVolume>
    <InitialOutputVolume>500</InitialOutputVolume>
    <InitialServiceSpecific>4</InitialServiceSpecific>
    <activationDateTime>32</activationDateTime>
    <expirationDateTime>28</expirationDateTime>
```

```
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
</definition>
]]></cdata></expr>
 </set>
 <where>
   <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
 </where>
</req>
```

Response #3

The request is successful, and the PoolDynamicQuota opaque data was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #4

A request is made to create the *PoolLocation* opaque data. The *PoolLocation* XML blob is supplied whole. *PoolLocation* is NOT a valid opaque data type. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolLocation"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<location>
  <town>Montreal</town>
  <province>Quebec</province>
  <country>Canada</country>
</location>
]]></cdata></expr>
  </set>
  <where>
   <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #4

The request fails. The `error` value indicates the opaque data type is invalid, and the `affected` rows are `0`. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="70015" affected="0"/>
</req>
```

Request #5

A request is made to create the *PoolQuota* opaque data. The *PoolQuota* XML blob is supplied whole. The Pool already has an associated PoolQuota. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolQuota"/><op value="="/><cdata>
```

```
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
  <cid>9223372036854775807</cid>
  <time>3422</time>
  <totalVolume>1000</totalVolume>
  <inputVolume>980</inputVolume>
  <outputVolume>20</outputVolume>
  <serviceSpecific>12</serviceSpecific>
  <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage >
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="500000"/></expr>
 </where>
</req>
```

Response #5

The request fails. The `error` value indicates the PoolQuota already exists, and the `affected` rows are `0`. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="70028" affected="0"/>
</req>
```

Request #6

A request is made to create the *PoolQuota* opaque data. The *PoolQuota* XML blob is supplied whole. The Pool already has an associated PoolQuota. The request includes the *odk* attribute, indicating that the PoolQuota should be updated if it exists already. The request is not required in the response.

```
<req name="insert" resonly="y" odk="yes">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolQuota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
  <cid>9223372036854775807</cid>
  <time>3422</time>
  <totalVolume>1000</totalVolume>
  <inputVolume>980</inputVolume>
  <outputVolume>20</outputVolume>
  <serviceSpecific>12</serviceSpecific>
  <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage >
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="600099"/></expr>
 </where>
</req>
```

Response #6

The request is successful, and the *PoolQuota* opaque data was updated. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Get Opaque Data

This operation retrieves the pool opaque data of the specified *poolOpaqueDataType* for the pool identified by the *poolId* in the request.

The response contains the entire XML blob for the requested pool opaque data.

Note: The *poolOpaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *poolOpaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.

The pool opaque data of the *poolOpaqueDataType* must exist for the pool.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <select>
    <expr><attr name="poolOpaqueDataType"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 108: Request Variable Definitions: Get Opaque Data (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
poolOpaqueDataType	A user defined type/name for the pool opaque data	• PoolQuota • PoolState • PoolDynamicQuota

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>
          <![CDATA[cdataRowValue]]>
      </rv>
    </row>
  </rset>
]
</req>
```

Table 109: Response Variable Definitions: Get Opaque Data (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Opaque Data (Pool)*	
affected	The number of pool opaque data rows returned. A value of "1" is expected for success.	
cdataRowValue	Contents of the XML data "blob"	

Note: The <rset> (row set) element is optional. It is only present if the request was successful. Only a single <row> element is returned, with a single <rv> (row value) element containing an XML CDATA construct containing the requested pool opaque data (i.e. XML blob).

Error Codes

Table 110: Error Codes: Get Opaque Data (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found

Examples

Request #1

A request is made to get the *PoolQuota* opaque data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Pool"/>
  <select>
    <expr><attr name="PoolQuota"/></expr>
  </select>
  <where>
   <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the PoolQuota opaque data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="AggregateLimit">
              <cid>9223372036854775807</cid>
              <time>3422</time>
              <totalVolume>1000</totalVolume>
              <inputVolume>980</inputVolume>
              <outputVolume>20</outputVolume>
              <serviceSpecific>12</serviceSpecific>
              <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
            </quota>
          </usage>]]>
      </rv>
    </row>
```

```
  </rset>
</req>
```

Update Opaque Data

This operation updates the pool opaque data of the specified *poolOpaqueDataType* for the pool identified by the *poolId* in the request.

The pool opaque data is provided in the request within a `<cdata>` construct. The existing pool opaque data is completely replaced by the data supplied in the request.

Note: The opaque data provided in an XML blob is always checked to be valid XML. If the entity is defined as transparent in the SEC, then the XML blob is fully validated against the definition in the SEC. If either validation check fails, then the request is rejected.

Note: The *poolOpaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *poolOpaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="poolOpaqueDataType"/><op value="="/><cdata>
<![CDATA[
cdataFieldValue
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 111: Request Variable Definitions: Update Opaque Data (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295

Variable	Definition	Value
poolOpaqueDataType	A user defined type/name for the pool opaque data	• PoolQuota • PoolState • PoolDynamicQuota
cdataFieldValue	Contents of the XML data "blob"	
poolId	PoolID value of the pool. Numeric value 1-22 digits in length	1-99999999999999999999

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 112: Response Variable Definitions: Update Opaque Data (Pool)

Variable	Defintion	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optionial) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Update Opaque Data (Pool)* for other values.	
affected	The number of pool opaque data rows updated. A value of "1" is expected for success.	

Error Codes

Table 113: Error Codes: Update Opaque Data (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element.
INVALID _XML	Invalid Input XML
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to update the *PoolState* opaque data. The *PoolState* XML blob is supplied whole. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolState"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>3</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the PoolState opaque data is updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Delete Opaque Data

This operation deletes the opaque data of the specified *poolOpaqueDataType* for the pool identified by the *poolId* in the request.

Only one opaque data type can be deleted per request.

Note: The deletion of a non-existent opaque data type (but that is defined in the SEC) is not considered as an error.

Note: The *poolOpaqueDataType* is case-sensitive, and must be entered as specified below else the request will fail.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *poolOpaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="opaqueDataType"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

Table 114: Request Variable Definitions: Delete Opaque Data (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
keyName	A user defined field identified as key for the subscriber	
keyValue	A key value identifying the subscriber	

Variable	Definition	Value
poolOpaqueDataType	A user defined type/name for the opaque data **Note:** The data is deleted by setting an empty field value, and also specifying the attribute `isnull="y"`.	• PoolQuota • PoolState • PoolDynamicQuota
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999

Response

```
<req name="update" [resonly="resonly] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 115: Response Variable Definitions: Delete Opaque Data (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Opaque Data (Pool)* for other values.	
affected	The number of pool opaque data entries deleted. A value of "1" is expected for success.	

Error Codes

Table 116: Error Codes: Delete Opaque Data (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to delete the *PoolDynamicQuota* opaque data. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolDynamicQuota"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the PoolDynamicQuota opaque data is deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to delete the *PoolState* opaque data. *PoolState* is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolState"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

Response #2

The request is successful, because no error is returned if the pool does not have the opaque data type.

```
<req name="update" resonly="y">
  <res error="0" affected="1"></res>
</req>
```

Pool Data Row Commands

A pooled transparent data entity may contain data that is organized in "rows". An example of a row is a specific quota within the PoolQuota entity.

The row commands allow operations (create/retrieve/update/delete) at the row level. The required row is identified in the request by the *RowIdName/RowIdValue*.

Table 117: Summary of Pool Data Row Commands

Command	Description	Key(s)	Command Syntax
Create Row	Insert data row into transparent data of the specified type.	PoolID and Row Identifier	`<req name="insert"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`
Get Row	Retrieve data row from transparent data of the specified type.		`<req name="select"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`
Update Row	Update data row within transparent data of the specified type.		`<req name="update"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`
Delete Row	Delete data row within transparent data of the specified type.		`<req name="delete"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`

Create Row

This operation creates a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. All *fieldNameX* fields specified are set within the row.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for PoolQuota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row already existed called "DayPass", then an attempt to update an existing row called "DAYPASS" would be successful, and two rows called "DayPass" and "DAYPASS" would be present.

Note: If the transparent entity specified in *entityName* does not exist for the pool, it will be created.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *entityName* must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.

Request

Note: This command allows 2 different formats. One with the *poolId* within the `<set>` element, and another with the *poolId* within a `<select>` element.

Format #1

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="PoolID"/><value val="poolId"/></expr>
    <expr><attr name="rowIdName"/><value val="rowIdValue"/></expr>
[
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]
  </set>
</req>
```

Format #2

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="rowIdName"/><value val="rowIdValue"/></expr>
[
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
  </set>
</req>
```

Table 118: Request Variable Definitions: Create Row (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
odk	(Optional) Indicates that the insert request should be converted to an update if the data row for the specified entry already exists	
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the Quota transparent data
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for PoolQuota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
fieldNameX	A user defined field within the data row	
fieldValueX	Corresponding field value assigned to *fieldNameX* **Note:** For multi-value fields, the value can contain a comma separated list of values on a single line. E.g. “a,b,c”.	

Note: Rows that have the same *rowIdName/rowIdValue* are permitted. Where duplicate rows occur, and an additional field is set to define uniqueness (such as `<cid>` in the Quota entity) no validation is performed by UDR to ensure uniqueness. Unique values must be supplied by the provisioning client otherwise operations (such as updating an existing row) may fail if more than one matching row is found.

Note: If the odk="yes" attribute is set (implying that an update be made if the row exists), then if **multiple** rows exist for the specified *rowIdName/rowIdValue*, then the request will fail because it is not known which of the multiple rows to update.

Response

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 119: Response Variable Definitions: Create Row (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Create Row (Pool)* for other values.	
affected	The number of data entities updated. A value of "1" is expected for success.	

Error Codes

Table 120: Error Codes: Create Row (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC

Error Code	Description
OCC_CONSTR_VIOLATION	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INVALID_SOAP_XML	Invalid SOAP XML
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
MULTIPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that matches the given row criteria **Note:** Only returned when the odk="yes" attribute is supplied, and duplicate candidate rows to update are found

Examples

Request #1

A request is made to create a data row in the *PoolQuotaEntity* (PoolQuota) data. The data row identifier field is *Name*, and the value is *Q1*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="name"/><value val="Q1"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
          <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>
```

Response #1

The request is successful, and the data row Q1 was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to create a data row in the *PoolQuotaEntity* (PoolQuota) data. PoolQuota is a valid opaque data type, but the pool does not have this opaque data type. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="300000"/></expr>
    <expr><attr name="name"/><value val="Q2"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
          <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>
```

Response #2

The request is successful, and the data row as well as the PoolQuota entity is created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Get Row

This operation retrieves a data row(s) for the pool identified by the *poolId*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName/instanceFieldValue*.

Note: All data rows that match the requested *rowName/rowValue* and *instanceFieldName/instanceFieldValue* (if specified) are returned.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for PoolQuota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row existed called "DayPass", then an attempt to retrieve a row called "DayPass" would be successful, but an attempt to retrieve a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to retrieve a row with a field with the value "Data" would be successful, but an attempt to retrieve a row with a field with the value "DATA" would fail.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

The transparent entity must exist for the pool.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 121: Request Variable Definitions: Get Row (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the Quota transparent data
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-9999999999999999999999
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for PoolQuota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the PoolQuota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Variable	Definition	Value

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
<
      <rv>
          <![CDATA[cdataRowValue1]]>
      </rv>
|
      <rv null="y"/>
>
    </row>
[
    <row>
      <rv>
          <![CDATA[cdataRowValue2]]>
      </rv>
    </row>
    :
    <row>
      <rv>
          <![CDATA[cdataRowValueN]]>
      </rv>
    </row>
]
  </rset>
]
</req>
```

Table 122: Response Variable Definitions: Get Row (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	

Variable	Definition	Value
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Row (Pool)* for other values.	
affected	The number of data rows returned. A value of "1" is expected for success.	
cdataRowValueN	Contents of the XML data "blob" containing one requested/matching data row	

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful. One `<row>` element is returned per matching row, with a single `<rv>` (row value) element containing an XML CDATA construct containing a single requested data row instance.

Note: If the transparent entity exists, but the row value was not found, then the <rv> (row value) will indicate the row does not exist by containing the value <rv null="y"/>.

Error Codes

Table 123: Error Codes: Get Row (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found

Examples

Request #1

A request is made to get the *Q1* data row from the PoolQuota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the PoolQuota data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
```

```
  <rset>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Q1">
            <cid>9223372036854775807</cid>
            <time>3422</time>
            <totalVolume>1000</totalVolume>
            <inputVolume>980</inputVolume>
            <outputVolume>20</outputVolume>
            <serviceSpecific>12</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

Request #2

A request is made to get the *Weekend* data row from the PoolQuota data. The PoolQuota data contains two rows called *Weekend*. One with `<cid>` of *11223344*, the other with a `<cid>` of *99887766*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and 2 PoolQuota data rows are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Weekend">
            <cid>11223344</cid>
            <time>3422</time>
            <totalVolume>1000</totalVolume>
            <inputVolume>980</inputVolume>
            <outputVolume>20</outputVolume>
            <serviceSpecific>12</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
```

```
        <usage>
          <version>3</version>
          <quota name="Weekend">
            <cid>99887766</cid>
            <time>1232</time>
            <totalVolume>2000</totalVolume>
            <inputVolume>440</inputVolume>
            <outputVolume>8220</outputVolume>
            <serviceSpecific>99</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

Request #3

A request is made to get the *Weekend* data row from the PoolQuota data, with the `<cid>` value of *11223344*. The PoolQuota data contains two rows called *Weekend*. One with `<cid>` of *11223344*, the other with a `<cid>` of *99887766*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the PoolQuota data with a `<cid>` of *11223344* is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Weekend">
            <cid>11223344</cid>
            <time>3422</time>
            <totalVolume>1000</totalVolume>
            <inputVolume>980</inputVolume>
            <outputVolume>20</outputVolume>
            <serviceSpecific>12</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

Request #4

A request is made to get the *LateNight* data row from the PoolQuota data, with the <cid> value of *11223344*. The PoolQuota data contains five rows called *LateNight*. Two with <cid> of *11223344*, one with a <cid> of *99887766*, and one with a <cid> of *55556666*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="LateNight"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #4

The request is successful, and the 2 PoolQuota data rows with a <cid> of *11223344* are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="LateNight">
            <cid>11223344</cid>
            <time>3422</time>
            <totalVolume>1000</totalVolume>
            <inputVolume>980</inputVolume>
            <outputVolume>20</outputVolume>
            <serviceSpecific>12</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="LateNight">
            <cid>11223344</cid>
            <time>1232</time>
            <totalVolume>2000</totalVolume>
            <inputVolume>440</inputVolume>
            <outputVolume>8220</outputVolume>
            <serviceSpecific>99</serviceSpecific>
            <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
          </quota>
        </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

Request #5

A request is made to get the *Weekday* data row in the PoolQuota data. The *Weekday* data row does NOT exist in the PoolQuota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>
```

Response #5

The request is successful, and indicates that the request row does not exist. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

Request #6

A request is made to get the *Weekday* data row in the PoolQuota data. *PoolQuota* is a valid opaque data type, but the pool does not have this opaque data type. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>
```

Response #6

The request fails. The *error* value indicates the opaque data type is not found, and the `affected` rows are 0. The original request is not included.

```
<req name="select" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

Delete Row

This operation deletes a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*.

If more than one row matches the requested *rowName*/*rowValue* and *instanceFieldName*/*instanceFieldValue* (if specified), then all matching rows are deleted.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for PoolQuota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row existed called "DayPass", then an attempt to delete a row called "DayPass" would be successful, but an attempt to delete a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to delete a row with a field with the value "Data" would be successful, but an attempt to delete a row with a field with the value "DATA" would fail.

Note: The deletion of a non-existent data row is not considered as an error.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *entityName* must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.

The transparent entity must exist for the pool.

Request

```
<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 124: Request Variable Definitions: Delete Row (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295

Variable	Definition	Value
entityName	A user defined entity type/name for the transparent data	Value is *QuotaEntity* for the PoolQuota transparent data
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the PoolQuota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="delete" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 125: Response Variable Definitions: Delete Row (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see	

Variable	Definition	Value
	Error Codes: Delete Row (Pool) for other values.	
affected	The number of data rows deleted. A value of "1" indicates the row(s) existed and were deleted. A value of "0" indicates the row did not exist.	

Error Codes

Table 126: Error Codes: Delete Row (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found

Examples

Request #1

A request is made to delete the *Q1* data row from the PoolQuota data. The *Q1* data row exists in the PoolQuota data, and is there is only one row called *Q1*. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the data row in the PoolQuota data is deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to update the *Weekend* data row in the PoolQuota data. The *Weekend* data row does NOT exist in the PoolQuota data. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
```

```
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

Response #2

The request is successful, because no error is returned if the data row is not present. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #3

A request is made to update the *Q3* data row in the PoolQuota data. The PoolQuota data contains two rows called *Q3*. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the data row in the PoolQuota data was deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #4

A request is made to delete the *Q4* data row in the PoolQuota data with the `<cid>` *11223344*. The PoolQuota data contains two rows called *Q4*. One with `<cid>` *11223344* the other with `<cid>` *99887766*. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q4"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #4

The request is successful, and the data row in the PoolQuota data is deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Pool Data Row Field Commands

A pooled transparent data entity may contain data that is organized in "rows". An example of a row is a specific quota within the PoolQuota entity.

The row/field commands allow operations (retrieve/update/delete) at the row/field level. The required row is identified in the request by the *rowIdName/rowIdValue*, and the field is identified by the *fieldName*.

Table 127: Summary of Pool Data Row Field Commands

Command	Description	Key(s)	Command Syntax
Get Row Field	Retrieve value(s) for the specified field(s).	PoolID and Row Identifier and Field name	`<req name="select"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`
Update Row Field	Update field(s) to the specified value(s).		`<req name="update"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`
Delete Row Field	Delete all value(s) for the specified field(s).		`<req name="delete"> <ent name="entityName"/> … <expr> <attr name="rowIdName"> <value val="rowIdValue"> </expr> …`

Get Row Field

This operation retrieves a fields(s) within a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName/instanceFieldValue*. The field name(s) are specified in *fieldNameX*.

All data rows that match the requested *rowName/rowValue* and *instanceFieldName/instanceFieldValue* (if specified) are returned.

Note: If the specified row does not exist, the request will fail. If the specified row exists, but the field does not exist, this is not treated as an error, and no row/field data is deleted.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for PoolQuota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row already existed called "DayPass", then an attempt to get a field in a row called "DayPass" would be successful, but an attempt to get a field in a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to delete a row with a field with the value "Data" would be successful, but an attempt to delete a row with a field with the value "DATA" would fail.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *entityName* must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.

A data row with the given identifier/instance within the transparent data should exist for the subscriber.

The field name(s) specified must be valid fields for the Entity as defined in the SEC.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <select>
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
[
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 128: Request Variable Definitions: Get Row Field (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
entityName	A user defined entity type/name for the transparent data	Value is *PoolQuotaEntity* for the PoolQuota transparent data
fieldNameX	A user defined field within the data row	
fieldValueX	Corresponding field value assigned to *fieldNameX*	
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for PoolQuota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the PoolQuota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv>
[
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv>
      :
```

```
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv>
]
    </row>
[
    <row>
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv>
[
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv>
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv>
]
    </row>
    :
    <row>
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv>
[
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv>
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv>
]
    </row>
]
  </rset>
]
</req>
```

Table 129: Response Variable Definitions: Get Row Field (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Row Field (Pool)* for other values.	
affected	The number of data rows from which data is returned. A value of "1" is expected for success.	
rowValue	The value of the requested field. **Note:** For multi-value fields, the value will contain a comma	

Variable	Definition	Value
	separated list of values on a single line. E.g. "a,b,c"	

Note: The <rset> (row set) element is optional. It is only present if the request was successful. One <row> element is returned per matching row. One <rv> (row value) element exists for every *fieldNameX* supplied in the original request. The <rv> elements are ordered the same as the *fieldNameX* fields were specified in the original request. If the field is valid, but not present in the entity, this is indicated with <rv null="y">. If the field is present, but has an empty value, this is indicated with <rv></rv>.

Error Codes

Table 130: Error Codes: Get Row Field (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found
ROW_NOT_FOUND	Data row specified is not found

Examples

Request #1

A request is made to get the *inputVolume* field in the *Q1* data row of the PoolQuota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <select>
    <expr><attr name="inputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the requested field value *980* is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>980</rv>
```

```
    </row>
  </rset>
</req>
```

Request #2

A request is made to get the *outputVolume* and *cid* fields in the *Q2* data row of the PoolQuota data. The PoolQuota data contains two rows called *Q2*. One with `<cid>` of *11223344*, the other with a `<cid>` of *99887766*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></expr>
    <expr><attr name="cid"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q2"/></expr>
  </where>
</req>
```

Response #2

The request is successful, and the requested field values are returned from each row. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>220</rv>
      <rv>11223344</rv>
    </row>
    <row>
      <rv>1050</rv>
      <rv>99887766</rv>
    </row>
  </rset>
</req>
```

Request #3

A request is made to get the *outputVolume* field in the *Q3* data row of the PoolQuota data, with the `<cid>` value of *11223344*. The PoolQuota data contains two rows called *Q3*. One with `<cid>` of *11223344*, the other with a `<cid>` of *99887766*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the requested field value *4000* is returned. The original request is not included.

```
<req name="select">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>4000</rv>
    </row>
  </rset>
</req>
```

Update Row Field

This operation updates a field(s) within a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName/instanceFieldValue*. The field name(s) are specified in *fieldNameX*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. If the specified field(s) are valid, but do not currently exist, they will be created.

If more than one row matches the requested *rowName/rowValue* and *instanceFieldName/instanceFieldValue* (if specified), then the update request will fail.

Note: If the specified row does not exist, the request will fail.

Note: If the requested field(s) exists, are valid, but not currently present, they will be created.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: he *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for PoolQuota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row already existed called "DayPass", then an attempt to update a field in a row called "DayPass" would be successful, but an attempt to update a field in a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to delete a row with a field with the value "Data" would be successful, but an attempt to delete a row with a field with the value "DATA" would fail.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

A data row with the given identifier/instance within the transparent data should exist for the subscriber.

The field name(s) specified must be valid fields for the Entity as defined in the SEC.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
[
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

Table 131: Request Variable Definitions: Update Row Field (Pool)

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
entityName	A user defined entity type/name for the transparent data	Value is *PoolQuotaEntity* for the PoolQuota transparent data
fieldNameX	A user defined field within the data row	
fieldValueX	Corresponding key field value assigned to *fieldNameX* **Note:** For multi-value fields, the value can contain a comma separated list of values on a single line. E.g. "a,b,c".	
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data

Variable	Definition	Value
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the PoolQuota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 132: Response Variable Definitions: Update Row Field (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Update Row Field (Pool)* or other values.	
affected	The number of data rows updated. A value of "1" is expected for success.	

Error Codes

Table 133: Error Codes: Update Row Field (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_VAL_INVALID	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FIELD_NOT_UPDATABLE	Field Cannot be Updated. The field is defined in the SEC as not be updatable
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found
ROW_NOT_FOUND	Data row specified is not found
MULITPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria

Examples

Request #1

A request is made to update the *inputVolume* field in the *Q1* data row of the PoolQuota data. The *Q1* data row exists in the PoolQuota data, and is there is only one row called *Q1*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the field in the data row in the PoolQuota data was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to update the *cid* field in the *Q1* data row in the PoolQuota data. The *Q1* data row exists in the PoolQuota data, and is there is only one row called *Q1*. The *cid* field is not allowed to be updated. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="cid"/><value val="11223344"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

Response #2

The request fails. The `error` value indicates the `cid` field cannot be updated, and the affected rows are `0`. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70016" affected="0"/>
</req>
```

Request #3

A request is made to update the *outputVolume* field in the *Q6* data row of the PoolQuota data. The *Q6* data row exists in the PoolQuota data, but there are two rows called *Q3*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q6"/></expr>
  </where>
</req
```

Response #3

The request fails because there was more than one row called *Q6*. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

Delete Row Field

This operation deletes a field(s) within a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowName*, and the row identifier value is specified in *rowValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field name(s) are specified in *fieldNameX*.

If more than one row matches the requested *rowName/rowValue* and *instanceFieldName/instanceFieldValue* (if specified), then the delete request will fail.

Note: If the specified row does not exist, the request will fail. If the specified row exists, but the field does not exist, this is not treated as an error, and no row/field data is deleted.

Note: The *entityName* is case-sensitive, and must be entered as specified below else the request will fail.

Note: The *rowIdName* is case-sensitive, and must be entered as defined in the SEC for the entity else the request will fail (for example for PoolQuota, the element <quota **name**="AggregateLimit"> contains the attribute called "name").

Note: The *rowIdValue* is case-sensitive. If a row existed called "DayPass", then an attempt to delete a field in a row called "DayPass" would be successful, but an attempt to delete a field in a row called "DAYPASS" would fail.

Note: The *instanceFieldName* is not case-sensitive.

Note: The *instanceFieldValue* is case-sensitive. If a field contained the value "Data", then an attempt to delete a field in a row with a field with the value "Data" would be successful, but an attempt to delete a field in a row with a field with the value "DATA" would fail.

Note: A request to delete a field(s) cannot be mixed with a request to update a field, otherwise the request will fail. A request to delete a field(s) must only contain field(s) being deleted.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *entityName* must reference a valid Entity in the Interface Entity Map table in the SEC.

At least one data row with the given identifier/instance within the transparent data should exist for the subscriber.

The field name(s) specified must be valid fields for the Entity as defined in the SEC.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><op value="="/><value val=""
isnull="y"/></expr>
[
    <expr><attr name="fieldName2"/><op value="="/><value val=""
isnull="y"/></expr>
              :
    <expr><attr name="fieldNameN"/><op value="="/><value val=""
isnull="y"/></expr>
]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
   <expr><attr name="rowIdName"/><op value="="/><value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
```

```
  </where>
</req>
```

Table 134: Request Variable Definitions: Delete Row Field

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
entityName	A user defined entity type/name for the transparent data	Value is *PoolQuotaEntity* for the PoolQuota transparent data
fieldNameX	A user defined field within the data row	
fieldValueX	Corresponding key field value assigned to *fieldNameX* **Note:** For multi-value fields, the value can contain a comma separated list of values on a single line. E.g. "a,b,c".	
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
rowIdName	Name of the XML attribute that identifies the row within the data blob	Value is *name* for Quota transparent data
rowIdValue	The row name value that identifies the row within the data blob	
instanceFieldName	A user defined field within the data row that is used to define a unique row instance	Value is *cid* for the Quota transparent data
instanceFieldValue	Corresponding field value assigned to *instanceFieldName*	

Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 135: Response Variable Definitions: Delete Row Field (Pool)

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Delete Row Field (Pool)* for other values.	
affected	The number of data entities updated. A value of "1" indicates the row existed and the field was deleted. A value of "0" indicates the field did not exist.	

Error Codes

Table 136: Error Codes: Delete Row Field (Pool)

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
REG_DATA_NOT_FOUND	Register Data Not Found
ROW_NOT_FOUND	Data row specified is not found

Error Code	Description
MULITPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria

Examples

Request #1

A request is made to delete the *inputVolume* field in the *Q1* data row of the PoolQuota data. The *Q1* data row exists in the PoolQuota data, and is there is only one row called *Q1*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

Response #1

The request is successful, and the field in the data row was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to delete the *outputVolume* field in the *Q3* data row of the PoolQuota data. The PoolQuota data contains two rows called *Q3*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
  </where>
</req>
```

Response #2

The request fails because there are two PoolQuota rows called *Q3*. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

Request #3

A request is made to update delete the *outputVolume* field in the *Q4* data row of the PoolQuota data with the `<cid>` 11223344. The Q4 data row exists in the PoolQuota data, and is there are two rows called *Q4*, one with `<cid>`*11223344* and one with `<cid>`*99887766*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q4"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response #3

The request is successful, and the *outputVolume* field in the *Q4* data row in the PoolQuota data was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Additional Pool Commands

Table 137: Summary of Additional Pool Commands

Command	Description	Key(s)	Command Syntax
Add Member to Pool	Add subscriber to a Pool	PoolID and (MSISDN, IMSI, NAI	`<req name="operation"><oper name="AddPoolMember">`

Command	Description	Key(s)	Command Syntax
Remove Member from Pool	Remove subscriber from a Pool	or AccountId)	`<req name="operation"> <oper name="DelPoolMember">`
Get Pool Members	Retrieve pool member subscribers by PoolID	PoolID	`<req name="operation"> <oper name="GetPoolMember">`
Get Pool by Member (key)	Retrieve PoolID for specified member subscriber	MSISDN, IMSI, NAI or AccountId	`<req name="operation"> <oper name="GetPoolId">`

Add Member to Pool

This operation adds one or more subscribers to a pool.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

Separate subscribers with the keys of the *keyNameX*/*keyValueX* supplied must exist.

Each subscriber must not already be a member of a pool.

The pool must have less than the maximum number of member subscribers allowed.

Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="AddPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="poolId"/></expr>
    <expr><param name="subKeyName1"/><op value="="/>
          <value val="subKeyValue1"/></expr>
[
    <expr><param name="subKeyName2"/><op value="="/>
          <value val="subKeyValue2"/></expr>
            :
    <expr><param name="subKeyName10"/><op value="="/>
          <value val="subKeyValue10"/></expr>
]
  </oper>
</req>
```

Table 138: Request Variable Definitions: Add Member to Pool

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including	• y — only provide the result, do NOT include the original request

Variable	Definition	Value
	the original request in the response	• n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999
subKeyNameX	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
subKeyValueX	Corresponding field value assigned to *KeyNameX*	

Note: Up to 10 subscribers can be added in one request.

Note: The number of subscribers being added must not cause the number of members in the pool to exceed the maximum allowed value, else the request will fail.

Note: If any subscriber specified is currently a member of a pool, the request will fail.

Note: The `ent` attribute is optional, and if supplied, the value is not validated.

Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 139: Response Variable Definitions: Add Member to Pool

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	

Variable	Definition	Value
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes* for other values.	
affected	The number of subscribers added to the pool. A value of "1" or more is expected for success.	

Error Codes

Table 140: Error Codes: Add Member to Pool

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
ALREADY_POOL_MEMBER	Already a Pool Member. The subscriber is already a member of a pool
MAX_POOL_MEMBERS	Pool Member List Max Limit Reached
POOL_NOT_FOUND	Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist

Examples

Request #1

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is not already a member of a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="100000"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="33123654862"/></expr>
  </oper>
</req>
```

Response #1

The request is successful, and the subscriber is added to the pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to add a subscriber to a pool. The pool exists, but the subscriber does not. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="200002"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="15141234567"/></expr>
  </oper>
</req>
```

Response #2

The request fails. The `error` value indicates the that subscriber does not exist, and the `affected` rows are `0`. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70019" affected="0"/>
</req>
```

Request #3

A request is made to add a subscriber to a pool. The subscriber exists, but the pool does not. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="300003"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="33123654862"/></expr>
  </oper>
</req>
```

Response #3

The request fails. The `error` value indicates the that pool does not exist, and the `affected` rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70036" affected="0"/>
</req>
```

Request #4

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is already a member of a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="200000"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="33123654862"/></expr>
  </oper>
</req>
```

Response #4

The request fails. The `error` value indicates the subscriber is already a member of a pool, and the `affected` rows are `0`. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70023" affected="0"/>
</req>
```

Request #5

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is not a member of a pool. The pool has the maximum number of members allowed. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="400000"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="33123654862"/></expr>
  </oper>
</req>
```

Response #5

The request fails. The `error` value indicates the pool has the maximum number of members allowed, and the `affected` rows are `0`. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70024" affected="0"/>
</req>
```

Request #6

A request is made to add 3 subscribers to a pool. The pool and all subscribers exist. No subscribers are already a member of a pool. The pool has the maximum number of members allowed. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="800000"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="15145551234"/></expr>
    <expr><param name="IMSI" /><op value="="/>
          <value val="302370123456789"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="14162221234"/>
  </oper>
</req>
```

Response #6

The request is successful, and the 3 subscribers are added to the pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="3"/>
</req>
```

Remove Member from Pool

This operation removes one or more subscribers from a pool.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

Separate subscribers with the keys of the *keyNameX*/*keyValueX* supplied must exist.

Each subscriber must be a member of the specified pool.

Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="DelPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="poolId"/></expr>
    <expr><param name="subKeyName1"/><op value="="/>
          <value val="subKeyValue1"/></expr>
[
    <expr><param name="subKeyName2"/><op value="="/>
          <value val="subKeyValue2"/></expr>
             :
    <expr><param name="subKeyName10"/><op value="="/>
         <value val="subKeyValue10"/></expr>
]
  </oper>
</req>
```

Table 141: Request Variable Definitions: Remove Member from Pool

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999
subKeyNameX	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
subKeyValueX	Corresponding key field value assigned to *keyName*	

Note: Up to 10 subscribers can be removed in one request.

Note: If any subscriber specified is not a member of the pool, the request will fail.

Note: The `ent` attribute is optional, and if supplied, the value is not validated.

Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

Table 142: Response Variable Definitions: Remove Member from Pool

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes* for other values.	
affected	The number of subscribers removed from the pool. A value of "1" or more is expected for success.	

Error Codes

Table 143: Error Codes: Remove Member from Pool

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
NOT_POOL_MEMBER	Not a Pool Member

Error Code	Description
POOL_NOT_FOUND	Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist

Examples

Request #1

A request is made to remove a subscriber from a pool. Both the pool and the subscriber exist. The subscriber is a member of the pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="DelPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="100000"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="33123654862"/></expr>
  </oper>
</req>
```

Response #1

The request is successful, and the subscriber is removed from the pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

Request #2

A request is made to remove a subscriber from a pool. Both the pool and the subscriber exist. The subscriber is NOT a member of the pool. The request is not required in the response.

```
<req name="operation">
  <oper name="DelPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="200000"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="33123654862"/></expr>
  </oper>
</req>
```

Response #2

The request fails. The `error` value indicates the subscriber is not a member of the pool, and the `affected` rows are `0`. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70025" affected="0"/>
</req>
```

Request #3

A request is made to remove 3 subscribers from a pool. The pool and all subscribers exist. All subscribers are a member of the pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="DelPoolMember">
    <expr><param name="PoolID" /><op value="="/><value val="800000"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="15145551234"/></expr>
    <expr><param name="IMSI" /><op value="="/>
          <value val="302370123456789"/></expr>
    <expr><param name="MSISDN" /><op value="="/>
          <value val="14162221234"/></expr>
  </oper>
</req>
```

Response #3

The request is successful, and the 3 subscribers are removed from the pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="3"/>
</req>
```

Get Pool Members

This operation gets the list of subscriber members of a pool by *poolId*.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="GetPoolMembers">
    <expr><param name="PoolID" /><op value="="/><value val="poolId"/></expr>
  </oper>
</req>
```

Table 144: Request Variable Definitions: Get Pool Members

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-999999999999999999999

Note: The `ent` attribute is optional, and if supplied, the value is not validated.

Response

```
<req name="operation" [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected">
[
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <members>
[
            <member>
              <id><name>keyName1</name><value>keyValue1</value></id>
[
              <id><name>keyName2</name><value>keyValue2</value></id>
              :
              <id><name>keyNameN</name><value>keyValueN</value></id>
]
            </member>
]
[
            <member>
              <id><name>keyName1</name><value>keyValue1</value></id>
[
              <id><name>keyName2</name><value>keyValue2</value></id>
              :
              <id><name>keyNameN</name><value>keyValueN</value></id>
]
            </member>
            :
            <member>
              <id><name>keyName1</name><value>keyValue1</value></id>
[
              <id><name>keyName2</name><value>keyValue2</value></id>
              :
              <id><name>keyNameN</name><value>keyValueN</value></id>
]
            </member>
]
          </members>]]>
      </rv>
    </row>
  </rset>
]
</req>
```

Table 145: Response Variable Definitions: Get Pool Members

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get Pool Members* for other values.	
affected	The number of subscriber Profiles from which data is returned. A value of "1" is expected for success.	
keyNameX	A key field for the member subscriber	• IMSI • MSISDN • NAI • AccountId
keyValueX	Corresponding key field value assigned to *keyNameX*	

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful and the subscriber is a member of a pool. Only a single `<row>` element is returned with one `<rv>` (row value) element which contains the PoolID of the subscriber.

Note: The `<member>` element is optional. There can be zero, one or many `<member>` elements. It is only present if the pool has member subscribers. One instance is present for every subscriber that is a member of the pool. A `<member>` element contains details about a single subscriber, containing all known user identities for that subscriber, one user identity per `<id>` element. There can be one or many `<id>` elements per `<member>` element.

Note: The format of this response can be returned in a legacy SPR compatible mode if UDR is configured to do so. See *Legacy SPR Compatibility Mode* for more details.

Error Codes

Table 146: Error Codes: Get Pool Members

Error Code	Description
POOL_NOT_FOUND	Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist.

Examples

Request #1

A request is made to get the list of subscribers for a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolMembers">
    <expr><param name="PoolID" /><op value="="/><value val="100000"/></expr>
  </oper>
</req>
```

Response #1

The request is successful, and the 3 member subscribers are returned. The original request is not included.

```
<req name="operation">
  <res error="0" affected="1">
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <members>
            <member>
              <id><name>IMSI</name><value>311480100000001</value></id>
              <id><name>IMSI</name><value>311480100532432</value></id>
              <id><name>NAI</name><value>dad@operator.com</value></id>
            </member>
            <member>
              <id><name>MSISDN</name><value>380561234777</value></id>
              <id><name>IMSI</name><value>311480100000999</value></id>
            </member>
            <member>
              <id><name>NAI</name><value>joe@wireless.com</value></id>
              <id><name>NAI</name><value>p12321@mynet.com</value></id>
            </member>
          </members>]]>
      </rv>
    </row>
    </rset>
</req>
```

Request #2

A request is made to get the list of subscribers for a pool. The pool exists, but has no member subscribers. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolMembers">
```

```
    <expr><param name="PoolID" /><op value="="/><value val="200000"/></expr>
  </oper>
</req>
```

Response #2

The request is successful, and no member subscribers are returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="0"/>
</req>
```

Request #3

A request is made to get the list of subscribers for a pool. The pool does not exist. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolMembers">
    <expr><param name="PoolID" /><op value="="/><value val="300000"/></expr>
  </oper>
</req>
```

Response #3

The request fails. The `error` value indicates the that pool was not found, and the `affected` rows are `0`. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70036" affected="0"/>
</req>
```

Get PoolID

This operation gets the PoolID related to a subscriber, based on the given user identity of the subscriber.

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist.

The subscriber must be a member of a pool.

Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="GetPoolId">
    <expr><param name="keyName"/><op value="="/>
          <value val="keyValue"/></expr>
  </oper>
</req>
```

Table 147: Request Variable Definitions: Get poolId

Variable	Definition	Value
resonly	(Optional) Indicates whether the response should consist of the result only, without including the original request in the response	• y — only provide the result, do NOT include the original request • n — include the original request in the response (default)
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
keyName	A key field within the subscriber Profile	• IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to *keyName*	

Note: The `ent` attribute is optional, and if supplied, the value is not validated.

Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>poolId</rv>
    </row>
  </rset>
]
</req>
```

Table 148: Response Variable Definitions: Get poolId

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"` attribute is set in the original request.	A string with 1 to 4096 characters

Variable	Definition	Value
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see *Error Codes: Get poolId* for other values.	
affected	The number of subscriber Profiles from which data is returned. A value of "1" is expected for success.	
poolId	PoolID value of the pool the subscriber is a member of. Numeric value, 1-22 digits in length	1-999999999999999999999

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful and the subscriber is a member of a pool. Only a single `<row>` element is returned with one `<rv>` (row value) element which contains the PoolID of the subscriber.

Error Codes

Table 149: Error Codes: Get poolId

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found

Examples

Request #1

A request is made to get the PoolID for a subscriber. The subscriber is a member of a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolId">
    <expr><param name="MSISDN" /><op value="="/>
          <value val="33123654862"/></expr>
  </oper>
</req>
```

Response #1

The request is successful, and the PoolID value is returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>100000</rv>
    </row>
  </rset>
</req>
```

Request #2

A request is made to get the PoolID for a subscriber. The subscriber exists, but is NOT a member of a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolId">
    <expr><param name="MSISDN" /><op value="="/>
          <value val="15141234567"/></expr>
  </oper>
</req>
```

Response #2

The request is successful, but the subscriber is not a member of a pool. No result set data is returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="0"/>
</req>
```

Request #3

A request is made to get the PoolID for a subscriber. The subscriber does NOT exist. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolId">
    <expr><param name="MSISDN" /><op value="="/>
          <value val="15145556789"/></expr>
  </oper>
</req>
```

Response #3

The request fails. The *error* value indicates that the subscriber does not exist, and the . The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70019" affected="0"/>
</req>
```

Appendix A

Error Codes

Topics:

This section describes the SOAP interface error codes that are returned by the XDS/SOAP Server.

Error Codes

SOAP error codes are returned by the XDS/SOAP Server in the `error` attribute parameter of the `<res>` message (see *Response*). 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 *Error Codes*.

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 configuring the entities/fields in the SEC etc.

Table 150: SOAP Interface Error Codes

Error Code	Value	Type	Description
NOT_PROCESSED	1	P	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.
INTF_ENTY_NOT_FOUND	70000	P*	Interface Entity Not Found
ENTY_DEF_NOT_FOUND	70002	P	Entity Definition Not Found
VER_BFS_NOT_FOUND	70003	P	Versioned Base Field Set for the Transparent Entity Not Found
NON_VER_BFS_NOT_FOUND	70004	P	Non Versioned Base Field Set for the Transparent Entity Not Found
MULT_VER_TAGS_FOUND	70005	P	Multiple Version Tags Found
FIELD_VAL_INVALID	70006	P*	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OCC_CONSTR_VIOLATION	70007	P*	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
INVAL_REPEATABLE_ELEM	70008	P	Invalid Repeatable Element
INVALID _XML	70009	P	Invalid Input XML
FLD_SET_NOT_FOUND	70010	P	Field Set Not Found
FLD_SET_EXISTS	70011	P	Field Set Already Exists
FIELD_NOT_FOUND	70012	P	Field Not Found
FIELD_EXISTS	70013	P	Field Already Exists
FLD_SET_UNDEFINED	70014	P	Field Set Not Defined

Error Code	Value	Type	Description
FIELD_UNDEFINED	70015	P*	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FIELD_NOT_UPDATABLE	70016	P*	Field Cannot be Updated. The field is defined in the SEC as not be updatable
ENT_CANNOT_RESET	70017	P*	Entity Cannot be Reset. The reset command cannot be used on the requested entity
DB_OPER_FAILED	70018	P	Database Operation Failed
KEY_NOT_FOUND	70019	P	Key Not Found. A subscriber/pool with the given key cannot be found
KEY_EXISTS	70020	P	Key Already Exists. A subscriber/pool already exists with the given key
SUB_IN_POOL	70021	P	Subscriber is Pool Member. The subscriber is a member of a pool. A subscriber cannot be deleted if they are a pool member
HAS_POOL_MEMBERS	70022	P	Has Pool Members. A pool cannot be deleted when it has member subscribers
ALREADY_POOL_MEMBER	70023	P	Already a Pool Member. The subscriber is already a member of a pool
MAX_POOL_MEMBERS	70024	P	Pool Member List Max Limit Reached
NOT_POOL_MEMBER	70025	P	Not A Pool Member
OPER_NOT_ALLOWED	70026	P	Operation Not Allowed
REG_DATA_NOT_FOUND	70027	P	Register Data Not Found
REG_EXISTS	70028	P	Register Already Exists
UNEXPECTED_ERROR	70029	P	Un-Expected Error
INV_SOAP_XML	70030	P	Invalid SOAP XML
SERVICE_UNAVAILABLE	70031	T	Service is unavailable. Provisioning has been disabled
ROW_NOT_FOUND	70032	P	Data row specified is not found
VALUE_EXISTS	70033	P	List value added already exists
FLD_NOT_MULTI	70034	P	Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value
MULTIPLE_ROWS_FOUND	70035	P	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
POOL_NOT_FOUND	70036	P	Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist

Error Code	Value	Type	Description
INVALID_KEY_VALUE	70037	P	The key value supplied is invalid, due to invalid characters/format etc.
DB_RETRY_EXHAUSTED	70038	T	Data could not be committed to database as the total number of retries to commit database transactions exhausted. **Note:** The client shall retry the command again
DURABILITY_DEGRADED	70039	T	Data could not be committed as Durability is degraded. **Note:** The client shall retry the command again
DURABILTY_TIMEOUT	70040	T	Data could not be made durable within the configured Durability Timeout. **Note:** The client shall retry the command again to get the data sent in the failed request to verify that it was stored by last request
TOO_BIG_MESSAGE	70041	P*	The provisioning request size exceeded the maximum allowed size
MEMORY_FULL	70042	P	Free system memory is low. Request cannot be performed

Appendix B

SOAP Interface System Variables

Topics:

This section describes the SOAP interfaces that have a set of system variables that affect the operation as it runs.

SOAP Interface System Variables

The SOAP interfaces have a set of system variables that affect its operation as it runs. SOAP Interface System variables (see *Table 151: SOAP Interface System Variables*) can be set via the UDR GUI and can be changed at runtime to effect dynamic server reconfiguration.

Table 151: SOAP Interface System Variables

Parameter	Description
SOAP Interface Port	SOAP/XML Interface TCP Listening Port. **Note:** Changes to the TCP listening port do not take effect until the 'xsas' process is restarted. Also, you must specify a different port than the REST interface. DEFAULT = 62001; RANGE = 0-65535
SOAP Interface Idle Timeout	The maximum time (in seconds) that an open SOAP/XML connection will remain active without a request being sent, before the connection is dropped. DEFAULT = 1200; RANGE = 1-86400
Maximum SOAP Connections	Maximum number of simultaneous SOAP/XML Interface client connections. **Note:** Changes to the Maximum SOAP Connections option do not take effect until the 'xsas' process is restarted. DEFAULT = 100; RANGE = 1-100
Allow SOAP Provisioning Connections	Whether or not to allow incoming provisioning connections on SOAP/XML Interface. DEFAULT = CHECKED
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
Compatibility Mode*	Indicates whether backwards compatibility is enabled. NOTE: Change to Compatibility Mode may cause the existing provisioning connections to be dropped. DEFAULT = R10+

Parameter	Description
Maximum Requests in SOAP <tx> XML	The maximum number of requests in a single SOAP tx transaction. DEFAULT = 12; RANGE = 1-50

Note: Parameters labeled with a "*" are existing system variables defined and used by other components of UDR.

Appendix C

Legacy SPR Compatibility Mode

Topics:

UDR can be configured to run in compatibility mode with the legacy Subscriber Provisioning Repository (SPR).

When the `Compatibility Mode` system option (see *SOAP Interface System Variables*), which is configurable by the UDR GUI [T8], is set to "R10+" then UDR will behave as described in the main body of this document. When `Compatiblity Mode` is set to "R9.x", then the differences contained in this appendix will apply.

Enabling this configuration option results in the format of some request/responses being different to the default UDR behavior. This appendix lists the different request/responses that enabling the option applies to.

Get Pool Members Response Format

UDR returns the list of pool members in the format using a <members> XML element, as returned by the REST interface.

When configured in legacy SPR mode, UDR returns the list of pool members in the format as described below.

Response:

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>publicIdentity</rv> | <rv></rv>
      <rv>MSISDN1[,MSISDN2[MSISDN3]]</rv> | <rv></rv>
      <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <rv></rv>
      <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv></rv>
      <rv>accountId</rv> | <rv></rv>
    </row>
]
[
    <row>
      <rv>publicIdentity</rv> | <rv></rv>
      <rv>MSISDN1[,MSISDN2[MSISDN3]]</rv> | <rv></rv>
      <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <rv></rv>
      <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv></rv>
      <rv>accountId</rv> | <rv></rv>
    </row>
     :
    <row>
      <rv>publicIdentity</rv> | <rv></rv>
      <rv>MSISDN1[,MSISDN2[MSISDN3]]</rv> | <rv></rv>
      <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <rv></rv>
      <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv></rv>
      <rv>accountId</rv> | <rv></rv>
    </row>
]
  </rset>
]
</req>
```

Table 152: Response Variable Definitions: Get Pool Members

Variable	Definition	Value
originalXMLRequest	(Optional) The text of the original XML request that was sent. **Note:** This is always present unless the `resonly="y"`	A string with 1 to 4096 characters

Variable	Definition	Value
	attribute is set in the original request	
resonly	(Optional) The *resonly* value from the original XML request, if supplied	
id	(Optional) The *id* value from the original XML request, if supplied	
error	Error code indicating outcome of request. "0" means success, see below for other values	
affected	The number of pools from which data is returned. A value of "1" or more is expected	
publicIdentity	The internal public identity value for the subscriber. This field is not used, and the no value will be present	
MSISDNX	Comma separated list of (up to 3) MSISDN values corresponding to subscriber in the pool. No values will be present if an MSISDN is not provisioned for the subscriber	A string with 8 to 15 digits (if value is set)
IMSIX	Comma separated list of (up to 3) IMSI values corresponding to subscriber in the pool. No values will be present if an IMSI is not provisioned for the subscriber	A string with 10 to 15 digits (if value is set)
NAIX	Comma separated list of (up to 3) NAI values corresponding to subscriber in the pool. No values will be present if an NAI is not provisioned for the subscriber	A string with 1 to 255 characters (if value is set) **Note:** NAI is in format "user@domain"
accountId	AccountId corresponding to subscriber in the pool. This value will not be present if an AccountId is not provisioned for the subscriber	A string with 1 to 255 characters (if value is set)

Note: The `<rset>` (row set) element is optional. It is only present if the request was successful. One subscriber is returned per `<row>` element returned, each always containing five `<rv>` (row value) elements.

Legacy SPR SOAP Request Format

The legacy SPR uses a different SOAP request format as described below. Requests can be sent using the same format as done for the legacy SPR.

Note: The `<soapenv:Header>` element in the request is now optional and can be omitted. It can be provided, but will be ignored. Authentication is no longer performed using the `Username/Passwd` in UDR.

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

  <soapenv:Header>
    <ns1:UserName
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="0"
    xsi:type="soapenc:string"
    xmlns:ns1="blueslice.com"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">UserName
    </ns1:UserName>

    <ns2:Passwd
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="0"
    xsi:type="soapenc:string"
    xmlns:ns1="blueslice.com"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">Password
    </ns2:Passwd>
  </soapenv:Header>

  <soapenv:Body>

    <processTransaction xmlns="http://webservice.blueslice.com">

      <![CDATA[REQUEST]]>

    </processTransaction>
```

Figure 4: Legacy SPR SOAP Request Format

Legacy SPR SOAP Response Format

When UDR is configured in legacy SPR mode, even when SOAP requests are sent using the SOAP request format as specified in *Figure 4: Legacy SPR SOAP Request Format*, the SOAP response format is different to the format that the legacy SPR returns. Based on the implementation of UDR, this is unavoidable.

The response should not cause a provisioning client that uses a native SOAP interface any issues (for example, one that uses the WSDL file), as it is still a valid SOAP response. But, if any provisioning

clients have been implemented to look for specific XML elements (such as <soapenv:Envelope> instead of <SOAP-ENV:Envelope>), this problem may arise.

The format of the SOAP response returned by UDR is listed below in *Figure 5: Legacy SPR SOAP Response Format*.

Note: The <SOAP-ENV:Header> is only returned if a <SOAP-ENV:Header> is included in the request. In UDR, the <SOAP-ENV:Header> is optional in the request and is ignored.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://webservice.blueslice.com">

  <SOAP-ENV:Header/>

<
  <SOAP-ENV:Body>

    <ns1:message error="ErrorCode">

      <![CDATA[RESPONSE]]>

    </ns1:message>

  </SOAP-ENV:Body>
|
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <SOAP-ENV:Fault>
    ...
    </SOAP-ENV:Fault>

  </SOAP-ENV:Body>
>

</SOAP-ENV:Envelope>
```

Figure 5: Legacy SPR SOAP Response Format

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://webservice.blueslice.com">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:message error="0">
      <![CDATA[<req name="insert" resonly="y">
                 <res affected="1" error="0"/>
               </req>]]>
    </ns1:message>
```

```
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6: Example of Successful SOAP Response in Legacy SPR Mode

Glossary

A

ACID	Atomicity, Consistency, Isolation and Durability

B

blob	binary large object A collection of binary data stored as a single entity in the Subscription Profile Repository.

C

CPS	Customer Provisioning System

E

ESPR	Enhanced Subscriber Profile Repository - Oracle Communications' database system that provides the storage and management of subscriber policy control data for PCRF nodes.

I

IMSI	International Mobile Subscriber Identity A unique internal network ID identifying a mobile subscriber.

L

LDAP	Lightweight Directory Access Protocol A protocol for providing and receiving directory information in a TCP/IP network.

M

M

MP	Message Processor - The role of the Message Processor is to provide the application messaging protocol interfaces and processing. However, these servers also have OAM&P components. All Message Processors replicate from their Signaling OAM's database and generate faults to a Fault Management System.
MSISDN	Mobile Station International Subscriber Directory Number. The unique, network-specific subscriber number of a mobile communications subscriber. MSISDN follows the E.164 numbering plan; that is, normally the MSISDN is 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.
NOAM	Network Operations, Administration, and Maintenance

O

OAMP	Operations, Administration, Maintenance and Provisioning

P

PCRF	Policy and Charging Rules Function. The ability to dynamically control access, services, network capacity, and charges in a network. Maintains rules regarding a subscriber's use of network

P

resources. Responds to CCR and AAR messages. Periodically sends RAR messages. All policy sessions for a given subscriber, originating anywhere in the network, must be processed by the same PCRF.

S

SDO

Subscriber Data Object

Subscription Data Object. An SDO consists of subscription state information and a collection of registers for storing entities. An individual SDO applies to one subscriber. A pool SDO applies to a group of subscribers.

SIP

Session Initiation Protocol

A peer-to-peer protocol used for voice and video communications.

SOAP

Simple Object Access Protocol

SPR

Subscriber Profile Repository

A logical entity that may be a standalone database or integrated into an existing subscriber database such as a Home Subscriber Server (HSS). It includes information such as entitlements, rate plans, etc. The PCRF and SPR functionality is provided through an ecosystem of partnerships.

SS7

Signaling System #7

A communications protocol that allows signaling points in a network to send messages to each other so that voice and data connections can be set up between these signaling points. These

S

messages are sent over its own network and not over the revenue producing voice and data paths. The EAGLE is an STP, which is a device that routes these messages through the network.

U

UDR

User Data Repository - A logical entity containing user data