Oracle® Communications

Provisioning

User Data Repository SOAP Provisioning Interface Specification

Release 12.2

E71664-03

June 2018

CAUTION: Use only the Installation procedure included in the Install Kit.

Before installing any system, access My Oracle Support (https://support.oracle.com) and review any Technical Service Bulletins (TSBs) that relate to this procedure.

My Oracle Support (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 My Oracle Support 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.

See more information on My Oracle Support in My Oracle Support.

ORACLE

Oracle Communications User Data Repository SOAP Provisioning Interface Specification, Release 12.2

E71664-03

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

1.1 Purpose and Scope

This document presents the SOAP Provisioning interface used by 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 can add, change, or delete subscriber/pool information in the Oracle Communications User Data Repository database.

The primary audience for this document includes customers, Oracle customer service, software development, and product verification organizations, and any other Oracle personnel who use the SOAP interface.

1.2 References

These document references were used to create this document.

[1] *IMS Sh interface; Signalling flows and message contents*, 3GPP TS 29.328, Release 11
[2] *Sh interface based on the Diameter protocol; Protocol details*, 3GPP TS 29.329, Release 11
[3] *User Data Convergence (UDC); Technical realization and information flows; Stage 2*, 3GPP TS 23.335, Release 11
[4] *SDM v9.3 Subscriber Provisioning Reference Manual*, 910-6870-001 Revision A, January 2014

1.3 Glossary

This section lists terms and acronyms specific to this document.

Table 1: Glossary

Acronym/Term	Definition
ACID	Atomic, Consistent, Isolatable, Durable
BLOB	Binary Large Object
CFG	Configuration Data. Data for components and system identification and configuration.
CPS	Customer Provisioning System
DP	Database Processor
FRS	Feature Requirements Specification
FTP	File Transfer Protocol
GUI	Graphical User Interface
IMSI	International Mobile Subscriber Identity, or IMSI [im-zee]
IP	Internet Protocol
KPI	Key Performance Indicator

Acronym/Term	Definition
MEAL	Measurements, Events, Alarms, and Logs
MP	Message Processor
MSISDN	Mobile Subscriber ISDN Number
NA	Not Applicable
NE	Network Element
NPA	Numbering Plan Area (Area Code)
NPHO	Non Pool Host UDR
OAMP	Operations, Administration, Maintenance, and Provisioning
NOAM&P	Network OAM and Provisioning
PCRF	Policy Charging and Rules Function
PS	Provisioning System
PSO	Pool Spanning UDRs
SDO	Subscriber Data Object
SEC	Subscriber Entity Configuration
SNMP	Simple Network Management Protocol
SOAM	System Operation, Administration, and Maintenance
SPR	Subscriber Profile Repository
TCP	Transmission Control Protocol
UDR	User Data Repository
UTC	Coordinated Universal Time
VIP	Virtual IP
XML	Extensible Markup Language

2. SYSTEM ARCHITECTURE

2.1 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 was traditionally stored in the HSS/HLR/AuC, Application servers, and so on, is stored in UDR as specified in 3GPP UDC information model [3]. 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 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 perform the service. User identities (for example, MSISDN, IMSI, NAI and AccountId), service data (for example, 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 (for example, transparent data stored by Application servers for service use, user status, usage, and so on).

Oracle Communications User Data Repository is a database system providing the storage and management of subscriber policy control data for PCRF nodes. Subscriber/Pool data is created/retrieved/modified or deleted through the provisioning or by the Sh interface peers (PCRF). The subscriber/pool data is stored in Oracle Communications User Data Repository:

- Subscriber
    - Profile
    - Quota
    - State
    - Dynamic Quota
- Pool
    - pool profile
    - Pool Quota
    - Pool State
    - Pool Dynamic Quota

Figure 1 illustrates a high level the Oracle Communications User Data Repository Architecture.

In Figure 1, User Data Repository 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 UDR database resides on the N-OAMP servers. This release focuses on the development of the Sh messaging interface for use with the UDR application.

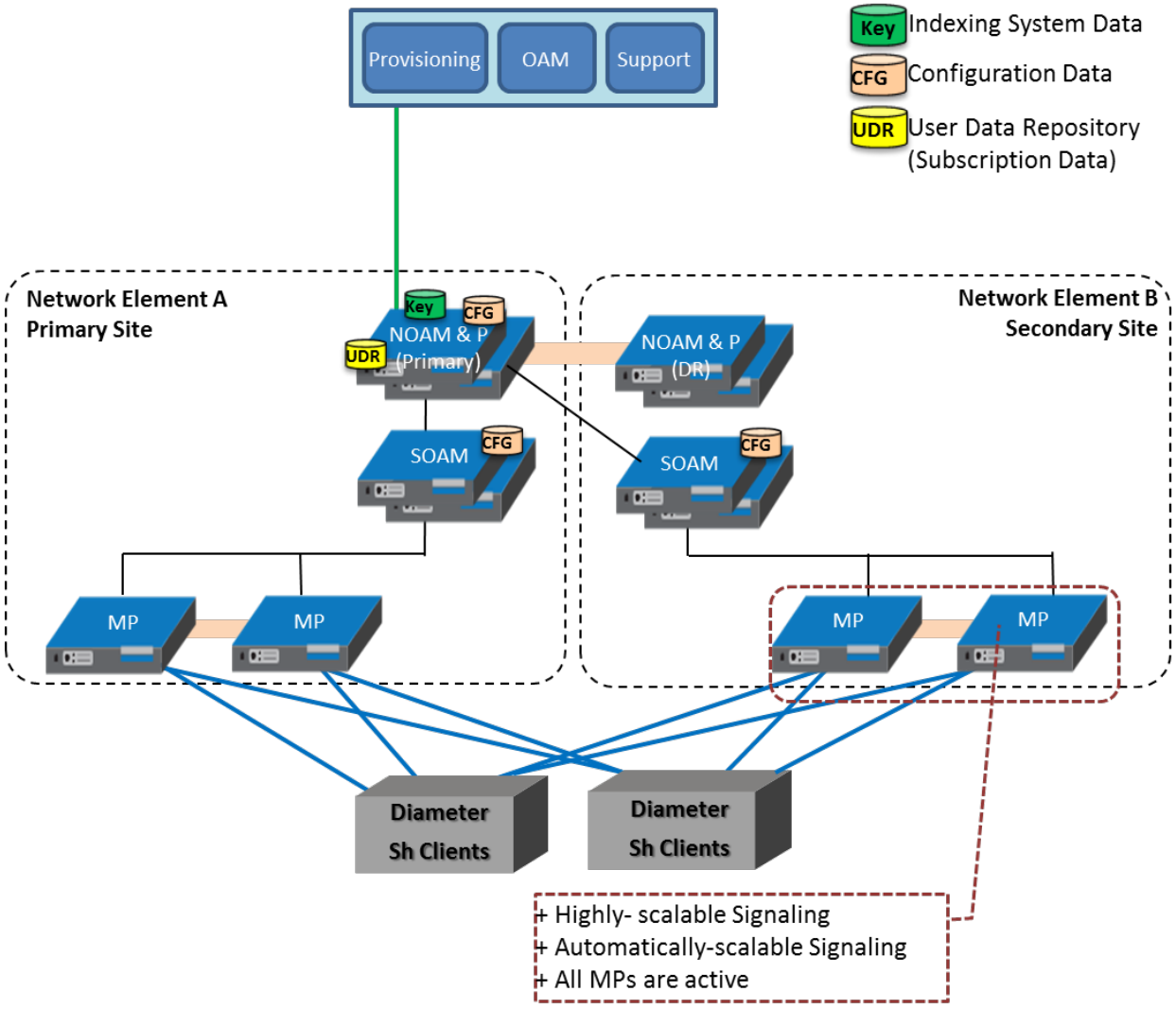
As the product evolves forward, the subscriber profiles in UDR can be expanded to support data associated with additional applications. Along with that, the MPs can be expanded to support additional Diameter interfaces associated with these applications. The IPFE can be integrated with the product to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

The Network level OAMP server (NOAM&P) 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 replicates data from NOAM&P and in turn replicates the data to the message processors.

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

Figure 1: User Data Repository High Level Architecture



2.2 Provisioning Interface

The SOAP provisioning interface provides data manipulation commands:

Subscriber:

- Subscriber profile create/retrieve/delete
- Subscriber profile field create/retrieve/modify/delete
- Subscriber opaque data create/retrieve/modify/delete

  Quota, State, and Dynamic Quota

- Subscriber data row and/or field create/modify/delete

  Quota, State and Dynamic Quota

- Subscriber transparent data create/retrieve/modify/delete

  Quota, State and Dynamic Quota

- Reset of Subscriber Quota transparent data row

Pool:

- Pool profile create/retrieve/delete
- Pool profile field create/retrieve/modify/delete
- Pool opaque data create/retrieve/modify/delete

  Pool Quota, Pool State and Pool Dynamic Quota

- Pool transparent data create/retrieve/modify/delete

  Pool Quota, Pool State and Pool Dynamic Quota

- Pool data row and/or field create/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
- Reset of Pool Quota transparent data row

2.3 XML SOAP Application Server

The application in the provisioning process that interfaces to SOAP provisioning clients and is run on every active NOAM&P server. The XML SOAP Application Server (XSAS) is responsible for:

- Accepting and authorizing SOAP provisioning client connections
- Processing and responding to SOAP requests received from provisioning clients
- Performing provisioning requests directly on the database
- Updating the provisioning command log with requests sent and responses received

2.4 Provisioning Clients

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 XSAS, the CPS may add, delete, change or retrieve information about any subscriber or pool.

CPS 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 VIP for the Primary NOAM&P.

Provisioning clients must re-establish connections with the XSAS using VIP for the Primary UDR after switchover from the active Primary UDR serverto the standby UDR server. Provisioning clients also redirect connections to the VIP for the Secondary after switchover from the Primary UDR site to the Disaster Recover UDR site.

Provisioning clients must run a timeout for the response to a request, if 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.

2.5 Security

For securing connections between the SOAP interfaces 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 and IP address ranges from which clients are authorized to establish a TCP/IP connection from.

The XSAS 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 active connections established from an IP address which is removed from the Authorized IP list are immediately closed.

2.6 Multiple Connections

The XSAS 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 section 2.9.3).

The provisioning client establishes a TCP/IP connection to XSAS before sending the first SOAP command. After performing of the request, the XSAS 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 Appendix B) are 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 on a single connection.
- When calculating the provisioning TPS rate, if any `<tx>` transactions are sent (see section 2.8.1), then the TPS rate is calculated using the number of requests contained in the `<tx>`. A `<tx>` request does not count as 1 TPS.

2.7 Request Queue Management

If multiple clients simultaneously issues requests, each request is queued and processed in the order in which it was received on a per connection basis. The client is not required to wait for a response from one request before issuing another.

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 can send a number of valid update requests, which are performed 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 the response to any/some/all of the previous requests.

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 are performed in parallel (but requests on each connection are still processed serially).

2.8 Database Transactions

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

2.8.1 Block Transaction Mode

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

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

If a block transaction fails, the request in the block that encountered an error has the appropriate error code set, all requests apart from the one that fails has 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 and Transaction Durability Timeout system variables, which are defined in Appendix B. If any of those limits are exceeded, the transaction is aborted and automatically rolled back.

If a block transaction is sent which contains more than Maximum Requests in SOAP <tx> XML requests, then the request fails with a SOAP error <message error="20"> (see section 3.1.3).

NOTE: The relevant transaction related measurements are incremented once per <tx> request (that is, by +1). The relevant request based measurements are incremented once per request contained in the <tx>. All requests share the same outcome. For example, if a <tx> request contained 5 requests, and the transaction was successful, then measurement *RxXsasProvMsgsSuccessful* is incremented by 5. If the first 3 requests in the transaction were successful, and the 4th request fails, then the transaction fails, is rolled back, and *RxXsasProvMsgsFailed* is incremented by 5.

2.8.1.1 Request Format

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

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

  NOTE: Any *resonly* value supplied in the <tx> takes precedence on any *resonly* value supplied in a contained request in <req>. If no *resonly* value is supplied in the <tx>, then the value supplied in a contained request in <req> takes precedence. The default value for *resonly* when not supplied is n.

- *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 Appendix B.

2.8.1.2 Response Format

```
<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>
```

- *nbreq*: The number of requests contained in the original XML <tx> request.
- *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 notes for resonly in section 2.8.1.1).

  Values: 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 was rolled back. Other values are dependent on the request being performed, and are listed in the description for that specific request
- *affected*: The number of subscribers affected by the request. A value of 1 indicates success.

  NOTE: This value may be non-zero for requests that were valid in a transaction, but where a subsequent request failed, and the transaction was rolled back. The *affected* value is given to indicate that the request will be successful when committed

NOTES

- For a transaction to be considered successful, all error values in all request responses must be 0.
- Results for a select request may be returned a response 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 two subscribers, and gets a thrid 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 is successful, but the second request fails. The transaction is rolled back.

The second request fails due to error FIELD_NOT_FOUND. The third command is 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 is 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 is 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>
```

2.8.2 ACID-Compliance

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

2.8.2.1 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 in that transaction which were perforemd 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.

2.8.2.2 Consistency

Data across all requests performed inside a transaction is consistent.

2.8.2.3 Isolation

All database changes made in 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 in a transaction cannot be seen by operations outside of the transaction.

2.8.2.4 Durability

After a transaction has been committed and become durable, it persists and is not 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 recovers committed transaction updates in spite of system software or hardware failures. If a failure (such as a loss of power) occurs in the middle of a transaction, the database returns 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 additive configurable levels of durability are supported:

- Durability to the disk on the active provisioning server (that is, just 1)
- Durability to the local standby server memory (that is, 1 + 2)
- Durability to the active server memory at the Disaster Recovery site (that is, 1 + 2 + 3)

2.9 Connection Management

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

2.9.1 Connections Allowed

The Allow SOAP Connections configuration variable (see Appendix C) 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. Any attempts to connect are rejected.

When Allow SOAP Connections is set back to ALLOWED, the connections are accepted again.

2.9.2 Disable Provisioning

When the Oracle Communications User Data Repository GUI option to disable provisioning is selected, existing connections remain up, and new connections are allowed. But, any provisioning request that is sent is 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 section 6.1.1.

2.9.3 Idle Timeout

HTTP connection between Provisioning client and XSAS is handled persistent fashion. The SOAP Interface Idle Timeout configuration variable (see Appendix B) indicates the time to wait before closing the connection due to inactivity (that is, no requests are received).

2.9.4 Maximum Simultaneous Connections

The Maximum SOAP Connections configuration variable (see Appendix B) defines the maximum number of simultaneous SOAP interface client connections. If an attempt is made to connect more than the number of SOAP connections allowed, the connection is rejected by the SOAP server.

2.9.5 TCP Port Number

The SOAP Interface Port configuration variable (see Appendix B) defines the SOAP interface TCP listening port.

2.10 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 are 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"/>
```

2.11 Multiple Subscriber Key Processing

UDR allows multiple key values for a subscriber to be the <where> element supplied in requests.

When multiple keys are supplied in a request (such as an IMSI and an MSISDN), UDR looks up all supplied keys, and only considers the subscriber record found if all supplied keys correspond to the same subscriber.

If any key value does not exist, then KEY_NOT_FOUND is returned. If multiple keys are provided, and all keys exist, but do not correspond to the same subscriber, then the error MULTIPLE_KEYS_NOT_MATCH is returned.

Example request:

```
<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="302370123456789"/></expr>
    <expr><attr name="MSISDN"/><op value="="/><value val="15145551234"/></expr>
  </where>
</req>
```

Multiple values for the same key type are also allowed, such as if a subscriber has two provisioned IMSIs, this structure is allowed.

```
<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="302370123456789"/></expr>
    <expr><attr name="IMSI"/><op value="="/><value val="206224111222333"/></expr>
    <expr><attr name="MSISDN"/><op value="="/><value val="15145551234"/></expr>
  </where>
</req>
```

UDR supports as many keys that are allowed for a subscriber in the request.

NOTE: For pool based requests, only a single PoolID is allowed. You cannot mix PoolID and subscriber key values in the same request, and doing so results in an `INVALID_XML` error response. For example, this structure is not allowed:

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
:
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/><value val="302370123456789"/></expr>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

2.12 Congestion Control

If UDR starts to encounter congestion (based on high CPU usage), then based on the congestion level, UDR rejects requests based on the *reqname* (see section 4.2.1).

- If the minor CPU usage threshold is crossed (CL1), then UDR rejects select requests
- If the major CPU usage threshold is crossed (CL2), then UDR rejects select, update, operation, and tx (transaction) requests
- If the critical CPU usage threshold is crossed (CL3), then UDR rejects all requests

The error returned by the SOAP interface when a request is rejected due to congestion is:

```
<res error="70045" affected="0"/>
```

2.13 Pools Spanning UDRs

Pools spanning UDRs allow subscribers to be a member of a pool that resides on a different UDR instance. A pool network is defined containing the list of UDR instances across which pools may span. These UDR instances are interconnected and networking/provisioning traffic passes between the instances.

A Pool Host UDR maintains pool data which may have pool members on other UDR instances. A Non Pool Host UDR hosts pool members for which pool data resides on a Pool Host UDR.

Pools Spanning UDR feature is only supported in combination with Oracle Communications Policy Management 9.7.4 or higher. This feature cannot be deployed unless the UDR is interworking with policy release 9.7.4 or higher.

2.14 Enterprise Pools

Enterprise Pools have the capability to support more than 25 members in a pool. Basic Pools maintain a threshold of 25 members as the maximum number of subscribers that are allowed. Enterprise Pools are pools containing more than 25 members and there is no maximum number of pool members enforced.

A field in the pool profile called Type is used to distinguish between a basic pool and an enterprise pool. If the Type field is not present, then this implies that the pool is a basic pool. A basic pool can be converted to an enterprise pool by updating the profile to set the Type field to have a value of enterprise. An enterprise pool can be converted to a basic pool by removing the Type field, as long as the number of members in the pool does not exceed the maximum allowed for a basic pool.

Pools spanning UDRs support the Enterprise Pool feature. With this feature, a pool profile on a Pool Host UDR can be provisioned as an enterprise pool (the Type field set to enterprise in the pool profile). A PSO that is provisioned as an Enterprise pool on the Pool Host UDR is considered an Enterprise pool on a Non Pool Host UDR. The Type field in the pool profile on the Non Pool Host UDR is not required to be explicitly provisioned. Provisioning a pool profile with the Type field on the NPHO is rejected with error Operation Not Allowed.

3. SOAP INTERFACE DESCRIPTION

Oracle Communications User Data Repository 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 an UDR format XML request/response. These XML request types are supported:

- Update
- Insert
- Delete
- Select
- Operation

A SOAP provisioning client application is responsible for:

- Establishing a TCP/IP connection with the SOAP server using the VIP for the Primary UDR and the SOAP XSAS listening port (as specified in section 2.9).
- Creating and sending SOAP request messages (as specified in section 4.2.1) to the SOAP server.
- Receiving and processing SOAP response messages (as specified in section 4.2.2) received from the SOAP server.
- Detecting and handling connection errors. It is recommended that the TCP keep-alive interval on the TCP/IP connection for the client is set so that a disconnection problem is quickly detected and reported.

3.1.1 SOAP Header Format

In SOAP messages, the authentication of the *username*/*password* is part of the SOAP Envelope Header. When the Authentication feature is enabled, UDR validates the *username*/*password* received in the header of the SOAP request to verify that the user who generated the request is a vaild user. Any requests that do not match valid users are rejected. The *username*/*password* provided in the SOAP Header are ignored when the authentication feature is disabled and the request is processed. In the context of SOAP requests, the SOAP header is outlined in Figure 2.

NOTE: User names and passwords must not contain extra spaces, additional characters, new line characters, and so on.

Figure 2: SOAP Header Format

```
<SOAP-ENV:Header>
  <ns1:UserName
           SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
           SOAP-ENV:mustUnderstand="1"
           xsi:type=" SOAP-ENC:string"
           xmlns:ns1="http://www.oracle.com/udr/"
      xmlns:SOAP-ENC c="http://schemas.xmlsoap.org/soap/encoding/">[Add UserName
here]</ns1:UserName>
  <ns1:Passwd
           SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
           SOAP-ENV:mustUnderstand="1"
           xsi:type=" SOAP-ENC:string"
           xmlns:ns2="http://www.oracle.com/udr/"
           xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">[Add Password
here]</ns1:Passwd>
</ SOAP-ENV:Header>
```

3.1.2 SOAP Request/Response Format

The SOAP interface uses SOAP as wrapper of XML requests and responses. The detailed format of the request is illustrated in Figure 3, and the response format in Figure 4.

Figure 3: SOAP Request 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://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 wrapper in Figure 4 for the XML response and error codes. Note that 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 listed here:

Figure 4: 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://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>
```

3.1.3 Status Codes and Error Messages

If an error occurred in processing the request or with the format of the message, an error result code is sent:

- `<message error="0">`: normal, request transaction was sent and processed
- `<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 (for example, a problem with format or value out of range). The Error code numbers are generated by UDR
- `<message error="10">`: Communication problem, unable to process the request transaction. The response does not contain any other response/error content
- `<message error="20">`: Unable to parse the request transaction. The response does not contain any other response/error content

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>
```

3.1.3.1 Error Codes

The list of error codes is described in Appendix A.

3.1.3.2 Legacy SPR Format SOAP Request/Response

UDR can be configured to operate in a compatibility mode for a legacy SPR, which affects the SOAP request/response format. See Appendix C for more details.

4. SOAP INTERFACE MESSAGE DEFINITIONS

4.1 Message Conventions

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

Table 2: Message Conventions

Symbol	Description
italics	Parameter values that are replaced by an actual parameter name or numeric value.
spaces	Spaces (that is, zero or more space characters) may be inserted anywhere except in a single name or number. At least one space is required to separate adjacent names or numbers.
…	Variable number of repeated entries. For example: dn DN1, dn DN2, …,dn DN7, dn DN8
< >	Angle brackets are used to enclose parameter values that are choices or names. For example, in parameter1 <1|2|3>, the numbers represent specific value choices. In parameter2 <ServerName>, the ServerName represents the actual value. In parameter3 <0..3600>, the numbers represent a choice in the range from 0 to 3600.
[ ]	Square brackets are used to enclose an optional parameter and its value, such as `[, parameter1 <1|2|3>]`. A parameter and its value that are not enclosed in square brackets are mandatory.
|	When the pipe symbol is used in a parameter value list, such as Parameter1 `<1|2|3>`, it indicates a choice between available values.
,	A literal comma is used in the message to separate each parameter that is specified.

4.2 Basic XML Message Format

4.2.1 Request

Figure 5 describes the basic layout of an XML request, with all the different options and parameters included. UDR requests are made up of different combinations of the parameters. All combinations are lised for illustrative purposes. Proper examples of which parameters are relevant for each request are described in the related sections.

Figure 5. Basic Layout of an XML Request

```
<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 (that is, result 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 (which mens, 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 *id* attribute is specified 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 (on duplicate key) allows an insert request to convert the insert request to an update request if the target entity exists. The *odk* attribute is optional. If *odk* is set to yes, then if the entity being inserted exists, the entity is updated instead of the request failing. The default value for the flag is to not convert the insert request to an update request. Therefore, if the target entity exists, the request fails.

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 compatibility. 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. In the <set> element, zero, one, or many <expr><attr name="fieldName"/><value val="fieldValue"/></expr> elements 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 is deleted, this is accomplished by setting the *fieldValue* as empty (that is, ""), and additionally specifying the attribute isnull="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. In the <select> element, one or many <expr><attr name="fieldName"/></expr> elements 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 are 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 values to a list) or RemoveFromSet (to remove a values from a list). The name of the field being modified is specified in *setFieldName*, and the values being added/removed are specified in *setFieldValue*. Multiple comma separated values can be specified in *setFieldValue*, or 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.

To define an XML data blob, the *cdataFieldValue* field must contain a <cdata> element, and then includ the data in the constructs of an XML CDATA section. The CDATA section:

1. Starts the section with <![CDATA[,
2. Add the *cdataFieldValue* containing the XML data blob.
3. End the CDATA section with]]>

Most commands identify the subscriber for which the provisioning 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 in Table 3.

Table 3: *keyValue* Validation

keyType	*keyValue* Validation
IMSI	10 to 15 numeric digits
MSISDN	8 to 15 numeric digits. **NOTE:** A preceding + (plus) symbol is not supported, and is rejected.
NAI	For NAI supported formats see Table 6.
AccountId	1 to 255 characters (allowed values are any ASCII printable characters in the range x20 to x7e).
PoolID	1 to 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 in the row 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*.

4.2.1.1 XML Comments in a Request

A SOAP request may contain XML comments, such as:

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

- If the comment is in the request, it is ignored.
- If the comment is contained with the XML data blob for an opaque entity in the CDATA constraint, then the comment is stored in the XML data blob.
- If the comment is contained with the XML data blob for a transparent entity in the CDATA constraint, then the comment is not stored in the XML data blob.

4.2.2 Response

Figure 6 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 listed for illustrative purposes. Proper examples of which parameters are relevant for each response are described in the related sections.

Figure 6. Basic Layout of an XML Response

```
<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 a table in each request section. The list of error codes is described in Appendix A.

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 (with operation requests), result data returned is contained in a <rset> (rowset) element. An <rset> can be zero (if no matching data was found) or one <row> element (UDR does not support returning multiple <row> elements). A <row> can have 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.

NOTES:

- An <rv> element can contain an 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 data blob, but has an empty value, this is indicated with <rv></rv>.
- Whenever XML data 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.

4.3 Encoding of Multiple Embedded CDATA Sections

Requests and responses may contain multiple embedded CDATA sections—that is, 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.

Chapters 6 and 7 describe CDATA sections in the UDR commands without any reference on how these sare represented after they are embedded in the SOAP envelope.

The subsections 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 are returned to the UDR client.

4.3.1 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 in the <message> element does not have to be changed).

- Replace all embedded occurrences of <![CDATA[with <![CDATA[
- Replace all embedded occurrences of]]> with]]>

For example:

```
<?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="Quota"/>
              <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>
```

4.3.2 Response

When a response is received, every < (less than) and > (greater than) character in the message element is replaced with `<` and `>` respectively (including the initial CDATA).

Example

```
<?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>
```

4.4 Case Sensitivity

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

For example, this request is valid:

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

But this request is not:

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

Entity names as specified in an *entityName* are not case sensitive.

Entity field names, key names, and row element/identifiers names are not case-sensitive, for example *fieldName*, *setFieldName*, *keyName*, *instanceFieldName*, and *rowIdName*.

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

Operation names as specified in an *operName* are not case sensitive.

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.
- A field name is used to specify an entity (for example a *fieldName*, *cdataFieldName* or *opaqueDataType*) is also not case-sensitive.
- 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 is 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 is used to delete the value WEEKEND, then this fails.
- When an attribute in the XML data blob contains the row identifier name (also known as, *rowIdName*). For example; for Quota, the element `<quota name="AggregateLimit">` contains the attribute called *name*. The row identifier name is not case-sensitive.
- 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 is used to get the row DAYPASS, then this fails.
- When an *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 an *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 that has the value Data, and the **`Get Row`** command is used to get the row with the field value DATA, then this fails.
- 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` does not find the subscriber.
- When an element in the XML data blob contains the row element name (for example, for Quota, the row `<quota name="AggregateLimit">` contains the element called quota) the row element name is not case-sensitive.
- When an operation name is specified in an *operName* (such as when using the `GetPoolID` operation), the operation name is not case-sensitive.
- When an entity name is specified in an *entityName* (such as when using the **`Create Row`** command), the entity name is not case-sensitive.

4.5 List of Messages

Table 4 lists the operations/messages for subscriber data. Each row of the table represents a command.

Table 4: 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

Operation Data/Type	Command	SOAP
	Delete Row	delete
Subscriber Data Row Field	Get Row Field	select
	Update Row Field	update
	Delete Row Field	update (using *isnull* attribute)
Subscriber Data Field	Create Data Field	insert
	Get Data Field	select
	Update Data Field	update
	Delete Data Field	update (using *isnull* attribute)
Subscriber Special Operation	Reset Quota	operation

Table 5 lists the operations/messages for pool data. Similar to the previous table, each row of the table represents a command.

Table 5: 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

Operation Data/Type	Command	SOAP
	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)
Pool Data Field	Create Data Field	insert
	Get Data Field	select
	Update Data Field	update
	Delete Data 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
Pool Special Operation	Reset Pool Quota	operation

5. UDR DATA MODEL

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.

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 OSS system (via a provisioning interface) and referenced by the PCRF (via the Sh interface).

- Quota

  Information that represents the use of managed resources (quota, pass, top-up, roll-over) for the subscriber. Although the UDR provisioning interfaces allow quota data to be manipulated, this data is written by the PCRF and is only referenced by the provisioning interfaces.

- State

  Subscriber-specific properties. Similar to 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). 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 subscriber can be associated with a single pool.

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 use of managed resources (quota, pass, top-up, roll-over) for the pool.

- Pool State

  Pool-specific properties.

- Pool Dynamic Quota

  Dynamically configured information related to managed resources (pass, top-up).

- 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:

- Subscription
- A subscription exists for every individual subscriber
- 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
- Pool Subscription
- A pool subscription exists for every pool
- Maps a pool subscription to the pool identity through which it can be accessed
- Maps a pool subscription to the individual subscriptions of the subscribers that are members of the pool
- 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
- 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 access to the Subscriber SDO via IMSI, MSISDN, NAI or AccountId. The pool SDO can be accessed via PoolID.

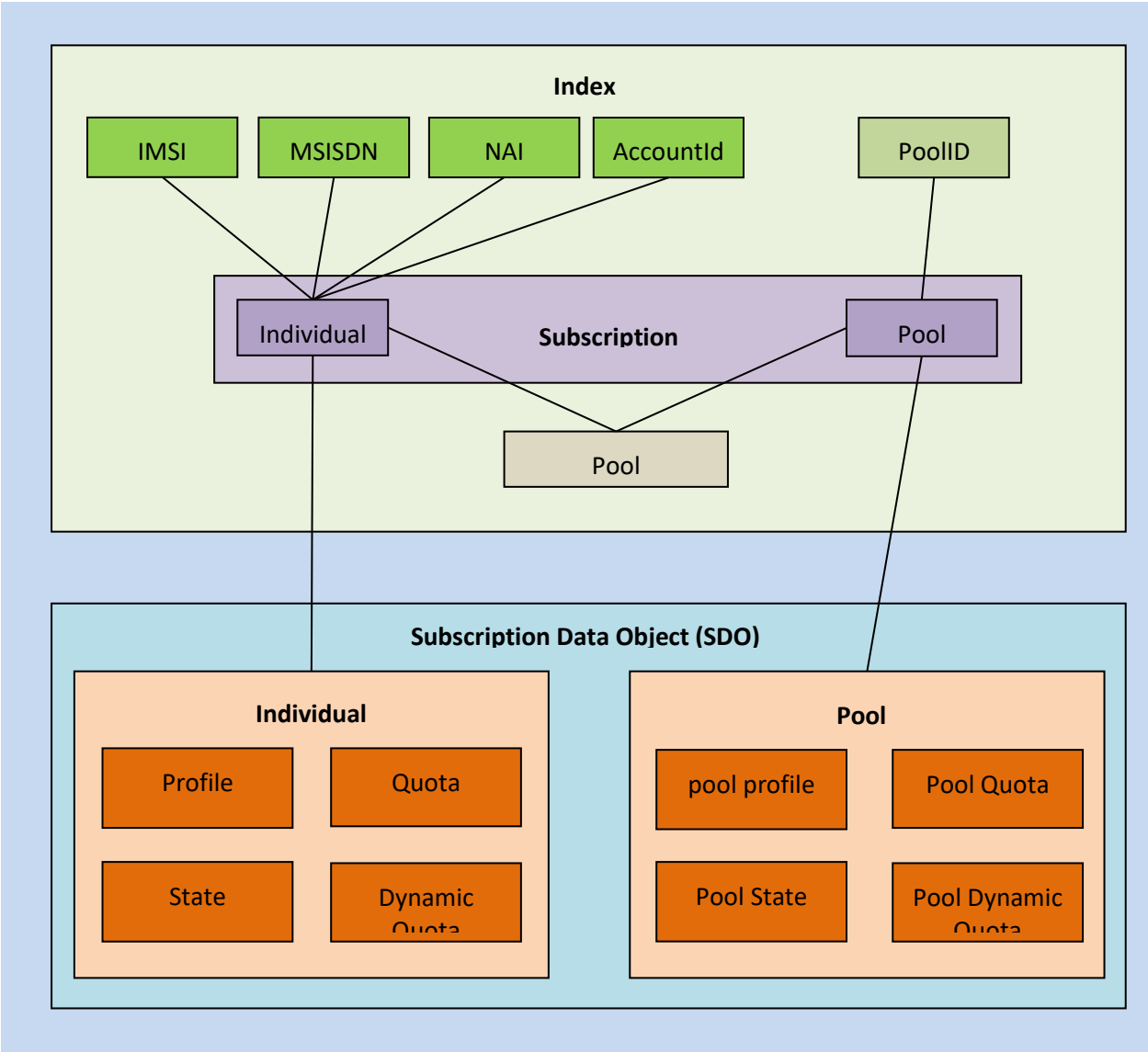
A field in an entity can:

- Be defined as mandatory, or optional. A mandatory field must exist, and cannot be deleted.
- Have a default value. If an entity is created, and the field is not specified, it is created with the default value.
- Be defined so that after it is created, it cannot be modified. Any attempt to update the field after it is created fails.

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

NOTE: This section describes the default UDR data model as defined in the Subscriber Entity Configuration (SEC). The data model can be customized via the UDR GUI.

Figure 7: Data Model



5.1 Subscriber Data

5.1.1 Subscriber Profile

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

The Subscriber profile supports a sequence of attributes. Each record must have at least one of the 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 to 15 numeric digits. A preceding + (plus) symbol is not supported, and is rejected.

Table 6: Subscriber Profile Entity Definition

Name (XML tag)	Type	Description
subscriber	---	Sequence (multiplicity = 1)
MSISDN	String	List of MSISDNs (8 to 15 numeric digits). A separate entry is included for each MSISDN associated with the profile for the subscriber.
IMSI	String	List of IMSIs (10 to 15 numeric digits). A separate entry is included for each IMSI associated with the profile for the subscriber.

NAI	String	User and domain length is between 0 to 63 characters
		NOTE: The limitation for 0 to 63 characters is because NAIs beyond 63 characters may not be possible to transfer through all devices. You must ensure that the combination of user and domain does not exceed 63 characters (not including the @ character).
		List of NAIs. A separate entry is included for each NAI associated with the profile for the subscriber. Allowable formats are user@domain, user or @domain
		The user or domain can be empty.
		Characters that are allowed for the user:
		!
		%
		$
		A to Z
		a to z
		0 to 9
		.
		-
		_
		/
		*
		=
		^
		`
		|
		#
		+
		?
		{
		}
		~
		Characters that are allowed for the domain:
		A to Z
		a to z
		0 to 9

Name (XML tag)	Type	Description
		. - _ Example NAI Formats: bob @privatecorp.example.net fred$@example.com eng.example.net!nancy@example.net eng%nancy@example.net bob#+ ?@example.net
AccountId	String	Any string that can be used to identify the account for the subscriber (1 to 255 characters). Allowed values are any ASCII printable character, values x20 to x7e.
BillingDay	String	Allowed values are 0 to 31. The day of the month (1 to 31) when the associated quota for the subscriber is reset. 0 indicates that the default value configured at the PCRF level should is 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 profile for the subscriber.
Tier	String	Tier for the subscriber.
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.

5.1.2 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 is associated with a subscriber record and supports the sequence of attributes listed in Table 7.

NOTES:

- The Quota entity contains a version number. Different attributes maybe be present based on the version number value of the entity being accessed. In UDR, only v3 of Quota is supported.
- Quota Entity is generally created, updated, and deleted by Oracle Communications Policy Management, therefore it is advised to avoid operations on Quota entity via Provisioning Interfaces such as SOAP or REST.
- 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`** command. If a field is defined as resettable, and the field exists, then it is set to the value indicated. If the field does not exist in the Quota, it is not created.

      NOTE: If a resettable field does not exist and the field is also defined as defaultable, then the field is created with the value indicated

Table 7: Quota Entity Definition

Name (XML tag)	Type	Default Value	Description	Quota Versions
usage	---	---	Sequence (multiplicity = 1)	1/2/3
version	String	---	Version of the schema.	1/2/3
quota	---	---	Sequence (multiplicity = N)	1/2/3
name	String	---	Quota name (identifier)	1/2/3

Name (XML tag)	Type	Default Value	Description	Quota Versions
cid	String	---	Internal identifier used to identity a quota in a subscriber profile.	1/2/3
time	String	Empty string ”	This element tracks the time-based resource consumption for a Quota.	1/2/3
totalVolume	String	“0”	This element tracks the bandwidth volume-based resource consumption for a Quota.	1/2/3
inputVolume	String	“0”	This element tracks the upstream bandwidth volume-based resource consumption for a Quota.	1/2/3
outputVolume	String	“0”	This element tracks the downstream bandwidth volume-based resource consumption for a Quota.	1/2/3
serviceSpecific	String	Empty string “”	This element tracks service-specific resource consumption for a Quota.	1/2/3
nextResetTime	String	Empty string “”	When set, it indicates the time after which the usage counters need to be reset. See section 5.3 for format details.	1/2/3
Type	String	Empty string “”	Type of the resource in use.	2/3
grantedTotalVolume	String	“0”	Granted Total Volume represents the granted total volume of all the subscribers in the pool for pool quota. For individual quota, it represents the granted volume to all the PDN connections for that subscriber.	2/3
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

5.1.3 State

The State entity is written by the PCRF to store the state of various properties managed as a part of the policy for the subscriber. 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.

Note: The default fields configured are not:

- Resettable
- Defaultable

The State entity supports the sequence of attributes listed in Table 8.

Table 8: State Entity Definition

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.

5.1.4 Dynamic Quota

The DynamicQuota entity records usage is associated with passes and top-ups. The DynamicQuota entity is associated with the Subscriber profile and may be created or updated by either the PCRF or the 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.

NOTE: The default fields configured are not:

- Resettable
- Defaultable

The DynamicQuota entity supports the sequence of attributes listed in Table 9.

Table 9: Dynamic Quota Entity Definition

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 is used to match top-ups to quota definitions on the PCRF. This name is used in policy conditions and actions on the PCRF.
InstanceId	String	A unique identifier to identify this instance of a dynamic quota object.

Name (XML tag)	Type	Description
Priority	String	An integer represented as a string. This number allows service providers to specify when a pass or top-up is 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 section 5.3 for format details.
expirationdatetime	String	The date/time after which the pass or top-up is considered to be exhausted. See section 5.3 for format details.
purchasedatetime	String	The date/time when a pass was purchased. See section 5.3 for format details.
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.

5.2 Pool Data

5.2.1 Pool Profile

The pool profile includes a set of custom fields that the provisioning system can use to store information associated with the pool. The values in custom fields are generally set by the 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 only included in the record if they are specified when the record is created/updated.

The Pool profile record consists of the sequence of attributes listed in Table 10.

Table 10: Pool Profile Entity Definition

Name (XML tag)	Type	Description
pool	---	Sequence (multiplicity = 1)
PoolID	String	Pool identifier (1 to 22 numeric digits, minimum value 1).
BillingDay	Uint8	The day of the month (1 to 31 when the associated quota for the pool is reset. 0 indicates that the default value configured at the PCRF level is used.
BillingType	String	The billing frequency, monthly, weekly, or daily.
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the profile for the pool.
Tier	String	Tier for the pool.
Type	String	Field used to identify an Enterprise Pool. Allowed value is enterprise and is not case-sensitive
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.
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.

Name (XML tag)	Type	Description
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.

5.2.2 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 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 version number of 3.

The PoolQuota entity attributes are the same as defined for the Quota entity in section 5.1.2.

NOTE: Pool Quota Entity is generally created, updated and deleted by Oracle Communications Policy Management, therfore it is advised to avoid operations on Pool Quota entity via Provisioning Interfaces such as SOAP or REST.

5.2.3 Pool State

The PoolState entity is written by the PCRF to store the state of various properties managed as a part of the policy for the pool. 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.

NOTE: The default fields configured are not:

- Resettable
- Defaultable

The PoolState entity attributes are the same as defined for the State entity in section 5.1.3.

5.2.4 Pool Dynamic Quota

The PoolDynamicQuota entity records usage associated with passes and top-ups 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 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.

NOTE: The default fields configured are not:

- Resettable
- Defaultable

The PoolDynamicQuota entity attributes are the same as defined for the DynamicQuota entity in section 5.1.4.

5.3 Date/Timestamp Format

The Date/Timestamp format used by many fields is:

```
CCYY-MM-DDThh:mm:ss[<Z|<+|->hh:mm>]
```

This corresponds to either:

4. CCYY-MM-DDThh:mm:ss (local time)
5. CCYY-MM-DDThh:mm:ssZ (UTC time)
6. CCYY-MM-DDThh:mm:ss+hh:mm (positive offset from UTC)
7. CCYY-MM-DDThh:mm:ss-hh:mm (negative offset from UTC)

Where:

- 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

These are valid examples of a field in Date/Timestamp format:

- 2015-06-04T15:43:00 (local time)
- 2015-06-04T15:43:00Z (UTC time)
- 2015-06-04T15:43:00+02:00 (positive offset from UTC)
- 2015-06-04T15:43:00-05:00 (negative offset from UTC)

6. SUBSCRIBER PROVISIONING

NOTE: For command responses, the error code values described are listed in Appendix A.

6.1 Subscriber Profile Commands

Table 11: Summary of subscriber profile Commands

Command	Description	Keys	Command Syntax
Create Profile	Create a subscriber or 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		`<req name="delete">` `<ent name="Subscriber"/>`

6.1.1 Create Profile

Description

This operation creates a 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` element. 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.

NOTES:

- The subscriber profile data provided is fully validated against the definition in the SEC. If the validation check fails, then the request is rejected.
- An entity for the subscriber can be created by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the XML data blob value in *cdataFieldValue*.
- 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="keyNameN"/><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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *fieldNameX*: A user defined field in the subscriber profile.
- *fieldValueX:* Corresponding field value assigned to *fieldNameX*.
- *cdataFieldNameX*: A user defined field in the subscriber profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC.

  Value is either Quota, State, or DynamicQuota.

- *cdataFieldValueX:* Contents of the XML data blob for *cdataFieldNameX*.

NOTES:

- One key is mandatory. Any combination of key types are allowed. More than one occurrence of each key type (such as, IMSI/MSISDN/NAI/AccountId) is supported, up to an engineering configured limit.
- Key/field order in the request is not important.

Response

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

- *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.

  Values: 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 Table 12 for other values.
- *affected*: The number of subscribers created by the request. A value of 1 indicates success.

Table 12: Create Profile Error Codes

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 in the entity as defined in the SEC
KEY_EXISTS	Key Already Exists. A subscriber/pool 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
AE_KEY_EXISTS	An AE subscriber exists with the given key. Only applicable when option *enableAEKeyAlreadyExistsErrCode* is enabled

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 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 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*, 2 *MSISDNs* 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="15145551234,15141234567"/></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>
         <property>
           <name>expire</name>
           <value>2014-02-09T11:20:32</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>
```

Request 5

A subscriber is created, with an invalid username and/or password in the SOAP header. The request is not required in the response.

```
<soapenv:Header>
<ns1:UserName
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="1"
    xsi:type="soapenc:string"
    xmlns:ns1="http://www.oracle.com/udr/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">[bad UserName]</ns1:UserName>
<ns1:Passwd
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="1"
    xsi:type="soapenc:string"
    xmlns:ns2="http://www.oracle.com/udr/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">[invalid Password]</ns1:Passwd>
</soapenv:Header>

<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 5

The request fails. The *error* value indicates that username/password authentication failed.

```
<req name="insert" resonly="y">
  <res error="70054" affected="0"/>
</req>
```

Request 6

A subscriber is created, with an AccountId, MSISDN and IMSI keys. An AE subscriber exists with the given IMSI and `enableAEKeyAlreadyExistsErrCode` option is enabled.

```
<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 6

The request fails. The *error* value indicates an AE subscriber exists with the given IMSI, and the *affected* rows are 0.

```
<req name="insert" resonly="y">
  <res error="70055" affected="0"/>
</req>
```

Request 6

A subscriber is created, with an AccountId, MSISDN and IMSI keys. An AE subscriber exists with the given IMSI and `enableAEKeyAlreadyExistsErrCode` option is disabled (the default case).

```
<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 6

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

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

6.1.2 Get Profile

Description

This operation retrieves all field-value pairs created for a subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

The *keyNameX* and *keyValueX* values 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 the keys of the *keyNameX*/*keyValueX* values supplied must exist.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

- y

  Only provide the result, do not include the original request

- n

  Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 13 for other values.
- *affected*: The number of subscribers returned. A value of 1 indicates 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 subscriber profile data (that is, XML data blob).

Table 13: Get Profile Error Codes

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

Examples

Request 1

A request is made to retreive 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>
```

Request 2

A request is made to retreive profile data for a subscriber. An IMSI and MSISDN are supplied, and both keys are valid for the same 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="15145551234"/></expr>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="302370123456789"/></expr>
  </where>
</req>
```

Response 2

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="MSISDN">15145551234</field>
```

```
            <field name="IMSI">302370123456789</field>
            <field name="BillingDay">6</field>
            <field name="Tier">Gold</field>
            <field name="Entitlement">Weekpass</field>
          </subscriber>]]>
        </rv>
    </row>
  </rset>
</req>
```

6.1.3 Delete Profile

Description

This operation deletes all profile data (field-value pairs) and opaque data for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

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

Request

```
<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 14 for other values.
- *affected*: The number of subscribers deleted. A value of 1 indicates success.

Table 14: Delete Profile Error Codes

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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

Examples

Request 1

The subscriber with the given *MSISDN* is deleted. The subscriber exists. The request (by default) is 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>
```

Request 3

The subscriber with the given *MSISDN* and *IMSI* is deleted. Subscribers exist with the specified MSISDN and IMSI, but they are not the same subscriber. 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>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="302370123456789"/></expr>
  </where>
</req>
```

Response 3

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

```
<req name="delete" resonly="y">
  <res error="70043" affected="0"/>
</req>
```

6.2 Subscriber Profile Field Commands

Table 15: Summary of Subscriber Profile Field Commands

Command	Description	Keys	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 values for the specified field		`<req name="select">` `<ent name="Subscriber"/>`
Update Field	Update fields to the specified values		`<req name="update">` `<ent name="Subscriber"/>`

Command	Description	Keys	Command Syntax
Delete Field	Delete all the values for the specified fields		<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">

6.2.1 Add Field Value

Description

This operation adds one or more values to the specified multi-value field for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

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

All existing values are retained, and the values specified are inserted. For example, if the current value of a field is a,b,c, and this command is used with value d, after the update the field has the value a,b,c,d.

NOTES:

- If a value being added exists, the request fails.
- The *fieldValue* is case-sensitive. An attempt to add the value a to current field value of a,b,c fails, but an attempt to add the value a is successful and result in the field value being a,b,c,A
- A request to add field values can also be mixed with a request to update or delete a fields. But, the same field for which an `AddToSet` operation is being performed cannot also be updated or deleted, else the request fails.
- A request to add field values using the `AddToSet` operation can also contain a `RemoveFromSet` operation to delete field values. If both operations are included in the same request, the `AddToSet` is performed before the `RemoveFromSet`, irrespective of the order in which they are supplied.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values 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 be present in the field.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="fieldName1"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
[
      <expr><attr name="fieldName2"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
      :
      <expr><attr name="fieldNameX"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
]
    </oper>
```

```
  </set>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *fieldNameX*: A user defined field in the subscriber profile.
- *fieldValueX:* Corresponding field value assigned to *fieldNameX*.
- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

***NOTES*:**

- One or more fieldValueX values for a *fieldNameX* can be supplied. To add more than one value, either supply a comma separated list of values, or include multiple <expr> elements for the field.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 16 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 16: Add Field Value Error Codes

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 field is not a valid field in 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 added 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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 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 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 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 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>
```

Request 5

A request is made to add the keys *14161234567* and *19191112222* to the *MSISDN* field, and also add the *DayPass* value to the *Entitlement* field. The subscriber has a single MSISDN value of 15145551234. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="MSISDN"/><value val="14161234567"/></expr>
      <expr><attr name="MSISDN"/><value val="19191112222"/></expr>
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
  </where>
</req>
```

Response 5

The request is successful, and the values were added to the *MSISDN* and *Entitlement* fields. The subscriber has 3 MSISDN values of *15145551234*, *14161234567*, and *19191112222*. The original request is not included.

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

6.2.2 Get Field

Description

This operation retrieves the values for the specified fields for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

NOTE: An entity for the subscriber can be retrieved by specifying an *opaqueDataType* corresponding to the interface entity name in the SEC.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *fieldNameX*: A user defined field in the subscriber profile.
- *opaqueDataTypeX*: A user defined field in the subscriber profile, that represents a transparent or opaque data entity.

  Value is either Quota, State, or DynamicQuota.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

NOTES:

- At least one *fieldNameX*/*opaqueDataTypeX* field must be requested
- The order in which *fieldNameX*/*opaqueDataTypeX* are specified in the request is not important
- Multiple subscriber key values can be supplied. See section 2.11 for details.

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>
```

- *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.

  Values: 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 Table 17 for other values.
- *affected*: The number of subscribers returned. A value of 1 indicates success.
- *rowValueX*: The value of the requested field (for normal fields, not for opaque/transparent entities).

  NOTE: For multi-value fields, the value can contain a comma separated list of values on a single line. For example: `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 *fieldName*/*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>`.

Table 17: Get Field Error Codes

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The given field is not a valid field in 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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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>Prepaid</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 data 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>Prepaid</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>
```

6.2.3 Update Field

Description

This operation updates a fields to the specified values for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*. This operation replaces (sets) the values of the fields, which means that any existing values for the fields are deleted first.

For multi-value fields, all existing values are removed, and only the values specified are inserted. Adding values to a current set is accomplished using `Add Field Value`. For example, if the current value of a field is a,b,c, and this command is used with value d, after the update the field has 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. That is, 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.

NOTES:

- If the requested fields are valid, but are not present, they are created.
- An entity for the subscriber can be replaced by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the XML data blob value in *cdataFieldValue*.
- 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.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

Each requested field *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"/>
|
      <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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *fieldNameX*: A user defined field in 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. For example: `a,b,c`.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *cdataFieldNameX*: A user defined field in the subscriber profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC.

  Value is either Quota, State, or DynamicQuota.

- *cdataFieldValueX:* Contents of the XML data blob for *cdataFieldNameX*.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 18 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 18: Update Field Error Codes

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 in 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

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
INVAL_REPEATABLE_ELEM	Invalid Repeatable Element
INVALID_XML	Invalid Input XML
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 *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* field to *14161234567*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="MSISDN"/>
          <value val="14161234567"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
  </where>
</req>
```

Response 6

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

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

Request 7

A request is made to update the value of the *MSISDN* field to *14161234567*. The subscriber has 3 existing MSISDN values of 15145551234, 14161234567, and 19191112222. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="MSISDN"/>
          <value val="14161234567"/></expr>
  </set>
  <where>
```

```
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
  </where>
</req>
```

Response 7

The request is successful, and the *MSISDN* value was updated and a single value of *14161234567*. The original request is not included.

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

Request 8

A request is made to update the value of the subscribers *NAI* to two values of *mum@foo.com* and *cust514@op.com*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="NAI"/>
          <value val="mum@foo.com,cust514@op.com"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
  </where>
</req>
```

Response 8

The request is successful, the *NAI* field was updated. The subscriber has 2 NAIs. The original request is not included.

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

6.2.4 Delete Field

Description

This operation the specified fields for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

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

NOTE: The field being deleted does notneed to have a current value. It can be empty (deleted), and the request succeeds.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

Each requested field *fieldNameX* 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *fieldNameX*: A user defined field in the subscriber profile.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 19 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 19: Delete Field Error Codes

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
ONE_KEY_REQUIRED	At least one key is required for a subscriber
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 fields. 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>
```

Request 3

A request is made to delete the MSISDN field. The subscriber has an IMSI and an MSISDN key field. The request is not required in the response.

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

Response 3

The request is successful, and the MSISDN field was deleted. The original request is not included.

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

Request 4

A request is made to delete the NAI field. The subscriber has an NAI key field only. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="NAI"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="NAI"/><op value="="/>
          <value val="mum@foo.com"/></expr>
  </where>
</req>
```

Response 4

The request fails. The *error* value indicates the only key cannot be deleted, and the *affected* rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70044" affected="0"/>
</req>
```

6.2.5 Delete Field Value

Description

This operation deletes one or more values from the specified field for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

This operation can only be performed 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 field is removed from the subscriber profile (that is, there is no XML element present).

NOTES:

- The fieldValue is case-sensitive. An attempt to remove the value a from a current field value of a,b,c is successful, but an attempt to remove the value a fails.
- A request to delete field values can also be mixed with a request to update or delete a fields. But, the same field for which a `RemoveFromSet` operation is being performed cannot also be updated or deleted, else the request fails.
- A request to delete field values using the `RemoveFromSet` operation can also contain an `AddToSet` operation to add field values. If both operations are included in the same request, the `AddToSet` is performed before the `RemoveFromSet` irrespective of the order in which they are supplied.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values 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 removed must be present in the field.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="fieldName1"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
[
      <expr><attr name="fieldName2"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
      :
      <expr><attr name="fieldNameX"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
]
    </oper>
  </set>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
```

```
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *fieldNameX*: A user defined field in the subscriber profile.
- *fieldValueX*: Corresponding field value assigned to *fieldname*.

***NOTES*:**

- One or more fieldValueX values for a *fieldNameX* can be supplied. To remove more than one value, either supply a comma separated list of values, or include multiple <expr> elements for the fielde.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 20 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 20: Delete Field Value Error Codes

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in 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
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
ONE_KEY_REQUIRED	At least one key is required for a subscriber
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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>
```

Request 3

A request is made to remove the key value *14161234567* from the *MSISDN* field. The subscriber has three MSISDN values, 14161234567, 151454551234 and 15141234567. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="MSISDN"/><value val="14161234567"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
  </where>
</req>
```

Response 3

The request is successful, and the value was removed from the *MSISDN* field. The subscriber has 2 MSISDN values, 151454551234 and 15141234567. The original request is not included.

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

6.3 Subscriber Opaque Data Commands

Table 21: Summary of Subscriber Opaque Data Commands

Command	Description	Keys	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"/>…`

6.3.1 Create Opaque Data

Description

This operation creates the opaque data of the specified *opaqueDataType* for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

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

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

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

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

No opaque data of the *opaqueDataType* must 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
 [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *odk* (optional): Indicates that the insert request is converted to an update if the specified opaque data type exists.
- *opaqueDataType*: A user defined type/name for the opaque data.

  Value is either Quota, State, or DynamicQuota.

- *cdataFieldValue*: Contents of the XML data blob.
- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 22 for other values.
- *affected*: The number of subscribers inserted/updated. A value of 1 indicates success.

Table 22: Create Opaque Data Error Codes

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 field is not a valid field in 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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

Examples

Request 1

A request is made to create the *Quota* opaque data. The Quota XML data 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 data 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 data 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>pass</Type>
    <InstanceId>15678</InstanceId>
    <Priority>4</Priority>
    <InitialTime>135</InitialTime>
    <InitialTotalVolume>2000</InitialTotalVolume>
    <InitialInputVolume>1500</InitialInputVolume>
    <InitialOutputVolume>500</InitialOutputVolume>
    <InitialServiceSpecific>4</InitialServiceSpecific>
    <activationdatetime>2015-03-09T11:20:32</activationdatetime>
    <expirationdatetime>2015-04-09T11:20:32</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 data 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>
```

6.3.2 Get Opaque Data

Description

This operation retrieves the opaque data of the specified *opaqueDataType* for the subscriber that is identified by the keys in *keyNameX* and *keyValueX*.

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

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
```

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

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *opaqueDataType*: A user defined type/name for the opaque data.

  Value is either Quota, State, or DynamicQuota.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

Response

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

- *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.

  Values: 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 Table 23 for other values.
- *affected*: The number of subscribers returned. A value of 1 indicates success
- *cdataRowValue*: Contents of the XML data blob.

***NOTES*:**

- 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 opaque data (that is, XML data blob).
- Multiple subscriber key values can be supplied. See section 2.11 for details.

Table 23: Get Opaque Data Error Codes

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
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in the entity as defined in the SEC
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 is successful, and indicates that the requested opaque data type 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 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>
```

Request 4

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

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
<select>
    <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 indicates that Quota exists and State does not exist. 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>
       <rv null="y"/>
    </row>
  </rset>
</req>
```

6.3.3 Update Opaque Data

Description

This operation updates the opaque data of the specified *opaqueDataType* for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

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

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *opaqueDataType*: A user defined type/name for the opaque data.

  Value is either Quota, State, or DynamicQuota.

- *cdataFieldValue*: Contents of the XML data blob.
- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 24 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 24: Update Opaque Data Error Codes

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 field is not a valid field in the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

Examples

Request 1

A request is made to update the *State* opaque data. The State XML data 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>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 1

The request is successful, and the State opaque 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 *Quota* opaque data. *Quota* is a valid opaque data type, but the subscriber does not have this opaque data type. The Quota XML data 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>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 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>
```

6.3.4 Delete Opaque Data

Description

This operation deletes the opaque data of the specified *opaqueDataType* for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

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.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *opaqueDataType*: A user defined type/name for the opaque data.

  Value is either Quota, State, or DynamicQuota.

  NOTE: The data is deleted by setting an empty field value, and also specifying the attribute `isnull="y"`.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 25 for other values.
- *affected*: The number of subscribers deleted. A value of 1 indicates success.

Table 25: Delete Opaque Data Error Codes

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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 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 *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>
```

6.4 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 in 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*.

NOTE: Subscriber data row commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an `OPER_NOT_ALLOWED` error being returned.

Table 26: Summary of Subscriber Data Row Commands

Command	Description	Keys	Command Syntax

Create Row	Insert data row into transparent data of the specified type.	(MSISDN, IMSI, NAI or AccountId) and Row Identifier
		or
		(MSISDN, IMSI, NAI or AccountId) and Row Identifier and Instance 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> …`
			`[`
			`  <expr>`
			`  <attr name="instanceFieldName">`
			`  <value val="instanceFieldValue">`
			`  </expr> …`
			`]`

Delete Row	Delete data row in transparent data of the specified type		`<req name="delete">`
			`  <ent name="entityName"/>`
			`  … <expr>`
			`  <attr name="rowIdName">`
			`  <value val="rowIdValue">`
			`  </expr> …`
			`[`
			`  <expr>`
			`  <attr name="instanceFieldName">`
			`  <value val="instanceFieldValue">`
			`  </expr> …`
			`]`

6.4.1 Create Row

Description

This operation creates a data row for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

The data row identifier field is specified in *rowIdName*, and the row identifier value is specified in *rowIdValue*. All *fieldNameX* fields specified are set in the row.

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to update an existing row called DAYPASS is successful, and two rows called DayPass and DAYPASS are present.
- If the transparent entity specified in entityName does not exist for the subscriber, it is created.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

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

Request

NOTE: This command has two different formats.

1. The *keyNameX*/*keyValueX* values in the <set> element.
2. The *keyNameX*/*keyValueX* values in a <where> element.

Format 1

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="keyName1"/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></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>
]
```

```
  </set>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *odk* (optional): Indicates that the insert request is converted to an update if the data row for the specified entity exists.
- *entityName*: A user defined entity type/name for the transparent data.
  - Value is QuotaEntity for the Quota transparent data.
  - Value is DynamicQuotaEntity for the DynamicQuota transparent data.
- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data
  - Value is name for DynamicQuota transparent data
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *fieldNameX*: A user defined field in 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. For example: `a,b,c`.

NOTES:

- 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.
- 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 fails because it is not known which of the multiple rows to update.

- Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 27 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 27: Create Row Error Codes

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 field is not a valid field in 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 match the given row criteria **NOTE:** Only returned when the `odk="yes"` attribute is supplied, and duplicate candidate rows to update are found
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 is 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 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="922337203685488888"/></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 exists. Two rows with the *name* of *Q3* 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="922337203685477777"/></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 (that is, more than one row has a *name* of *Q3*, and therfore 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>
```

Request 6

A request is made to create a data row in the *QuotaEntity* (Quota) data. The *MSISDN*, *IMSI* , and *AccountId* keys are supplied, which reference the same subscriber. 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="15145551234"/></expr>
    <expr><attr name="IMSI"/><value val="302370123456789"/></expr>
    <expr><attr name="AccountId"/><value val="123456"/></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 6

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 7

A request is made to create a data row in the *QuotaEntity* (Quota) data, using the alternate request format. The *MSISDN* and *NAI* keys are supplied, which reference the same subscriber. 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>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
    <expr><attr name="NAI"/><op value="="/>
          <value val="dad@foo.com"/></expr>
  </where>
</req>
```

Response 7

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 8

A request is made to create a data row in the *DynamicQuotaEntity* (DynamicQuota) data. The *MSISDN* key is supplied. The data row identifier field is *name*, and the value is *DQ1*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="name"/><value val="DQ1"/></expr>
    <expr><attr name="Type"/><value val="top-up"/></expr>
    <expr><attr name="InstanceId"/><value val="15678"/></expr>
    <expr><attr name="Priority"/><value val="4"/></expr>
    <expr><attr name="InitialTime"/><value val="135"/></expr>
    <expr><attr name="InitialTotalVolume"/><value val="2000"/></expr>
    <expr><attr name="InitialInputVolume"/><value val="1500"/></expr>
    <expr><attr name="InitialOutputVolume"/><value val="500"/></expr>
    <expr><attr name="InitialServiceSpecific"/><value val="4"/></expr>
    <expr><attr name="activationdatetime"/><value val="2015-05-22T00:00:00-05:00"/></expr>
    <expr><attr name="expirationdatetime"/><value val="2015-05-29T00:00:00-05:00"/></expr>
    <expr><attr name="InterimReportingInterval"/><value val="100"/></expr>
    <expr><attr name="Duration"/><value val="10"/></expr>
  </set>
</req>
```

Response 8

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>
```

6.4.2 Get Row

Description

This operation retrieves a data rows for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

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

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

NOTES:

- The *rowIdValue* is case-sensitive. If a row existed called DayPass, then an attempt to retrieve a row called DayPass is successful, but an attempt to retrieve a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to retrieve a row with a field that has the value Data is successful, but an attempt to retrieve a row with a field that has the value DATA fails.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

The *entityName* must reference a valid transparent 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
    <expr><attr name="rowIdName"/><op value="="/>
          <value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is QuotaEntity for the Quota transparent data.
  - Value is DynamicQuotaEntity for the DynamicQuota transparent data.
- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the Quota transparent data
  - Value is `InstanceId` or `Type` for the DynamicQuota 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>
```

- *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.

  Values: 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 Table 28 for other values.
- *affected*: The number of subscribers returned. A value of 1 or more indicates success, whether or not a row was found.
- *cdataRowValueN*: Contents of the XML data blob containing one requested/matching data row.

***NOTES*:**

- 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.
- If the transparent entity exists, but the row value was not found, then the `<rv>` (row value) indicates the row does not exist by containing the value `<rv null="y"/>`.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

Table 28: Get Row Error Codes

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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 four 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" 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 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>
```

Request 7

A request is made to get the *DQ1* data row from the DynamicQuota data, with the *InstanceId* value of *11223344*. The DynamicQuota data contains four rows called *DQ1*. Two with *InstanceId* of *11223344*, one with an *InstanceId* of *99887766* and one with an *InstanceId* of *55556666*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="DQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/><value val="11223344"/></expr>
```

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

Response 7

The request is successful and the 2 DynamicQuota data rows with an *InstanceId* 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"?>
        <definition>
          <version>1</version>
          <DynamicQuota name="DQ1">
            <Type>pass</Type>
            <InstanceId>11223344</InstanceId>
            <Priority>4</Priority>
            <InitialTime>135</InitialTime>
            <InitialTotalVolume>2000</InitialTotalVolume>
            <InitialInputVolume>1500</InitialInputVolume>
            <InitialOutputVolume>500</InitialOutputVolume>
            <InitialServiceSpecific>4</InitialServiceSpecific>
            <activationdatetime>2015-05-22T00:00:00-05:00</activationdatetime>
            <expirationdatetime>2015-05-29T00:00:00-05:00</expirationdatetime>
            <InterimReportingInterval>100</InterimReportingInterval>
            <Duration>10</Duration>
          </DynamicQuota>
        </definition>]]>
      </rv>
    </row>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <definition>
          <version>1</version>
          <DynamicQuota name="DQ1">
            <Type>top-up</Type>
            <InstanceId>11223344</InstanceId>
            <Priority>3</Priority>
            <InitialTime>125</InitialTime>
            <InitialTotalVolume>1000</InitialTotalVolume>
            <InitialInputVolume>500</InitialInputVolume>
            <InitialOutputVolume>500</InitialOutputVolume>
            <InitialServiceSpecific>4</InitialServiceSpecific>
            <activationdatetime>2015-05-22T00:00:00-05:00</activationdatetime>
            <expirationdatetime>2015-05-29T00:00:00-05:00</expirationdatetime>
            <InterimReportingInterval>100</InterimReportingInterval>
            <Duration>10</Duration>
          </DynamicQuota>
        </definition>]]>
      </rv>
    </row>
  </rset>
</req>
```

6.4.3 Delete Row

Description

This operation deletes a data row for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

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

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

NOTES

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to delete a row called DayPass is successful, but an attempt to delete a row called DAYPASS fails
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to delete a row with a field that has the value Data is successful, but an attempt to delete a row with a field that has the value DATA fails.
- The deletion of a non-existent data row is not considered an error.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

The *entityName* must reference a valid transparent 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
    <expr><attr name="rowIdName"/><op value="="/>
          <value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is QuotaEntity for the Quota transparent data.
  - Value is DynamicQuotaEntity for the DynamicQuota transparent data.
- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName.*

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 29 for other values.
- *affected*: A value of 1 indicates that the rows existed and were deleted, or that the row did not exist

Table 29: Delete Row Error Codes

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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

Examples

Request 1

A request is made to delete 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 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 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 2

A request is made to delete 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 from the Quota data, with the <cid> value of *11223344*. The Quota data contains two rows called *Q4*. One with <cid> of *11223344*, the other with a <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 was 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>
```

Request 6

A request is made to delete the *DQ1* data row from the DynamicQuota data, with the *Type* value of *pass*. The DynamicQuota data contains two rows called *DQ1*. One with *Type* of *pass* the other with a *Type* of *top-up*. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="DynamicQuotaEntity"/>
```

```
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
<expr><attr name="name"/><op value="="/>
      <value val="DQ1"/></expr>
<expr><attr name="Type"/><op value="="/>
      <value val="pass"/></expr>
  </where>
</req>
```

Response 6

The request is successful, and the data row in the DynamicQuota data was deleted that matched the *Type*. The original request is not included.

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

6.5 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 in 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*.

NOTE: Subscriber data row field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an `OPER_NOT_ALLOWED` error being returned.

Table 30: Summary of Subscriber Data Row Field Commands

Command	Description	Keys	Command Syntax

Get Row Field	Retrieve values for the specified fields	(MSISDN, IMSI, NAI or AccountId) and Row Identifier and Field name or (MSISDN, IMSI, NAI or AccountId) and Row Identifier, Instance Identifier and Field name	`<req name="select">`
			`<ent name="entityName"/>`
			`… <expr>`
			`<attr name="rowIdName">`
			`<value val="rowIdValue">`
			`</expr> …`
			`[`
			`<expr>`
			`<attr name="instanceFieldName">`
			`<value val="instanceFieldValue">`
			`</expr> …`
			`]`

Update Row Field	Update fields to the specified values		`<req name="update">`
			`<ent name="entityName"/>`
			`… <expr>`
			`<attr name="rowIdName">`
			`<value val="rowIdValue">`
			`</expr> …`
			`[`
			`<expr>`
			`<attr name="instanceFieldName">`
			`<value val="instanceFieldValue">`
			`</expr> …`
			`]`

Command	Description	Keys	Command Syntax
Delete Row Field	Delete all values for the specified fields		<req name="update"> <ent name="*entityName*"/> … <expr> <attr name="*rowIdName*"> <value val="*rowIdValue*"> </expr> … [<expr> <attr name="*instanceFieldName*"> <value val="*instanceFieldValue*"> </expr> …]

6.5.1 Get Row Field

Description

This operation retrieves a fields in a data row for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

The data row identifier field is specified in *rowIdName*, and the row identifier value is specified in *rowIdValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field names are specified in *fieldNameX*.

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

NOTES:

- If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and empty field data is returned.
- The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to get a field in a row called DayPass is successful, but an attempt to get a field in a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to delete a row with a field that has the value Data is successful, but an attempt to delete a row with a field that has the value DATA fails.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

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

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

The field names 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
    <expr><attr name="rowIdName"/><op value="="/>
          <value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is QuotaEntity for the Quota transparent data.
  - Value is DynamicQuotaEntity for the DynamicQuota transparent data.
- *fieldNameX*: A user defined field in the data row.
- *fieldValueX* (optional): Corresponding field value assigned to *fieldNameX*.

  NOTE: For multi-value fields, the value can contain a comma separated list of values.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName.*

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

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>
```

- *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.

  Values: 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 Table 31 for other values.
- *affected*: The number of subscribers returned.
  - A value of 1 indicates that the specified row exists (whether or not the field was found).
  - A value of 0 indicates that the row does not exist.
- *rowValue*: The value of the requested field.

  NOTE: For multi-value fields, the value can contain a comma separated list of values on a single line. For example: `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>.

Table 31: Get Row Field Error Codes

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in 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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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

Request 5

A request is made to get the *outputVolume* field in the *Q1* data row of the Quota data. The *Q1* row exists, but the *outputVolume* field is not set. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></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 5

The request is successful, and the field is indicated to not be set. 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 *outputVolume* field in the *Q2* data row of the Quota data. The *Q2* row does not exist. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></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 6

The request fails. The *error* value indicates the row does not exist, and the *affected* rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70032" affected="0"/>
</req>
```

Request 7

A request is made to get the *InstanceId, InitialTotalVolume* and *InitialInputVolume* fields in the *DQ3* data row of the Dynamic Quota data, with the *InstanceId* value of 15678. The Dynamic Quota data contains two rows called *DQ3*. One with *InstanceId* of *15570*, the other with an *InstanceId* of 15678. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <select>
    <expr><attr name="InstanceId"/></expr>
    <expr><attr name="InitialTotalVolume"/></expr>
    <expr><attr name="InitialInputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
    <expr><attr name="name"/><op value="="/>
          <value val="DQ3"/></expr>
    <expr><attr name="InstanceId"/><op value="="/>
          <value val="15678"/></expr>
  </where>
</req>
```

Response 7

The request is successful, and the requested field values of *InstanceId, InitialTotalVolume* and *InitialInputVolume* fields are returned. The original request is not included.

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

```
      <rv>15678</rv>
      <rv>2000</rv>
      <rv>1500</rv>
    </row>
  </rset>
</req>
```

6.5.2 Update Row Field

Description

This operation updates fields in a data row for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

The data row identifier field is specified in *rowIdName* and the row identifier value is specified in *rowIdValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field names are specified in *fieldNameX*. If the specified fields are valid, but do not exist, they are created.

If more than one row matches the requested *rowIdName/rowIdValue* and *instanceFieldName*/*instanceFieldValue* (if specified), then the update request fails.

NOTES

- If the specified row does not exist, the request fails.
- If the requested fields are valid, but are not present, they are created.
- The *rowIdValue* is case-sensitive. If a row existed called DayPass, then an attempt to update a field in a row called DayPass is successful, but an attempt to update a field in a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to delete a row with a field that has the value Data is successful, but an attempt to delete a row with a field that has the value DATA fails.
- Multiple subscriber key values can be supplied. See section 2.11 for details.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

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

The data row with the given identifier/instance in the transparent data should exist for the subscriber.

The field names 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
```

```
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
    <expr><attr name="rowIdName"/><op value="="/>
          <value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is QuotaEntity for the Quota transparent data.
  - Value is DynamicQuotaEntity for the DynamicQuota transparent data.
- *fieldNameX*: A user defined field in 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. For example: `a,b,c`.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName.*

Response

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

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

- *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.

  Values: 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 Table 32for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 32: Update Row Field Error Codes

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 field is not a valid field in 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
MULTIPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 *Q6*. 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>
```

Request 4

A request is made to update the *InitialTotalVolume* and *InitialInputVolume* fields in the *DQ1* data row of the DynamicQuota data, with the *InstanceId* value of 15678. The *DQ1* data row exists in the DynamicQuota data, but there are two rows called *DQ1* one with *InstanceId* of *15570*, the other with an *InstanceId* of 15678. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <set>
    <expr><attr name="InitialTotalVolume"/><value val="1000"/></expr>
    <expr><attr name="InitialInputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
    <expr><attr name="name"/><op value="="/>
          <value val="DQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/>
          <value val="15678"/></expr>
  </where>
</req>
```

Response 4

The request is successful, and the *InitialTotalVolume* and *InitialInputVolume* fields in the data row with *InstanceId* value of 15678 were updated. The original request is not included.

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

6.5.3 Delete Row Field

Description

This operation deletes a fields in a data row for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

The data row identifier field is specified in *rowIdName*, and the row identifier value is specified in *rowIdValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field names are specified in *fieldNameX*.

If more than one row matches the requested *rowIdName/rowIdValue* and *instanceFieldName*/*instanceFieldValue* (if specified), then the delete request fails.

NOTES

- If the specified row does not exist, the request fails. 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.
- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to delete a field in a row called DayPass is successful, but an attempt to delete a field in a row called DAYPASS fails.
- 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 that has the value Data is successful, but an attempt to delete a field in a row with a field that has the value DATA fails.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

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

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

The field names 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
    <expr><attr name="rowIdName"/><op value="="/>
          <value val="rowIdValue"/></expr>
[
    <expr><attr name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is QuotaEntity for the Quota transparent data.

- Value is DynamicQuotaEntity for the DynamicQuota transparent data.

- *fieldNameX*: A user defined field in 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. For example: `a,b,c`.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName.*

Response

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

- *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.

  Values: 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 Table 33 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates that the row existed and the field was deleted. A value of 0 indicates that the field did not exist

Table 33: Delete Row Field Error Codes

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_UNDEFINED	Field Not Defined. The field is not a valid field in the entity as defined in the SEC

Error Code	Description
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
MULTIPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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" 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="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 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>
```

Request 4

A request is made to delete the *InitialInputVolume* field in the *DQ1* data row of the DynamicQuota data with the *InstanceId 11223344*. The *DQ1* data row exists in the DynamicQuota data, and is there are two rows called *DQ1*, one with *InstanceId 11223344* and one with *InstanceId 99887766*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <set>
    <expr><attr name="InitialInputVolume"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15145551234"/></expr>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="302370123456789"/></expr>
    <expr><attr name="name"/><op value="="/>
          <value val="DQ1"/></expr>
<expr><attr name="InstanceId"/><op value="="/>
      <value val="11223344"/></expr>
  </where>
</req>
```

Response 4

The request is successful, and the *InitialInputVolume* field in the *DQ1* data row in the DynamicQuota data was deleted for the row with the specified *InstanceId*. The original request is not included.

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

Request 5

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, there is only one row called *Q1* and the *inputVolume* field does not exist. 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 5

The request is successful and the original request is not included.

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

6.6 Subscriber Data Field Commands

A transparent data entity can contain data that is organized in fields where each field is defined as a name value paired with an element. For example, the State entity has a `<name>` element for the name, and a `<value>` element for the value in a `<property>` element.

```
<property>
  <name>fieldName</name>
  <value>fieldValue</value>
</property>
```

The data commands allow operations (create/retrieve/update/delete) at the field level. The required field is identified in the request by the *fieldName*.

NOTE: Subscriber data field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an `OPER_NOT_ALLOWED` error being returned.

Table 34: Summary of Subscriber Data Commands

Command	Description	Keys	Command Syntax
Create Data Field	Create/update data field in transparent data of the specified type.	(MSISDN, IMSI, NAI or AccountId) and Field Name	`<req name="insert"><ent name="entityName"/> … <expr><attr name="fieldName"><value val="fieldValue"></expr> …`
Get Data Field	Retrieve data field from transparent data of the specified type.		`<req name="select"><ent name="entityName"/> … <expr><attr name="fieldName"><value val="fieldValue"></expr> …`
Update Data Field	Update data field in transparent data of the specified type.		`<req name="update"><ent name="entityName"/> … <expr><attr name="fieldName"><value val="fieldValue"></expr> …`
Delete Data Field	Delete data field in transparent data of the specified type.		`<req name="update"><ent name="entityName"/> … <expr><attr name="fieldName"><value val="fieldValue"></expr> …`

6.6.1 Create Data Field

Description

This operation creates or updates an existing field in a transparent data for the subscriber identified by the *keyNameX* and *keyValueX*.

The field name is specified in *fieldNameX*, and the field value is specified in *fieldValueX*.

If the specified field does not exist, it is created. If the field does exist, it is updated/replaced only if the optional *odk* flag is set to yes.

NOTES

- The *fieldNameX* is not case-sensitive. If a field existed called mcc, then an attempt to create/update an existing field called MCC is successful.
- If the transparent entity specified in entityName does not exist for the subscriber, it is created.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

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

Request

NOTE: This command has two different formats.

1. The *keyNameX*/*keyValueX* values in the <set> element.
2. The *keyNameX*/*keyValueX* values in a <where> element.

Format 1

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="keyName1"/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></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="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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
  </set>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *odk* (optional): Indicates that the insert request is converted to an update if the field for the specified entity exists.
- *entityName*: A user defined entity type/name for the transparent data.

  Value is StateEntity for the State transparent data.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *fieldNameX*: The requested user defined field in the subscriber profile.

  For the State entity, this corresponds to a property in the entity.

  The *fieldNameX* case is stored exactly as it is sent in the request. This means the original case stored changes if an update is received.

- *fieldValueX*: Corresponding field value assigned to *fieldNameX*.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

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

- *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.

  Values: 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 Table 35 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 35: Create Data Field Error Codes

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

Error Code	Description
INVALID_SOAP_XML	Invalid SOAP XML
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in the entity as defined in the SEC
KEY_NOT_FOUND	Key Not Found. A subscriber/pool with the given key cannot be found
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber
REG_EXISTS	The property exists and the *odk* flag was not set to indicate the insert request be converted to an update.

Examples

Request 1

A request is made to create a property in the *State* transparent data for a subscriber. The property name is *mcc* and the property value is *315*. The subscriber does not have an existing *State* property called *mcc*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

Response 1

The request is successful, and the property *mcc* with value *315* 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 property in the *State* transparent data for a subscriber, using the alternate request format. The property name is *mcc* and the property value is *315*. The subscriber does not have an existing *State* property called *mcc*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="15141234567"/></expr>
  </where>
</req>
```

Response 2

The request is successful, and the property *mcc* with value *315* 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 property in the *State* transparent data for a subscriber. *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="insert" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

Response 3

The request is successful, and the property as well as the State entity is 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 property in the *State* transparent data for a subscriber. The property name is *mcc* and the property value is *315*. (The property exists) The *odk* attribute is included requesting the data be updated if it exists. The request is not required in the response.

```
<req name="insert" resonly="y" odk="yes">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

Response 4

The request is successful, and the *mcc* property 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 property in the *State* transparent data for a subscriber. The property name is *mcc* and the property value is *315*. (The property exists) The *odk* attribute is not included (requests the data be updated if it exists.) The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

Response 5

The request fails. The *error* value indicates the mcc property exists, and the *affected* rows are 0. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="70028" affected="0"/>
</req>
```

6.6.2 Get Data Field

Description

This operation retrieves a field for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*.

The field is specified in *fieldNameX*. The fields that match the requested *fieldNameX* are returned.

NOTES:

- If the specified field does not exist, `null="y"` is returned.
- The *fieldNameX* is not case-sensitive. If a field called mcc exists, then an attempt to get a field name called MCC is successful.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

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

A field with the identifier in the transparent data should exist for the subscriber.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <select>
    <expr><attr name="fieldName1"/></expr>
[
    <expr><attr name="fieldName2"/></expr>
            :
    <expr><attr name="fieldNameN"/><expr>
]
  </select>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

- n

  Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.

  Value is StateEntity for the State transparent data.

- *fieldNameX*: The requested user defined field in the subscriber profile.

  For the State entity, this corresponds to a property in the entity.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
<     <rv>fieldValue1</rv> | <rv null="y"> | <rv></rv> >
[
<     <rv>fieldValue2</rv> | <rv null="y"> | <rv></rv> >
      :
<     <rv>fieldValueN</rv> | <rv null="y"> | <rv></rv> >
]
    </row>
  </rset>
]
</req>
```

- *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.

  Values: 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 Table 36 for other values.
- *affected*: The number of subscribers returned. A value of 1 indicates that the specified data exists (whether or not the field was found). A value of 0 indicates that the data does not exist.
- *fieldValueX*: The value of the requested field.

NOTE: The `<rset>` (row set) element is optional. It is only present if the request was successful. One `<row>` element is returned per matching data. One `<rv>` (field value) element exists for every *fieldNameX* supplied in the original request. The `<rv>` elements are ordered the same as the *fieldNameX* properties 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>`.

Table 36: Get Data Field Error Codes

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in 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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

Examples

Request 1

A request is made to get three state properties from the *State* transparent data for a subscriber. The request is not required in the response.

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

Response 1

The request is successful, two values are returned and one property was indicated to not be set. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>315</rv>
      <rv>2015-02-09T11:20:32</rv>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

Request 2

A request is made to get the *mcc* property in the *State* transparent data for a subscriber. The *mcc* property is not set. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="StateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
```

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

Response 2

The request is successful, and the property is indicated to not be set. 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 3

A request is made to get the *mcc* property in the *State* transparent data for a subscriber. The *State* Entity does not exist. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="StateEntity"/>
  <select>
    <expr><attr name="mcc"/></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 state entity does not exist, and the *affected* subscribers are 0. The original request is not included.

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

6.6.3 Update Data Field

Description

This operation updates an existing field in a transparent data for the subscriber identified by the *keyNameX* and *keyValueX*.

The field name is specified in *fieldNameX*, and the field value is specified in *fieldValueX*.

If more than one existing field matches the requested *fieldNameX*, then the update request fails.

NOTES:

- If the requested fields are valid, but are not present, they are created.
- The *fieldNameX* is not case-sensitive. If a field name called mcc exists, then an attempt to update a field called MCC is successful.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

The *entityName* must reference a valid transparent Entity in the Interface Entity Map table 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.

  Value is StateEntity for the State transparent data.

- *fieldNameX*: The requested user defined field in the subscriber profile.

  For the State entity, this corresponds to a property in the entity.

  The *fieldNameX* case is stored exactly as it was sent in the request. This means the original case stored changes if an update is received.

- *fieldValueX:* Corresponding field value assigned to *fieldNameX*.
- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

Response

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

- *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.

  Values: 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 Table 37 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates success.

Table 37: Update Data Field Error Codes

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 field is not a valid field in 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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

Examples

Request 1

A request is made to update the *mcc* and *approved* properties of the *State* transparent data for a subscriber. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><value val="302"/></expr>
    <expr><attr name="approved"/><value val="no"/></expr>
  </set>
```

```
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="302370123456789"/></expr>
  </where>
</req>
```

Response 1

The request is successful, and *mcc* and *approved* property values are updated. The original request is not included.

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

Request 2

A request is made to update the *approved* properties in the *State* data. The *State* Entity does not exist for the subscriber. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="approved"/><value val="no"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
  </where>
</req
```

Response 2

The request fails because the *State* entity does not exist for the subscriber. The original request is not included.

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

6.6.4 Delete Data Field

Description

This operation deletes a data field in a transparent data for the subscriber identified by the *keyName* and *keyValue*.

The field identifier is specified in *fieldName*.

If more than one data field matches the requested *fieldName*, then all matching fields are deleted.

NOTES:

- The deletion of a non-existent field is not considered an error.
- The *fieldNameX* is not case-sensitive. If a field called mcc exists, then an attempt to delete a field called MCC is successful.
- Multiple subscriber key values can be supplied. See section 2.11 for details.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values supplied must exist.

The *entityName* must reference a valid transparent Entity in the Interface Entity Map table 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="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.

  Value is StateEntity for the State transparent data.

- *fieldNameX*: The requested user defined field in the subscriber profile.

  For the State entity, this corresponds to a property in the entity.

- *fieldValueX:* Corresponding field value assigned to *fieldNameX*.
- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

Response

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

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

- *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.

  Values: 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 Table 38 for other values.
- *affected*: The number of subscribers updated. A value of 1 indicates that the field was deleted. A value of 0 indicates that the field did not exist.

Table 38: Delete Data Field Error Codes

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_UNDEFINED	Field Not Defined. The field is not a valid field in 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
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

Examples

Request 1

A request is made to delete the mcc property of the State transparent data for a subscriber. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="302370123456789"/></expr>
  </where>
</req>
```

Response 1

The request is successful and mcc property 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 mcc property in State data for a subscriber. The mcc property does not exist for the State data. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="302370123456789"/></expr>
  </where>
</req>
```

Response 2

The request is successful and the original request is not included.

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

Request 3

A request is made to delete the mcc property in State data for a subscriber. The StateEntity does not exist for the subscriber. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
          <value val="302370123456789"/></expr>
  </where>
</req>
```

Response 3

The request fails. The *error* value indicates the subscriber does not have State data, and the *affected* rows are 0. The original request is not included.

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

6.7 Subscriber Special Operation Commands

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

The required row is identified in the request by the *rowIdName*/*rowIdValue*.

A specific instance of a quota (that is, a specified row) in the Quota transparent data entity can have its fields reset to pre-defined values using a provisioning command.

Table 39: Summary of Subscriber Special Operation Commands

Command	Description	Keys	Command Syntax

Reset Quota	Reset the fields in the specified Quota	(MSISDN, IMSI, NAI or AccountId) and Row Identifier	**Format 1**
			`<req name="operation">`
			`<oper name="Reset" ent="entityName">`
			Format 2
			(different/old supported format)
			`<req name="operation">`
			`<oper name="ResetQuota">`

6.7.1 Reset Quota

Description

This operation resets a particular quota row in the Quota data associated with a subscriber.

If more than one row matches the requested *rowIdName*, then the reset request fails.

If the subscriber has Quota data, then the configured values in the specified quota row are reset to the configured default values.

NOTES:

- The *rowIdName* is case-sensitive. If a row exists called DayPass, then an attempt to reset a quota row called DayPass is successful, but an attempt to reset a quota row called DAYPASS fails.
- When a Quota instance is reset using the **`Reset Quota`** command, each field that can be reset is set to its defined reset value. If the field does not exist, it is not created. But, if a resettable field does not exist, and the field has a default value, then the field is created with the default value.

Prerequisites

A subscriber with the keys of the *keyNameX/keyValueX* values supplied must exist.

The Quota transparent data must exist for the subscriber.

The specified Quota row must exist in the Quota transparent data.

Request

Format 1

```
<req name="operation" [resonly="resonly"] [id="id"]>
 <oper name="Reset" ent="entityName">
    <expr><param name="keyName1"/><op value="="/>
          <value val="keyValue1"/></expr>
[
    <expr><param name="keyName2"/><op value="="/>
          <value val="keyValue2"/></expr>
    :
    <expr><param name="keyNameN"/><op value="="/>
          <value val="keyValueN"/></expr>
]
    <expr><param name="rowIdName"/><op value="="/>
          <value val="rowIdValue"/></expr>

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

```
]
  </oper>
</req>
```

Format 2

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="ResetQuota">
    <expr><param name="keyName1"/><op value="="/>
          <value val="keyValue1"/></expr>
[
    <expr><param name="keyName2"/><op value="="/>
          <value val="keyValue2"/></expr>
    :
    <expr><param name="keyNameN"/><op value="="/>
          <value val="keyValueN"/></expr>
]
    <expr><param name="name"/><op value="="/>
          <value val="quotaName"/></expr>
```

NOTE: Format 2 is deprecated and Format 1 is used (Format 2 is still supported).

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provides the result, does not include the original request

  - n

    Includes the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.

  Value is QuotaEntity for the Quota transparent data.

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.

  Value is name for Quota transparent data.

- *rowIdValue/quotaName*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in 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.*

NOTE: Multiple subscriber key values can be supplied. See section 2.11 for details.

Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
```

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

- *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.

  Values: 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 Table 40 for other values.
- *affected*: The number of subscribers reset. A value of 1 indicates success.

Table 40: Reset Quota Error Codes

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
MULTIPLE_ROWS_FOUND	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
MULTIPLE_KEYS_NOT_MATCH	Multiple keys supplied do not refer to the same subscriber

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 Quota 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>
```

Request 4

A request is made to reset the *Q1* Quota row for a subscriber. Three keys are provided for the subscriber, and all keys are valid for the subscriber. The subscriber has Quota data, and the Quota data contains a Quota 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="IMSI"/><op value="="/>
          <value val="184569547984229"/></expr>
    <expr><param name="NAI"/><op value="="/>
          <value val="mum@foo.com"/></expr>
    <expr><param name="name"/><op value="="/><value val="Q1"/></expr>
  </oper>
</req>
```

Response 4

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 5

A request is made to reset the *Q1* Quota row for a subscriber. The subscriber has Quota data, and the Quota data contains a Quota row called *Q1*. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="QuotaEntity">
    <expr><param name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
<expr><param name="name"/><op value="="/><value val="Q1"/></expr>
  </oper>
</req>
```

Response 5

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 6

A request is made to reset the *Q1* Quota row for a subscriber. The subscriber has Quota data, the Quota data contains a Quota row called *Q1* and *instanceFieldName cid* field with value of *1234*. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="QuotaEntity">
    <expr><param name="MSISDN"/><op value="="/>
          <value val="33123654862"/></expr>
<expr><param name="name"/><op value="="/><value val="Q1"/></expr>
<expr><param name="cid" /><op value="="/><value val="1234"/></expr>
  </oper>
</req>
```

Response 6

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>
```

7. POOL PROVISIONING

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 Appendix A.

7.1 Pool Profile Commands

Table 41: Summary of Pool Profile Commands

Command	Description	Keys	Command Syntax
Create Pool	Creates a 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 data	PoolID	`<req name="select">` `  <ent name="Pool"/>`
Delete Pool	Delete pool profile data and all opaque data associated with the pool		`<req name="delete">` `  <ent name="Pool"/>`

7.1.1 Create Pool

Description

This operation creates a 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` element. Request content includes *poolId*, and field-value pairs, all as specified in the Subscriber Entity Configuration.

NOTES:

- The pool profile data provided is fully validated against the definition in the SEC. If the validation check fails, then the request is rejected.
- An entity for the pool can be created by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the XML data blob value in *cdataFieldValue*.
- 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.
- If the PSO feature is enabled and the PoolID falls in a range that is maintained by a different UDR instance, then the pool is created as a Non Pool Host UDR pool (remote pool); otherwise the pool is created as a Pool Host UDR pool.
- If the PSO feature is enabled, a pool cannot be provisioned with the *Type* field on a Non Pool Host UDR system.

Prerequisites

A pool 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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

- *fieldNameX*: A user defined field in the pool 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. For example: `a,b,c`.

- *cdataFieldNameX*: A user defined field in the pool profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC.

  Value is either PoolQuota, PoolState, or 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>
```

- *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.

  Values: 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 Table 42 for other values.
- *affected*: The number of Pools created. A value of 1 indicates success.

Table 42: Create Pool Error Codes

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 field is not a valid field in the entity as defined in the SEC
KEY_EXISTS	Key Already Exists. A subscriber/pool exists with the given key
INVALID_KEY_VALUE	The key value supplied is invalid, due to invalid characters/format.
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

Error Code	Description
OPER_NOT_ALLOWED	Operation Not Allowed

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>2014-02-01T00:00:00</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>
```

Request 3

A pool is created, with *PoolID*. PSO feature is enabled.The *PoolID* falls in a range that is maintained by a different UDR instance. 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>
    <expr><attr name="Type"/><value val="Enterprise"/></expr>
  </set>
</req>
```

Response 3

The request fails. The error indicates this operation is not allowed on Non Pool Host UDR.

```
<req name="insert" resonly="y">
  <res error="70026" affected="0"/>
</req>
```

7.1.2 Get Pool

Description

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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

Response

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

- *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.

  Values: 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 Table 43 for other values.
- *affected*: The number of pools returned. A value of 1 indicates 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 (that is, XML data blob).

Table 43: Get Pool Error Codes

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>
```

7.1.3 Delete Pool

Description

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 fails.

Request

```
<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
```

```
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

Response

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

- *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.

  Values: 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 Table 44 for other values.
- *affected*: The number of pools returned. A value of 1 indicates success.
- *cdataRowValue*: Contents of the pool profile XML data blob.

Table 44: Delete Pool Error Codes

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>
```

7.2 Pool Field Commands

Table 45: Summary of Pool Field Commands

Command	Description	Keys	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 values for the specified fields		`<req name="select">` `<ent name="Pool"/>`
Update Field	Updates fields to the specified values		`<req name="update">` `<ent name="Pool"/>`

Delete Field	Delete all values for the specified fields		`<req name="update">`
			`<ent name="Pool"/>`
			`… <value val="" isnull="y"/>…`

Delete Field Value	Delete fields with the specified values		`<req name="update">`
			`<ent name="Pool"/>`
			`<oper name="RemoveFromSet">`

7.2.1 Add Field Value

Description

This operation adds one or more values to the specified multi-value field for the pool identified by the *poolId*.

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

All existing values are retained, and the values specified are inserted. For example, if the current value of a field is a,b,c, and this command is used with value d, after the update the field has the value a,b,c,d.

NOTES:

- If a value being added exists, the request fails.
- The *fieldValue* is case-sensitive. An attempt to add the value a to current field value of a,b,c fails, but an attempt to add the value a is successful and result in the field value being a,b,c,A.
- A request to add field values can also be mixed with a request to update or delete a fields. But, the same field for which an `AddToSet` operation is being performed cannot also be updated or deleted, else the request fails.
- A request to add field values using the `AddToSet` operation can also contain a `RemoveFromSet` operation to delete field values. If both operations are included in the same request, the `AddToSet` is performed before the `RemoveFromSet`, irrespective of the order in which they are supplied.

Prerequisites

A pool with a 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 be present in the field.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="fieldName1"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
[
      <expr><attr name="fieldName2"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
      :
      <expr><attr name="fieldNameX"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
]
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *fieldName*: A user defined field in the pool profile.
- *fieldValueX*: Corresponding field value assigned to *fieldname*.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

NOTE: One or more *fieldValueX* values for a *fieldNameX* can be supplied. To add more than one value, either supply a comma separated list of values, or include multiple <expr> elements for the field.

Response

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

- *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.

  Values: 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 Table 46 for other values.
- *affected*: The number of pools returned. A value of 1 indicates success.

Table 46: Add Field Value Error Codes

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 field is not a valid field in 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 added 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 present in the *Entitlement* field. The request is not 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 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>
```

7.2.2 Get Field

Description

This operation retrieves the values for the specified fields for the pool identified by the specified *poolId*.

NOTE: An entity for the pool can be retrieved by specifying an opaqueDataType corresponding to the interface entity name in the SEC.

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.

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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *fieldNameX*: A user defined field in the pool profile.
- *fieldValueX:* Corresponding field value assigned to *fieldNameX*.
- *opaqueDataTypeX*: A user defined field in the subscriber profile, that represents a transparent or opaque data entity.

  Value is either PoolQuota, PoolState, or PoolDynamicQuota.

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

NOTES:

- At least one *fieldNameX*/*opaqueDataTypeX* field must be requested.
- 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>
```

- *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.

  Values: 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 Table 47 for other values.
- *affected*: The number of pools returned. A value of 1 indicates success.
- *rowValueX*: The value of the requested field (for normal fields, not for opaque/transparent entities).

  NOTE: For multi-value fields, the value can contain a comma separated list of values on a single line. For example: `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>`.

Table 47: Update Field Response Status/Error Codes

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in 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>Prepaid</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 data 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>Prepaid</rv>
      <rv></rv>
    </row>
  </rset>
</req>
```

7.2.3 Update Field

Description

This operation updates fields to the specified values for the pool identified by the specified *poolId*. This operation replaces (sets) the values of the fields, which means that any existing values for the fields are deleted first.

For multi-value fields, all existing values are removed, and only the values 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 is used with value d, after the update the field has the value d (it would not be a,b,c,d).

All fields are updated at one time in the database. All fields and all values must be valid for a successful update. That is, as soon as one error is detected during processing, the request is abandoned and an error is returned. For example, if the third specified field fails validation, then none of the fields are updated.

NOTES:

- If the requested fields are valid, but are not present, they are created.
- An entity for the pool can be replaced by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the XML data blob value in *cdataFieldValue*.
- 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.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.
- If the PSO feature is enabled, a pool cannot be updated with the *Type* field on a Non Pool Host UDR system.

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- Each requested field *fieldName* must be a valid field in the pool profile.
- Each requested *cdataFieldName* 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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *fieldNameX*: A user defined field in the pool profile.
- *fieldValueX:* Corresponding field value assigned to *fieldNameX*.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

- *cdataFieldNameX*: A user defined field in the pool profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC.

  Value is either PoolQuota, PoolState, or 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>
```

- *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.

  Values: 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 Table 48 for other values.
- *affected*: The number of pools updated. A value of 1 indicates success.

Table 48: Update Field Error Codes

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 field is not a valid field in 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
OPER_NOT_ALLOWED	Operation Not Allowed

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 were 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 *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 were updated. The original request is not included.

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

Request 3

A request is made to delete the *Type* field from an enterprise pool that has more than the maximum members allowed for a basic pool. The request is not required in the response.

NOTE: deleting the *Type* field triggers a conversion from an enterprise pool to a basic pool.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
<expr><attr name="Type"/><op value="="/>
      <value val="" isnull=”y”/></expr>
  </set>
  <where>
<expr><attr name="PoolID"/><op value="="/>
<value val="100000"/></expr>
  </where>
</req>
```

Response 3

The request fails. The *error* value indicates the enterprise to basic pool conversion failed because the pool has more members than the maximum threshold for a basic pool. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70052" affected="1"/>
</req>
```

Request 4

A request is made to update the *Type* field. PSO feature is enabled.The *PoolID* falls in a range that is maintained by a different UDR The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
<expr><attr name="Type"/><op value="="/>
      <value val="Enterprise"/></expr>
  </set>
  <where>
<expr><attr name="PoolID"/><op value="="/>
<value val="100000"/></expr>
  </where>
</req>
```

Response 4

The request fails. The error indicates this operation is not allowed on Non Pool Host UDR.

```
<req name="insert" resonly="y">
  <res error="70026" affected="0"/>
</req>
```

7.2.4 Delete Field

Description

This operation the specified fields for the pool identified by *poolId* in the request.

If the field is a multi-value field then all values are deleted. Deletion of a field results in removal of the field from the pool profile (that is, the field is not present, not just the value is empty).

NOTES:

- The field being deleted does not need to have a current value. It can be empty (deleted) and the request succeeds.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied

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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *fieldNameX*: A user defined field in the pool profile.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

Response

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

- *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.

  Values: 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 Table 49 for other values.
- *affected*: The number of pools updated. A value of 1 indicates success.

Table 49: Delete Field Error Codes

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in 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>
```

7.2.5 Delete Field Value

Description

This operation deletes one or more values from the specified field for the pool identified by the *poolId* in the request.

This operation can only be performed 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 field is removed from the pool profile (that is, the XML element is not present).

NOTES:

- The *fieldValue* is case-sensitive. An attempt to remove the value A from a current field value of A,B,C is successful, but an attempt to remove the value a fails.
- A request to delete field values can also be mixed with a request to update or delete a fields. But, the same field for which a *RemoveFromSet* operation is being performed cannot also be updated or deleted, else the request fails.
- A request to delete field values using the *RemoveFromSet* operation can also contain an *AddToSet* operation to add field values. If both operations are included in the same request, the *AddToSet* is performed before the *RemoveFromSet* irrespective of the order in which they are supplied.

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 removed must be present in the field.

Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="fieldName1"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
```

```
[
      <expr><attr name="fieldName2"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
      :
      <expr><attr name="fieldNameX"/>
            <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    </oper>
]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *fieldName*: A user defined field in the subscriber profile.
- *fieldValueX*: Corresponding field value assigned to *fieldname*.

  NOTE: For multi-value fields, the value can contain a comma separated list of values on a single line. For example: `a,b,c`.

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

NOTE: One or more *fieldValueX* values for a *fieldNameX* can be supplied. To remove more than one value, either supply a comma separated list of values, or include multiple <expr> elements for the field.

Response

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

- *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.

  Values: 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 Table 50 for other values.
- *affected*: The number of pools updated. A value of 1 indicates success.

Table 50: Delete Field Value Error Codes

Error Code	Description
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in 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>
```

7.3 Pool Opaque Data Commands

Table 51: Summary of Pool Opaque Data Commands

Command	Description	Keys	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"/>…`

7.3.1 Create Opaque Data

Description

This operation creates 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 in a `<cdata>` construct.

NOTES:

- The opaque data provided in an XML data blob is always checked to be valid XML. If the entity is defined as transparent in the SEC, then the XML data blob is fully validated against the definition in the SEC. If either validation check fails, then the request is rejected.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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 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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value provided in request, and is passed back in the response

  Values: 1 through 4294967295

- *odk* (optional): Indicates that the insert request is converted to an update if the specified opaque data type exists.
- *poolOpaqueDataType*: A user defined type/name for the pool opaque data.

  Value is either PoolQuota, PoolState, or PoolDynamicQuota.

- *cdataFieldValue*: Contents of the XML data blob.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

Response

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

- *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.

  Values: 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 Table 52 for other values.
- *affected*: The number of pools inserted/updated. A value of 1 indicates success.

Table 52: Create Opaque Data Error Codes

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 field is not a valid field in 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 data 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 data 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 data 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>
  <DynamicQuota name="AggregateLimit">
    <Type>pass</Type>
    <InstanceId>15678</InstanceId>
    <Priority>4</Priority>
    <InitialTime>135</InitialTime>
```

```
    <InitialTotalVolume>2000</InitialTotalVolume>
    <InitialInputVolume>1500</InitialInputVolume>
    <InitialOutputVolume>500</InitialOutputVolume>
    <InitialServiceSpecific>4</InitialServiceSpecific>
    <activationdatetime>2015-03-09T11:20:32</activationdatetime>
    <expirationdatetime>2015-04-09T11:20:32</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 data 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 data blob is supplied whole. The Pool 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 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 data blob is supplied whole. The Pool has an associated PoolQuota. The request includes the *odk* attribute, indicating that the PoolQuota is updated if it exists. 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="600000"/></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>
```

7.3.2 Get Opaque Data

Description

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 XML data blob for the requested pool opaque data.

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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolOpaqueDataType*: A user defined type/name for the pool opaque data.

  Value is either PoolQuota, PoolState, or PoolDynamicQuota.

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

Response Content

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

```
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>
          <![CDATA[cdataRowValue]]>
      </rv>
    </row>
  </rset>
]
</req>
```

- *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.

  Values: 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 Table 53 for other values.
- *affected*: The number of pools returned. A value of 1 indicates success.
- *cdataRowValue*: Contents of the XML data blob.

***NOTES*:**

- 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 (that is, XML data blob).
- If the PSO feature is enabled, and the UDR is the Non Pool Host UDR for the specified pool, then empty entity data is returned, as pool entity data is only stored on the Pool Host UDR.

Table 53: Get Opaque Data Error Codes

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 get the *PoolQuota* opaque data for a pool. 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>
```

Request 2

A request is made to get the *PoolQuota* opaque data for a PSO pool, where the UDR instance receiving the request is not the Pool Host UDR for the pool in the request. 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 2

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

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

7.3.3 Update Opaque Data

Description

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

NOTES:

- The opaque data provided in an XML data blob is always checked to be valid XML. If the entity is defined as transparent in the SEC, then the XML data blob is fully validated against the definition in the SEC. If either validation check fails, then the request is rejected.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolOpaqueDataType*: A user defined type/name for the pool opaque data.

  Value is either PoolQuota, PoolState, or PoolDynamicQuota.

- *cdataFieldValue*: Contents of the XML data blob.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

Response

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

- *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

  Values: 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 Table 54 for other values.
- *affected*: The number of pools updated. A value of 1 indicates success.

Table 54: Update Opaque Data Error Codes

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 field is not a valid field in 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 data 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>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 1

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

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

7.3.4 Delete Opaque Data

Description

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.

NOTES:

- The deletion of a non-existent opaque data type (but that is defined in the SEC) is not considered as an error.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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>
```

- *keyName*: A user defined field identified as key for the subscriber.
- *keyValue*: A key value identifying the subscriber.
- *poolOpaqueDataType*: A user defined type/name for the pool opaque data.

  Value is either PoolQuota, PoolState, or PoolDynamicQuota.

  NOTE: The data is deleted by setting an empty field value, and also specifying the attribute `isnull="y"`

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

Response

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

- *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

  Values: 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 Table 55 for other values.
- *affected*: The number of pools deleted. A value of 1 indicates success.

Table 55: Delete Opaque Data Error Codes

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 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 *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="select" resonly="y">
  <res error="0" affected="1"/>
</req>
```

7.4 Pool Data Row Commands

A transparent data entity may contain data that is organized in rows. An example of a row is a specific quota in 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*.

NOTE: Pool data row commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an `OPER_NOT_ALLOWED` error being returned.

Table 56: Summary of Pool Data Row Commands

Command	Description	Keys	Command Syntax

Create Row	Insert data row into transparent data of the specified type.	(PoolID and Row Identifier) Or (PoolID, Row Identifier and Instance 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> …`
			`[`
			`  <expr>`
			`  <attr name= "instanceFieldName">`
			`  <value val="instanceFieldValue">`
			`  </expr> …`
			`]`

Command	Description	Keys	Command Syntax
Delete Row	Delete data row in transparent data of the specified type.		<req name="delete"> <ent name="*entityName*"/> … <expr> <attr name="*rowIdName*"> <value val="*rowIdValue*"> </expr> … [<expr> <attr name="*instanceFieldName*"> <value val="*instanceFieldValue*"> </expr> …]

7.4.1 Create Row

Description

This operation creates a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowIdName*, and the row identifier value is specified in *rowIdValue*. All *fieldNameX* fields specified are set in the row.

NOTES:

- The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to update an existing row called DAYPASS is successful, and two rows called DayPass and DAYPASS are present.
- If the transparent entity specified in entityName does not exist for the pool, it is created.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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 two different formats.

1. The poolId in the <set> element.
2. The poolId in the <where> 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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *odk* (optional): Indicates that the insert request is converted to an update if the data row for the specified entity exists.
- *entityName*: A user defined entity type/name for the transparent data.
  - Value is PoolQuotaEntity for the PoolQuota transparent data.
  - Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *fieldNameX*: A user defined field in 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. For example: `a,b,c`.

NOTES:

- 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 PoolQuota 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.

- If the `odk="yes"` attribute is set (implying that an update is made if the row exists), then if multiple rows exist for the specified *rowIdName/rowIdValue*, the request fails 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>
```

- *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.

  Values: 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 Table 57 for other values.
- *affected*: The number of pools updated. A value of 1 indicates success.

Table 57: Create Row Error Codes

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 field is not a valid field in 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 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 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>
```

Request 3

A request is made to create a data row in the *PoolDynamicQuotaEntity* (PoolDynamicQuota) data. The *PoolID* key is supplied, which references the pool. The data row identifier field is *name*, and the value is *PDQ1*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="400000"/></expr>
    <expr><attr name="name"/><value val="PDQ1"/></expr>
    <expr><attr name="Type"/><value val="top-up"/></expr>
    <expr><attr name="InstanceId"/><value val="15678"/></expr>
    <expr><attr name="Priority"/><value val="4"/></expr>
    <expr><attr name="InitialTime"/><value val="135"/></expr>
    <expr><attr name="InitialTotalVolume"/><value val="2000"/></expr>
    <expr><attr name="InitialInputVolume"/><value val="1500"/></expr>
    <expr><attr name="InitialOutputVolume"/><value val="500"/></expr>
    <expr><attr name="InitialServiceSpecific"/><value val="4"/></expr>
    <expr><attr name="activationdatetime"/><value val="2015-05-22T00:00:00-05:00"/></expr>
    <expr><attr name="expirationdatetime"/><value val="2015-05-29T00:00:00-05:00"/></expr>
    <expr><attr name="InterimReportingInterval"/><value val="100"/></expr>
    <expr><attr name="Duration"/><value val="10"/></expr>
  </set>
</req>
```

Response 3

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>
```

7.4.2 Get Row

Description

This operation retrieves a data rows for the pool identified by the *poolId*.

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

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

NOTES

- The *rowIdValue* is case-sensitive. If a row called DayPassexists, then an attempt to retrieve a row called DayPass is successful, but an attempt to retrieve a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to retrieve a row with a field with the value Data is successful, but an attempt to retrieve a row with a field that has the value DATA fails.

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="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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is PoolQuotaEntity for the PoolQuota transparent data.
  - Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota 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>
```

- *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.

  Values: 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 Table 58 for other values.
- *affected*: The number of pools returned.

  A value of 1 or more is expected for success, whether or not a row was found

- *cdataRowValueN*: Contents of the XML data blob containing one requested/matching data row.

NOTES:

- 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.
- If the transparent entity exists, but the row value was not found, then the `<rv>` (row value) indicates that the row does not exist by containing the value `<rv null="y"/>`.
- If the PSO feature is enabled, and the UDR is the Non Pool Host UDR for the specified pool, then empty entity data is returned, as pool entity data is only stored on the Pool Host UDR.

Table 58: Get Row Error Codes

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 four 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 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 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>
```

Request 7

A request is made to get the *PDQ1* data row from the PoolDynamicQuota data, with the *InstanceId* value of *11223344*. The PoolDynamicQuota data contains four rows called *PDQ1*. Two with *InstanceId* of *11223344*, one

with an *InstanceId* of *99887766*, and one with an *InstanceId* of *55556677*. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
          <value val="4000000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="PDQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response 7

The request is successful, and the 2 PoolDynamicQuota data rows with an *InstanceId* 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"?>
        <definition>
          <version>1</version>
          <DynamicQuota name="PDQ1">
            <Type>top-up</Type>
            <InstanceId>11223344</InstanceId>
            <Priority>4</Priority>
            <InitialTime>135</InitialTime>
            <InitialTotalVolume>2000</InitialTotalVolume>
            <InitialInputVolume>1500</InitialInputVolume>
            <InitialOutputVolume>500</InitialOutputVolume>
            <InitialServiceSpecific>4</InitialServiceSpecific>
            <activationdatetime>2015-05-22T00:00:00-05:00</activationdatetime>
            <expirationdatetime>2015-05-29T00:00:00-05:00</expirationdatetime>
            <InterimReportingInterval>100</InterimReportingInterval>
            <Duration>10</Duration>
          </DynamicQuota>
        </definition>]]>
      </rv>
    </row>
    <row>
      <rv>
       <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <definition>
          <version>1</version>
          <DynamicQuota name="DQ1">
            <Type>pass</Type>
            <InstanceId>11223344</InstanceId>
            <Priority>2</Priority>
            <InitialTime>135</InitialTime>
            <InitialTotalVolume>1000</InitialTotalVolume>
            <InitialInputVolume>500</InitialInputVolume>
            <InitialOutputVolume>500</InitialOutputVolume>
            <InitialServiceSpecific>4</InitialServiceSpecific>
            <activationdatetime>2015-05-22T00:00:00-05:00</activationdatetime>
            <expirationdatetime>2015-05-29T00:00:00-05:00</expirationdatetime>
            <InterimReportingInterval>100</InterimReportingInterval>
            <Duration>10</Duration>
          </DynamicQuota>
        </definition>]]>
      </rv>
    </row>
```

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

Request 8

A request is made to get the *Weekday* data row in the PoolQuota data for a PSO pool, where the UDR instance receiving the request is not the Pool Host UDR for the pool in the request. 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 8

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

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

7.4.3 Delete Row

Description

This operation deletes a data row for the pool identified by the *poolId*.

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

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

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to delete a row called DayPass is successful, but an attempt to delete a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to delete a row with a field that has the value Data is successful, but an attempt to delete a row with a field that has the value DATA fails.
- The deletion of a non-existent data row is not considered an error.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is PoolQuotaEntity for the PoolQuota transparent data.
  - Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName.*

Response

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

- *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.

  Values: 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 Table 59 for other values.
- *affected*: A value of 1 indicates that the rows existed, or that the row did not exist

Table 59: Delete Row Error Codes

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 in 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 was 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 delete 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 delete 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 from the PoolQuota data, with the *cid* value of *11223344*. The PoolQuota data contains two rows called *Q4*. One with *cid* of *11223344*, the other with a *cid* of *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 was 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 *PDQ1* data row from the PoolDynamicQuota data, with the *Type* value of *pass*. The PoolDynamicQuota data contains two rows called *PDQ1*. One with a *Type* of *pass* the other with a *Type* of *quota*. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
```

```
  <where>
    <expr><attr name="PoolID"/><op value="="/>
          <value val="500000"/></expr>
<expr><attr name="name"/><op value="="/>
      <value val="PDQ1"/></expr>
<expr><attr name="Type"/><op value="="/>
      <value val="pass"/></expr>
  </where>
</req>
```

Response 5

The request is successful, and the data row in the PoolDynamicQuota data was deleted that matched the *Type*. The original request is not included.

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

7.5 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 in 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*.

NOTE: Pool data row field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an `OPER_NOT_ALLOWED` error being returned.

Table 60: Summary of Pool Data Row Field Commands

Command	Description	Keys	Command Syntax

Get Row Field	Retrieve values for the specified fields	(PoolID and Row Identifier and Field name) Or (PoolID and Row Identifier, Instance Identifier and Field name)	`<req name="select">`
			`  <ent name="entityName"/>`
			`  … <expr>`
			`  <attr name="rowIdName">`
			`  <value val="rowIdValue">`
			`  </expr> …`

Update Row Field	Update fields to the specified values		`<req name="update">`
			`  <ent name="entityName"/>`
			`  … <expr>`
			`  <attr name="rowIdName">`
			`  <value val="rowIdValue">`
			`  </expr> …`
			`[`
			`  <expr>`
			`  <attr name="instanceFieldName">`
			`  <value val="instanceFieldValue">`
			`  </expr> …`
			`]`

Command	Description	Keys	Command Syntax
Delete Row Field	Delete all values for the specified fields		<req name="update"> <ent name="*entityName*"/> … <expr> <attr name="*rowIdName*"> <value val="*rowIdValue*"> </expr> … [<expr> <attr name="*instanceFieldName*"> <value val="*instanceFieldValue*"> </expr> …]

7.5.1 Get Row Field

Description

This operation retrieves a fields in a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowIdName*, and the row identifier value is specified in *rowIdValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field names are specified in *fieldNameX*.

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

NOTES:

- If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and empty field data is returned.
- The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to get a field in a row called DayPass is successful, but an attempt to get a field in a row called DayPass fails.
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to delete a row with a field that has the value Data is successful, but an attempt to delete a row with a field that has the value DATA fails.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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 in the transparent data should exist for the subscriber.

The field names 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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is PoolQuotaEntity for the PoolQuota transparent data.
  - Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data.
- *fieldNameX*: A user defined field in the data row.
- *fieldValueX* (optional): Corresponding field value assigned to *fieldNameX*.

  NOTE: For multi-value fields, the value can contain a comma separated list of values.

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota 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>
```

- *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.

  Values: 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 Table 61 for other values.
- *affected*: The number of pools returned.
  - A value of 1 indicates that the specified row exists (whether or not the field was found).
  - A value of 0 indicates that the row does not exist.
- *rowValue*: The value of the requested field.

  NOTE: For multi-value fields, the value can contain a comma separated list of values on a single line. For example: `a,b,c`.

NOTES:

- 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>`.
- If the PSO feature is enabled, and the UDR is the Non Pool Host UDR for the specified pool, then empty entity data is returned, as pool entity data is only stored on the Pool Host UDR.

Table 61: Get Row Field Error Codes

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in 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" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>4000</rv>
    </row>
  </rset>
</req>
```

Request 4

A request is made to get the *InstanceId, InitialTotalVolume and InitialInputVolume* fields in the *PDQ3* data row of the PoolDynamicQuota data, with the *InstanceId* value of 15678. The PoolDynamicQuota data contains two rows called *PDQ3*. One with *InstanceId* of *15570*, the other with an *InstanceId* of *15678*. The request is not required in the response.

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

```
  <select>
    <expr><attr name="InstanceId"/></expr>
    <expr><attr name="InitialTotalVolume"/></expr>
    <expr><attr name="InitialInputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
          <value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/>
          <value val="PDQ3"/></expr>
    <expr><attr name="InstanceId"/><op value="="/>
          <value val="15678"/></expr>
  </where>
</req>
```

Response 4

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

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>15678</rv>
      <rv>2000</rv>
      <rv>1500</rv>
    </row>
  </rset>
</req>
```

Request 5

A request is made to get the *inputVolume* field in the *Q1* data row of the PoolQuota data for a PSO pool, where the UDR instance receiving the request is not the Pool Host UDR for the pool in the request. 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 5

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

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

7.5.2 Update Row Field

Description

This operation updates a fields in a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowIdName*, and the row identifier value is specified in *rowIdValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field names are specified in *fieldNameX*. If the specified fields are valid, but do not exist, they are created.

If more than one row matches the requested *rowIdName*/*rowIdValue* and *instanceFieldName*/*instanceFieldValue* (if specified), then the update request fails.

NOTES:

- If the specified row does not exist, the request fails.
- If the requested fields are valid, but are not present, they are created.
- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to update a field in a row called DayPass is successful, but an attempt to update a field in a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to delete a row with a field that has the value Data is successful, but an attempt to delete a row with a field that has the value DATA fails.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

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 in the transparent data should exist for the pool.

The field names 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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is PoolQuotaEntity for the PoolQuota transparent data.
  - Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data.
- *fieldNameX*: A user defined field in 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. For example: `a,b,c`.

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName.*

Response

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

- *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.

  Values: 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 Table 62 for other values.

- *affected*: The number of pools updated. A value of 1 indicates success.

Table 62: Update Row Field Error Codes

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 field is not a valid field in 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
MULTIPLE_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 *Q6*. 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>
```

Request 4

A request is made to update the *InitialTotalVolume* and *InitialInputVolume* fields in the *PDQ1* data row of the PoolDynamicQuota data where *InstanceId* is 15678. The *PDQ1* data row exists in the PoolDynamicQuota data, but there are two rows called *PDQ1* one with *InstanceId* of *15570*, the other with an *InstanceId* of 15678. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <set>
    <expr><attr name="InitialTotalVolume"/><value val="1000"/></expr>
    <expr><attr name="InitialInputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
          <value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/>
          <value val="PDQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/>
          <value val="15678"/></expr>
```

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

Response 4

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

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

7.5.3 Delete Row Field

Description

This operation deletes a fields in a data row for the pool identified by the *poolId*.

The data row identifier field is specified in *rowIdName*, and the row identifier value is specified in *rowIdValue*. An additional field can be specified to indicate a unique row in *instanceFieldName*/*instanceFieldValue*. The field names are specified in *fieldNameX*.

If more than one row matches the requested *rowIdName/rowIdValue* and *instanceFieldName*/*instanceFieldValue* (if specified), then the delete request fails.

NOTES:

- If the specified row does not exist, the request fails. 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.
- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to delete a field in a row called DayPass is successful, but an attempt to delete a field in a row called DayPass fails.
- The *instanceFieldValue* is case-sensitive. If a field contains the value Data, then an attempt to delete a field in a row with a field that has the value Data is successful, but an attempt to delete a field in a row with a field that has the value DATA fails.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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.

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

The field names 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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is PoolQuotaEntity for the PoolQuota transparent data.
  - Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data.
- *fieldNameX*: A user defined field in 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. For example: `a,b,c`.

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 9999999999999999999

- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.
  - Value is name for Quota transparent data.
  - Value is name for PoolDynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
  - Value is `cid` or `Type` for the PoolQuota transparent data.
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName.*

Response

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

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

- *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.

  Values: 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 Table 63 for other values.
- *affected*: The number of pools updated. A value of 1 indicates that the row existed and the field was deleted. A value of 0 indicates that the field did not exist.

Table 63: Delete Row Field Error Codes

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_UNDEFINED	Field Not Defined. The field is not a valid field in 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
MULTIPLE_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>
```

Request 4

A request is made to delete the *InitialInputVolume* field in the *PDQ1* data row of the PoolDynamicQuota data with the *InstanceId 11223344*. The *PDQ1* data row exists in the PoolDynamicQuota data, and is there are two rows called *PDQ1*, one with *InstanceId 11223344* and one with *InstanceId 99887766*. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <set>
    <expr><attr name="InitialInputVolume"/><op value="="/>
          <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
          <value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="PDQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

Response 4

The request is successful, and the `InitialInputVolume` field in the *PDQ1* data row in the PoolDynamicQuota data was deleted for the row with the specified *InstanceId*. The original request is not included.

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

Request 5

A request is made to delete the *inputVolume* field in the *PQ1* data row of the PoolQuota data. The *PQ1* data row exists in the PoolQuota data, there is only one row called *PQ1* and the *inputVolume* field does not exist. 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="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="PQ1"/></expr>
  </where>
</req>
```

Response 5

The request is successful and the original request is not included.

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

7.6 Pool Data Field Commands

A transparent data entity may contain data that is organized in fields where each field is defined as a name value pair in an element. For example, the PoolState entity has a `<name>` element for the name and a `<value>` element for the value in a `<property>` element.

```
<property>
  <name>fieldName</name>
  <value>fieldValue</value>
</property>
```

The data commands allow operations (create/retrieve/update/delete) at the field level. The required field is identified in the request by the *fieldName*.

NOTE: Pool data commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an `OPER_NOT_ALLOWED` error being returned.

Table 64: Summary of Pool Data Commands

Command	Description	Keys	Command Syntax
Create Data Field	Create/update data field in transparent data of the specified type.	PoolID and Field Name	`<req name="insert"> <ent name="entityName"/> … <expr> <attr name="fieldName"> <value val="fieldValue"> </expr> …`
Get Data Field	Retrieve data field from transparent data of the specified type.		`<req name="select"> <ent name="entityName"/> … <expr> <attr name="fieldName"> <value val="fieldValue"> </expr> …`
Update Data Field	Update data field in transparent data of the specified type.		`<req name="update"> <ent name="entityName"/> … <expr> <attr name="fieldName"> <value val="fieldValue"> </expr> …`
Delete Data Field	Delete data field in transparent data of the specified type.		`<req name="update"> <ent name="entityName"/> … <expr> <attr name="fieldName"> <value val="fieldValue"> </expr> …`

7.6.1 Create Data Field

Description

This operation creates or updates an existing field in a transparent data for the pool identified by the *poolId*.

The field name is specified in *fieldNameX*, and the field value is specified in *fieldValueX*.

If the specified field does not exist, it is created. If the field does exist, it is updated/replaced only if the optional *odk* flag is set to *yes*.

NOTES:

- The *fieldName* is not case-sensitive. If a field existed called mcc, then an attempt to create/update an existing field called MCC is successful.
- If the transparent entity specified in entityName does not exist for the pool, it is created.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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 two different formats.

1. The poolId in a <set> element.
2. The poolId in a <where> 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="fieldName1"/><value val="fieldValue"/></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="fieldName"/><value val="fieldValue"/></expr>
[
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
              :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
]

  <where>
    <expr><attr name="PoolID"/><value val="poolId"/></expr>
  </where>
  </set>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *odk* (optional): Indicates that the insert request is converted to an update if the field for the specified entity exists.
- *entityName*: A user defined entity type/name for the transparent data.

  Value is PoolStateEntity for the PoolState transparent data.

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

- *fieldNameX*: The requested user defined field in the pool profile.

  For the PoolState entity, this corresponds to a property in the entity.

  The *fieldNameX* case is stored exactly as it was sent in the request (This means the original case stored changes if an update is received)

- *fieldValueX:* Corresponding field value assigned to *fieldNameX*.

Response

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

- *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.

  Values: 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 Table 65 for other values.
- *affected*: The number of pools updated. A value of 1 indicates success.

Table 65: Create Data Field Error Codes

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 field is not a valid field in 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 create a property in the *PoolState* transparent data for a pool. The property name is *mcc* and the property value is *315*. The pool does not have an existing *PoolState* property called *mcc*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

Response 1

The request is successful, and the property *mcc* with value *315* 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 property in the *PoolState* transparent data for a pool, using the alternate request format. The property name is *mcc* and the property value is *315*. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
  <where>
<expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response 2

The request is successful, and the property 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 property in the *PoolState* transparent data for a pool. PoolState is a valid data type, but the pool does not have this entity type. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
<expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

Response 3

The request is successful, and the property as well as the PoolState entity is 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 property in the *PoolState* transparent data for a pool. The property name is *mcc* and the property value is *315*. The *odk* attribute is included requesting the data be updated if it exists. The request is not required in the response.

```
<req name="insert" resonly="y" odk="yes">
  <ent name="PoolStateEntity"/>
  <set>
<expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

Response 4

The request is successful, and the existing property 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 property in the *PoolState* transparent data for a pool. The property name is *mcc* and the property value is *315*. (The property exists) The *odk* attribute is not included (requests the data be updated if it exists.) The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

Response 5

The request fails. The *error* value indicates the *mcc* property exists, and the *affected* rows are 0. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="70028" affected="0"/>
</req>
```

7.6.2 Get Data Field

Description

This operation retrieves a data field in a transparent data for the pool identified by the *poolId*.

All fields that match the requested *fieldNameX* are returned.

If more than one field matches the requested *fieldNameX*, then all matching fields are returned.

The transparent data field is specified in *fieldNameX*.

NOTES

- If the specified field does not exist, `null="y"` is returned.
- The *fieldNameX* is not case-sensitive. If a field called mcc exists, then an attempt to get a field name called MCC is successful.

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 field with the identifier in the transparent data should exist for the pool.

Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <select>
    <expr><attr name="fieldName1"/></expr>
[
    <expr><attr name="fieldName2"/></expr>
            :
    <expr><attr name="fieldNameN"/><expr>
]
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.

  Value is PoolStateEntity for the PoolState transparent data.

- *fieldNameX*: The requested user defined field in the pool profile.

  For the PoolState entity, this corresponds to a property in the entity.

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
<     <rv>fieldValue1</rv> | <rv null="y"> | <rv></rv> >
[
<     <rv>fieldValue2</rv> | <rv null="y"> | <rv></rv> >
      :
<     <rv>fieldValueN</rv> | <rv null="y"> | <rv></rv> >
]
    </row>
  </rset>
]
</req>
```

- *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.

  Values: 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 Table 66 for other values.
- *affected*: The number of pools returned. A value of 1 is expected if the specified data exists (whether or not the field was found). A value of 0 is expected if the data does not exist
- *fieldValue*: The value of the requested field.

NOTES

- The `<rset>` (row set) element is optional. It is only present if the request was successful. One <row> element is returned per matching data. One `<rv>` (field value) element exists for every *fieldNameX* supplied in the original request. The `<rv>` elements are ordered the same as the *fieldNameX* properties 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>`.
- If the PSO feature is enabled, and the UDR is the Non Pool Host UDR for the specified pool, then empty data is returned, as pool entity data is only stored on the Pool Host UDR.

Table 66: Get Data Field Error Codes

Error Code	Description
INTF_ENTY_NOT_FOUND	Interface Entity Not Found
FIELD_UNDEFINED	Field Not Defined. The field is not a valid field in 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
REG_DATA_NOT_FOUND	Register Data Not Found

Examples

Request 1

A request is made to get three *PoolState* properties from the *PoolState* transparent data for a pool. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolStateEntity"/>
  <select>
<expr><attr name="mcc"/></expr>
<expr><attr name="expire"/></expr>
    <expr><attr name="duration"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response 1

The request is successful, two values are returned and one was not set. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>315</rv>
      <rv>2015-02-09T11:20:32</rv>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

Request 2

A request is made to get the *mcc* property in the *PoolState* transparent data for a pool. The *mcc* property is not set. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolStateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response 2

The request is successful, and the property is indicated to not be set. 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 3

A request is made to get the *mcc* property in the *PoolState* transparent data for a pool. The *PoolState* Entity does not exist. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolStateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response 3

The request fails. The *error* value indicates the *PoolState* entity does not exist, and the *affected* subscribers are 0. The original request is not included.

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

Request 4

A request is made to get three *PoolState* properties from the *PoolState* transparent data for a PSO pool, where the UDR instance receiving the request is not the Pool Host UDR for the pool in the request. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolStateEntity"/>
  <select>
<expr><attr name="mcc"/></expr>
<expr><attr name="expire"/></expr>
    <expr><attr name="duration"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response 4

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

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

7.6.3 Update Data Field

Description

This operation updates an existing field in a transparent data for the pool identified by the *poolId*.

The field name is specified in *fieldNameX*, and the field value is specified in *fieldValueX*.

If more than one existing fields matches the requested *fieldNameX*, then the update request fails.

NOTES:

- If the requested fields are valid, but are not present, they are created.
- The *fieldNameX* is not case-sensitive. If a field called mcc exists, then an attempt to update a field called MCC is successful.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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

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

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.

Value is PoolStateEntity for the PoolState transparent data.

- *fieldNameX*: The requested user defined field in the pool profile.

  For the PoolState entity, this corresponds to a property in the entity.

  The *fieldNameX* case is stored exactly as it was sent in the request. This means the original case stored changes if an update is received.

- *fieldValueX:* Corresponding field value assigned to *fieldNameX*
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

Response

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

- *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.

  Values: 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 Table 67 for other values.
- *affected*: The number of pools updated. A value of 1 indicates success.

Table 67: Update Data Field Error Codes

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 field is not a valid field in 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

Examples

Request 1

A request is made to update the *mcc* and *approved* properties of the *PoolState* transparent data for a pool. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="mcc"/><value val="302"/></expr>
    <expr><attr name="approved"/><value val="no"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response 1

The request is successful, and *mcc* and *approved* property values are updated. The original request is not included.

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

Request 2

A request is made to update the *approved* property in the *PoolState* transparent data for a pool. The *PoolState* Entity does not exist for the pool. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="approved"/><value val="no"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req
```

Response 2

The request fails because the *PoolState* entity does not exist for the pool. The original request is not included.

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

7.6.4 Delete Data Field

Description

This operation deletes a data field in a transparent data for the pool identified by the *poolId*.

The field identifier is specified in *fieldNameX*.

If more than one data field matches the requested *fieldNameX*, then all matching fields are deleted.

NOTES:

- The deletion of a non-existent field is not considered an error.
- The *fieldNameX* is not case-sensitive. If a field called mcc exists, then an attempt to delete a field called MCC is successful.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

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

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

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.

  Value is PoolStateEntity for the PoolState transparent data.

- *fieldNameX*: The requested user defined field in the pool profile.

  For the PoolState entity, this corresponds to a property in the entity.

- *fieldValueX:* Corresponding field value assigned to *fieldNameX*.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

Values: 1 through 9999999999999999999

Response

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

- *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.

  Values: 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 Table 68 for other values.
- *affected*: The number of pools updated. A value of 1 indicates that the field was deleted. A value of 0 indicates that the field did not exist.

Table 68: Delete Data Field Error Codes

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_UNDEFINED	Field Not Defined. The field is not a valid field in 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

Examples

Request 1

A request is made to delete the mcc property of the PoolState transparent data for a pool. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="mcc"/><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 mcc property 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 mcc property in PoolState transparent data for a pool. The mcc property does not exist for the PoolState data. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

Response 2

```
The request is successful and the original request is not included.
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

7.7 Additional Pool Commands

Table 69: Summary of Additional Pool Commands

Command	Description	Keys	Command Syntax
Add Member to Pool	Add subscriber to a Pool	PoolID and (MSISDN, IMSI, NAI or AccountId)	`<req name="operation"><oper name="AddPoolMember">`
Remove Member from Pool	Remove subscriber from a Pool		`<req name="operation"><oper name="DelPoolMember">`
Get Pool Members	Retrieve pool member subscribers by PoolID	PoolID	`<req name="operation"><oper name="GetPoolMembers">`
Get Pool by Member (Key)	Retrieve PoolID for specified member subscriber	MSISDN, IMSI, NAI or AccountId	`<req name="operation"><oper name="GetPoolID">`
Get All Pool Members	Retrieve pool member subscribers from all local or local/remote systems by PoolID	PoolID	`<req name="operation"><oper name="GetAllPoolMembers">`

7.7.1 Add Member to Pool

Description

This operation adds one or more Subscribers to a Pool.

When the PSO flag is enabled, a pool member added to a pool on a Non Pool Host UDR is not treated as a full pool member until the pool is created or pool profile updated on the Pool Host UDR when the connection between the two is active. In this situation, the subscriber is treated as a pool member by SOAP commands, but not as a pool member by Diameter Sh commands.

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 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="subKeyName25"/><op value="="/>
          <value val="subKeyValue25"/></expr>
]
  </oper>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

- *subKeyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *subKeyValueX*: Corresponding key field value assigned to *keyNameX*.

***NOTES*:**

- Up to 25 subscribers can be added in one request.

- On a low capacity server configuration, the maximum number of *AddPoolMember* requests that can be included in a <tx> is 3, if each request adds 25 members. Although nothing in the software limits the number of requests to 3, this is the recommended value.
- 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 fails.
- If any subscriber specified is a member of a pool, the request fails.

Response

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

- *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.

  Values: 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 Table 70 for other values.
- *affected*: The number of pools updated. A value of 1 or more is expected for success.

Table 70: Add Member to Pool Response Status/Error Codes

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 a member of a pool
POOL_NOT_FOUND	Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist
MAX_MEMBERS_BASIC_POOL	The maximum number of subscribers in a basic pool has been exceeded.

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 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 that the 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 that the 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 (either local or remote). Both the pool and the subscriber exist. The subscriber is 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 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 basic 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 basic 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="70051" 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 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="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 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="1"/>
</req>
```

Request 7

A request is made to add a subscriber to a PSO pool, where the UDR instance receiving the request is not the Pool Host UDR for the pool in the request. The Non Pool Host UDR pool has 25 subscribers. The Pool profile from the Pool Host UDR is not received on the Non Pool Host UDR. 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 7

The request is successful, and the subscriber is added to the pool as no limit is enforced on a Non Pool Host UDR as long as the pool profile from the Pool Host UDR is not received. The original request is not included.

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

7.7.2 Remove Member from Pool

Description

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="subKeyName25"/><op value="="/>
          <value val="subKeyValue25"/></expr>
]
  </oper>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - Y

    only provide the result, do not include the original request

  - n

    include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

- *subKeyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *subKeyValueX*: Corresponding key field value assigned to *keyNameX*.

NOTES

- Up to 25 subscribers can be removed in one request.
- On a low capacity server configuration, the maximum number of *DelPoolMember* requests that can be included in a `<tx>` is 3, if each request deletes 25 members. Although nothing in the software limits the number of requests to 3, this is the recommended value.
- If any subscriber specified is not a member of the pool, the request fails.

Response

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

- *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.

  Values: 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 Table 71 for other values.
- *affected*: The number of pools updated. A value of 1 or more is expected for success.

Table 71: Remove Member from Pool Error Codes

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
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" resonly="y">
  <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>
```

7.7.3 Get Pool Members

Description

This operation gets the list of Subscriber members of a Pool by *poolId*. This operation only gets the list of subscribers and addresses for a local pool on the UDR instance where the request was received, regardless if the pool is a PSO pool or not.

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>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

Response

```
<req name="operation" [resonly="resonly"] [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>
```

- *resonly* (optional): The *resonly* value from the original XML request, if supplied.
- *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.

  Values: 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 Table 72 for other values.
- *affected*: The number of Pools returned. A value of 1 indicates success.
- *keyNameX*: A key field for the member subscriber.

  Value is either IMSI, MSISDN, NAI, or 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. Only a single <row> element is returned, with one `<rv>` (row value) element.
- 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 user identities for that subscriber, one user identity per `<id>` element. There can be one or many `<id>` elements per `<member>` element.
- The format of this response can be returned in a legacy SPR compatible mode if UDR is configured to do so. See Appendix C for more details.

Table 72: Get Pool Members Error Codes

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"></res>
  <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="1"/>
</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 that the 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>
```

7.7.4 Get PoolID

Description

This operation gets the PoolID related to a subscriber, based on the given user identities of the subscriber.

Prerequisites

A subscriber with the keys of the *keyNameX*/*keyValueX* values 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="keyName1"/><op value="="/>
          <value val="keyValue1"/></expr>
[
    <expr><param name="keyName2"/><op value="="/>
          <value val="keyValue2"/></expr>
    :
    <expr><param name="keyNameN"/><op value="="/>
          <value val="keyValueN"/></expr>
]
  </oper>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *keyNameX*: A key field in the subscriber profile.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

Response

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

- *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.

  Values: 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 Table 73 for other values.
- *affected*: The number of subscribers returned. A value of 1 indicates success.
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 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.

Table 73: Get PoolID Error Codes

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. The subscriber is not a member of a pool

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 was 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 fails. The *error* value indicates that the subscriber is not a member of a pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70025" 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 *affected* rows are 0. The original request is not included.

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

Request 4

A request is made to get the PoolID for a subscriber. Both *MSISDN* and *AccountId* keys are supplied, and reference the same 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="15141234567"/></expr>
    <expr><param name="AccountId"/><op value="="/>
          <value val="111222333"/></expr>
  </oper>
</req>
```

Response 4

The request is successful, and the PoolID value was 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>
```

7.7.5 Get All Pool Members

Description

This operation has the option to get the list of Subscriber members of a Pool by *poolId*, from the local or from the local and all remote UDR systems. The Pool Spanning UDRs feature needs to be enabled in order to retrieve remote pool members. This command can be used on a single local Pool Host UDR as well, when the Pool Spanning UDRs feature is not enabled.

NOTE: The addressList values (IMSI and ALL) are not case-sensitive.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/>
      <value val="poolId"/></expr>
[   <expr><param name="AddressList"/><op value="="/>
    <value val="addressList"/></expr> ]
[   <expr><param name="PSO"/><op value="="/>
      <value val="pso"/></expr> ]
  </oper>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

- *addressList (optional):* The requested address type of the subscriber for a pool is returned in the response

  Values:

  - IMSI

    Only pool members with an IMSI address type are returned in the response.

  - All

    All pool members are returned in the response (default).

- *pso (optional):* Indicates whether the response includes pool members across all UDR instances in the pool network or only the UDR instance where the request is received

  Values:

  - yes

    Response includes pool members across all UDR instances in the pool network.

  - no

    Response only includes pool members on the local UDR instance (default).

NOTES:

- If a timeout occurs while waiting for a response from a remote UDR instance, then the UDR fails the request with a REQUEST_TIMEOUT error.
- If a request fails to be sent to a remote UDR instance, then the UDR fails the provisioning request with a REQUEST_TIMEOUT.
- This request can be used even when the Pool Spanning UDRs feature is not enabled.

Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[ originalXMLRequest ]
  <res error="error" affected="affected"/>
[ <rset>
    <row>
      <rv>
 <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
   <members>
[           <member>
[              <udrId>udrId</udrId> ]
         <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>
[              <udrId>udrId</udrId> ]
         <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>
[              <udrId>udrId</udrId> ]
          <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>
```

- *resonly* (optional): The *resonly* value from the original XML request, if supplied.
- *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.

  Values: A string with 1 to 4096 characters.

- *udrId* (optional): A subscriber key range that is hosted by each UDR in the remote pool network. The response only include sthe *udrId* if the Pools Spanning UDRs feature is enabled.

  Value: 1 to 10 digits.

- *id* (optional): The *id* value from the original XML request, if supplied.
- *error*: Error code indicating outcome of request. 0 means success, see Table 74 for other values.
- *affected*: The number of Pools returned. A value of 1 indicates success.
- *keyNameX*: A key field for the member subscriber.

  Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

NOTES:

- The `<rset>` (row set) element is optional. It is only present if the request was successful. Only a single `<row>` element is returned, with one `<rv>` (row value) element.
- 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 user identities for that subscriber, one user identity per `<id>` element. There can be one or many `<id>` elements per `<member>` element.
- The response includes pool members across all UDR instances in the pool network if the pso parameter was set to yes in the request and the PSO feature is enabled. Otherwise, only pool members on the local UDR instance are returned.
- The response includes only the address types specified in the *addresslist* parameter in the request (that is, those where *keyNameX* is in the list). Only the requested address types for a subscriber are returned in the response. If a subscriber is a pool member but does not have any of the address types, then the pool member is not included in the response.
- If the PSO feature is enabled, then each pool member returned also includes the *udrId* corresponding to the UDR instance where they reside in the pool network.

Table 74: Get All Pool Members Error Codes

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
REQUEST_TIMEOUT	Provisioning Request Timeout, as no response was received from Remote UDR

Examples

Request 1

A request is made to get the list of subscriber members for a pool. The PSO feature is not enabled. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
    <expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
  </oper>
</req>
```

Response 1

The request is successful, and the 3 subscriber members are returned. Only pool members from the local UDR instance are returned. The original request is not included.

```
<req name="operation">
  <res error="0" affected="1"></res>
  <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 subscriber members for a pool. The pool exists, but has no subscriber members. The request is not required in the response. The PSO feature is enabled.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
    <expr><param name="PoolID"/><op value="="/><value val="200000"/></expr>
```

```
  </oper>
</req>
```

Response 2

The request is successful, and no subscriber members are returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/></res>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <members>
          </members>]]>
      </rv>
    </row>
    </rset>
</req>
```

Request 3

A request is made to get the list of subscriber members for a pool. The pool does not exist (local or any remote UDR instances). The PSO feature is enabled and the pso parameter is set to yes. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="300000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>
```

Response 3

The request fails. The error value indicates that the pool does not exist on all queried UDR instances (local or remote), 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 get the list of subscriber members for a pool across all UDR instances in the pool network. The PSO feature is enabled and the pso parameter is set to yes. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>
```

Response 4

The request is successful, and the 3 member subscribers are returned. The response includes pool members across all UDR instances in the pool network. The original request is not included.

```
<req name="operation">
  <res error="0" affected="1"></res>
  <rset>
    <row>
      <rv>
```

```
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <members>
            <member>
              <udrId>1</udrId>
              <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>
              <udrId>2</udrId>
              <id><name>MSISDN</name><value>380561234777</value></id>
              <id><name>IMSI</name><value>311480100000999</value></id>
            </member>
            <member>
              <udrId>3</udrId>
              <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 5

A request is made to get the list of pool members across all UDR instances in the pool network. The *pso* parameter is set to *yes*. The request is not required in the response. The PSO feature is enabled. There is a connection issue between the local and remote UDR instances.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>
```

Response 5

The request fails. The *error* value indicates a provisioning request timeout, the request could not be sent to the remote UDR due to the connection being down, and the *affected* rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70053" affected="0"/>
</req>
```

Request 6

A request is made to get the list of subscribers for a pool on all UDRs. The PSO feature is enabled. The request is not required in the response. For this case, no response is received from the remote UDR.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>
```

Response 6

The request fails. The *error* value indicates a provisioning request timeout, no response was received from the remote UDR. The *affected* rows are 0 and the original request is not included.

```
<req name="operation" resonly="y">
  <res error="70053" affected="0"/>
</req>
```

Request 7

A request is made to get the list of subscriber members for a pool. The requested *addressList* type is *IMSI* and none of the subscriber members have *IMSIs*. The request is not required in the response. The PSO feature is enabled.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="200000"/></expr>
<expr><param name="AddressList"/><op value="="/><value val="IMSI"/></expr>
  </oper>
</req>
```

Response 7

The request is successful, and no subscriber members are returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/></res>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <members>
          </members>]]>
      </rv>
    </row>
    </rset>
</req>
```

Request 8

A request is made to get the list of subscriber members for a pool when the PSO feature is disabled and the *pso* parameter is set to yes. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>
```

Response 8

The request is successful, and the 3 member subscribers are returned. The response includes local pool members only. The original request is not included.

```
<req name="operation">
  <res error="0" affected="1"></res>
  <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>
```

7.8 Pool Special Operation Commands

A transparent data entity may contain data that is organized in rows. An example of a row is a specific PoolQuota in the PoolQuota entity.

The required row is identified in the request by the *rowIdName*.

A specific instance of a PoolQuota (a specified row) in the PoolQuota transparent data entity can have its fields reset to pre-defined values using a provisioning command.

Table 75: Summary of Pool Special Operation Commands

Command	Description	Keys	Command Syntax
Reset Pool Quota	Reset the fields in the specified PoolQuota	Pool ID and Row Identifier	`<req name="operation">` `<oper name="Reset" ent="PoolQuotaEntity">`

7.8.1 Reset PoolQuota

Description

This operation resets a particular quota row in the PoolQuota data associated with a pool.

If more than one row matches the requested *quotaName*, then the reset request fails.

If the pool has PoolQuota data, then the configured values in the specified PoolQuota row are reset to the configured default values.

NOTES

- The *rowIdName* is case-sensitive. If a row existed called DayPass, then an attempt to reset a quota row called DayPass is successful, but an attempt to reset a quota row called DAYPASS fails.
- When a PoolQuota instance is reset using the **`Reset`** command, each resettable field is set to its defined reset value. If the field does not exist, it is not created. But, if a resettable field does not exist, and the field has a default value, then the field is created with the default value.
- This operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The PoolQuota transparent data must exist for the pool.

The specified PoolQuota row must exist in the PoolQuota transparent data.

Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="Reset" ent="entityName">
<expr><param name="PoolID"/><op value="="/>
          <value val="poolID"/></expr>
    <expr><param name="rowIdName"/><op value="="/>
          <value val="rowIdValue"/></expr>
[
    <expr><param name="instanceFieldName"/><op value="="/>
          <value val="instanceFieldValue"/></expr>
]

  </oper>
</req>
```

- *resonly (optional):* Indicates whether the response should consist of the result only, without including the original request in the response

  Values:

  - y

    Only provide the result, do not include the original request

  - n

    Include the original request in the response (default)

- *id* (optional): Transaction ID value that is in the request and is passed back in the response.

  Values: 1 through 4294967295

- *entityName*: A user defined entity type/name for the transparent data.
  - Value is PoolQuotaEntity for the PoolQuota transparent data.
- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.

  Values: 1 through 999999999999999999999

- *rowIdName*: Name of the XML attribute that identifies the row in the XML data blob.

  Value is name for Quota transparent data.

- *rowIdValue*: The row name value that identifies the row in the XML data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.

  Value is `cid` or `Type` for the PoolQuota transparent data.

- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName.*

Response

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

- *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.

  Values: 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 Table 76 for other values.
- *affected*: The number of pools reset. A value of 1 indicates success.

Table 76 Reset PoolQuota Error Codes

Error Code	Description
KEY_NOT_FOUND	Key Not Found. A 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
MULTIPLE_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 PQ1 PoolQuota row for a pool. The pool has PoolQuota data, and the PoolQuota data contains a PoolQuota row called PQ1. The request is not required in the response.

```
<req name="operation">
 <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
          <value val="100000"/></expr>
    <expr><param name="name"/><op value="="/><value val="PQ1"/></expr>
  </oper>
</req>
```

Response 1

The request is successful, and the specified PoolQuota 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 PoolQuota row. The pool does not have PoolQuota data. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
          <value val="200000"/></expr>
    <expr><param name="name"/><op value="="/><value val="Monthly"/></expr>
  </oper>
</req>
```

Response 2

The request fails. The error value indicates the pool does not have PoolQuota 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 PQ6 PoolQuota row. The pool has PoolQuota data, but the PoolQuota data does not contain a PoolQuota row called PQ6. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
          <value val="300000"/></expr>
    <expr><param name="name"/><op value="="/><value val="PQ6"/></expr>
  </oper>
</req>
```

Response 3

The request fails, because the PQ6 data row was not present. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70032" affected="0"/>
</req>
```

Request 4

A request is made to reset the PQ6 PoolQuota row. The pool has PoolQuota data with multiple rows called PQ6. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
          <value val="400000"/></expr>
    <expr><param name="name"/><op value="="/><value val="PQ6"/></expr>
  </oper>
</req>
```

Response 4

The request fails, because multiple rows exist with the rows called PQ6. The original request is not included.

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

Request 5

A request is made to reset the PQ1 PoolQuota row for a pool with cid field 1234. The pool has PoolQuota data, and the PoolQuota data contains a PoolQuota row called PQ1. The request is not required in the response.

```
<req name="operation">
 <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
          <value val="400000"/></expr>
<expr><param name="name"/><op value="="/><value val="PQ1"/></expr>
<expr><param name="cid" /><op value="="/><value val="1234"/></expr>
  </oper>
</req>
```

Response 5

The request is successful, and the specified PoolQuota row was reset. The original request is not included.

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

Request 6

A request is made to reset the PQ1 PoolQuota row for a pool with cid field 1234. The pool has PoolQuota data, the PoolQuota data contains a PoolQuota row called PQ1 and cid field. The request is not required in the response.

```
<req name="operation">
 <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
          <value val="500000"/></expr>
<expr><param name="name"/><op value="="/><value val="PQ1"/></expr>
    <expr><param name="cid" /><op value="="/><value val="1234"/></expr>
  </oper>
</req>
```

Response 6

The request is successful, and the specified PoolQuota row was reset. The original request is not included.

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

APPENDIX A. ERROR CODES

SOAP error codes are returned by the SOAP interface in the `error` attribute parameter of the `<res>` message (see section 4.2.2). 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 inTable 77.

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 is 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 * (asterisk) are permanent errors that can be fixed by means of configuration, such as configuring the entities/fields in the SEC.

Table 77: SOAP Interface Error Codes

Error Code	Value	Type	Description
NOT_PROCESSED	1	P	Not processed. The request was in a block transaction, and was not processed due to an error with another request in 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 field is not a valid field in 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 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 a member of a pool.
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	This error code is utilized for two scenarios: 1. Register Already Exists 2. A property exists and the *odk* flag was not set to indicate the insert request be converted to an update.
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 exists.

Error Code	Value	Type	Description
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.
INVALID_KEY_VALUE	70037	P	The key value supplied is invalid, due to invalid characters/format.
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 retries the command again.
DURABILITY_DEGRADED	70039	T	Data could not be committed as Durability is degraded. **NOTE:** The client retries the command again.
DURABILITY_TIMEOUT	70040	T	Data could not be made durable in the configured Durability Timeout. **NOTE:** The client retries 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.
MULTIPLE_KEYS_NOT_MATCH	70043	P	Multiple keys supplied do not refer to the same subscriber.
ONE_KEY_REQUIRED	70044	P	At least one key is required for a subscriber.
SYSTEM_CONGESTION	70045	T	Request rejected due to system congestion.
CONNECTION_ERROR	70046	P	Request to be sent to different UDR instance could not be sent because no valid connection is available to send the request on.
MAX_MEMBERS_BASIC_POOL	70051	P	The maximum number of subscribers in a basic pool has been exceeded.

Error Code	Value	Type	Description
ENTERPRISE_TO_BASIC_POOL_ FAILED	70052	P	Enterprise to Basic Pool Conversion failed because the Pool has more members than the maximum threshold for a basic pool.
REQUEST_TIMEOUT	70053	P	Provisioning Request Timeout, as no response was received from Remote UDR.
INVALID_USER_CREDENTIALS	70054	P	Authentication of the Username/Password received in the SOAP message header has failed validation.
AE_KEY_EXISTS	70055	P	An AE subscriber exists with the given key. Only applicable when option *enableAEKeyAlreadyExistsErrCode* is enabled.

APPENDIX B. SOAP INTERFACE SYSTEM VARIABLES

The SOAP interface has a set of system variables that affect its operation as it runs. SOAP interface variables (Table 70) can be set via the UDR GUI and can be changed at runtime to effect dynamic server reconfiguration.

Table 78: SOAP Interface System variables

Parameter	Description
SOAP Interface Port	SOAP Interface TCP Listening Port. **NOTE:** Changes to the TCP listening port do not take effect until the `udrprov` 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 connection remains active without a request being sent, before the connection is dropped. DEFAULT = 1200; RANGE = 1-86400
Maximum SOAP Connections	Maximum number of simultaneous SOAP Interface client connections. **NOTE:** Changes to the Maximum SOAP Connections option do not take effect until the `udrprov` process is restarted. DEFAULT = 100; RANGE = 1-100
Allow SOAP Connections	Whether or not to allow incoming provisioning connections on SOAP Interface. DEFAULT = UNCHECKED
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 is 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 = R10.0+
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 *(asterisk) are existing system variables defined and used by other components of UDR.

APPENDIX C. LEGACY SPR COMPATIBILITY MODE

UDR can be configured to run in a compatibility mode with the legacy SPR.

When the Compatibility Mode system option (see Appendix B), which is configurable by the UDR GUI, is set to R10+ then UDR behaves as described in the main body of this document. When Compatibility Mode is set to R9.x, then the differences contained in this appendix apply.

Enabling this configuration option results in the possibility of the format for requests and responses being different to the default UDR behavior. This appendix lists the different requests and responses that enabling the option affects.

C.1 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 shown.

Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
[
  <rset>
    <row>
<     <rv>publicIdentity</rv> | <rv/> >
<     <rv>MSISDN1[,MSISDN2[,MSISDN3]]</rv> | <rv/> >
<     <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <rv/> >
<     <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv/> >
<     <rv>accountId</rv> | <rv/> >
    </row>
]
[
    <row>
<     <rv>publicIdentity</rv> | <rv/> >
<     <rv>MSISDN1[,MSISDN2[,MSISDN3]]</rv> | <rv/> >
<     <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <r/> >
<     <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv/> >
<     <rv>accountId</rv> | <rv/> >
    </row>
    :
    <row>
<     <rv>publicIdentity</rv> | <rv/> >
<     <rv>MSISDN1[,MSISDN2[,MSISDN3]]</rv> | <rv/> >
<     <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <rv/> >
<     <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv/> >
<     <rv>accountId</rv> | <rv/> >
    </row>
]
  </rset>
]
</req>
```

- *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.

  Values: 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 Table 77 for other values.
- *affected*: The number of pools returned. A value of 1 or more is expected for success
- *publicIdentity*: The internal public identity value for the subscriber. This field is not used, and the no value are present
- *MSISDNX*: Comma separated list of (up to 3) MSISDN values corresponding to subscriber in the pool. Values are not present if an MSISDN is not provisioned for the subscriber

  Values: 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. Values are not present if an IMSI is not provisioned for the subscriber

  Values: 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. Values are not present if an NAI is not provisioned for the subscriber

  Refer to Table 5 for supported NAI formats and length

- *accountId*: AccountId corresponding to subscriber in the pool. This value is not be present if an AccountId is not provisioned for the subscriber
- Values: 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.

C.2 Legacy SPR SOAP Request Format

The legacy SPR uses a different SOAP request format as shiwn in Figure 6. Requests can be sent using the same format as done for the legacy SPR.

NOTE: The `<soapenv:Header>` element in the request is optional and can be omitted. It can be provided, but is ignored. Authentication is no longer performed using the *UserName*/*Passwd* in UDR.

Figure 8: Legacy SPR SOAP Request Format

```
<?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:ns2="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>

  </soapenv:Body>

</soapenv:Envelope>
```

C.3 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 6: 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 problems may arise.

The format of the SOAP response returned by UDR is listed in Figure 7.

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.

Figure 9: 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="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>
```

Example of a successful SOAP response in legacy SPR mode:

```
<?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>
```

C.4 soapAttributeOrderInResponse

In ProvOptions table, if *soapAttributeOrderInResponse* flag is set as TRUE then SOAP Response has the *error* attribute followed by by *affected* attribute in `res` element, otherwise the order is reversed.

Example of a successful SOAP response when *soapAttributeOrderInResponse* flag is set as FALSE (Default case):

```
<?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/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
     <ns1:message error="0">
      <![CDATA[<?xml version="1.0" encoding="UTF-8"?><req name="insert" resonly="y">
               <res affected="1" error="0"/>
               </req>]]>
     </ns1:message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of a successful response when *soapAttributeOrderInResponse* flag is set as TRUE:

```
<?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/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
     <ns1:message error="0">
     <![CDATA[<?xml version="1.0" encoding="UTF-8"?><req name="insert" resonly="y">
              <res error="0" affected="1"/>
              </req>]]>
     </ns1:message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

C.5 validateProvResponse

In ProvOptions table, if *validateProvResponse* is set as TRUE then the response for the provisioning Read/Get requests is validated and changed as defined by the SEC configuration.

Example of a successful SOAP response when *validateProvResponse* flag is set as FALSE (default case):

BillingDay field is changed to *billingday* for the subscriber. Provisioning Read/Get request is not validated as defined by the SEC configuration while retrieving subscriber.

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/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
    <ns1:message error="0">
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
    <req name="select" resonly="y">
    <res affected="1" error="0"/>
       <rset>
              <row>
                <rv>
                   <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
                   <subscriber>
                     <field name="MSISDN">9876543210</field>
                     <field name="billingday">23</field>
                     <field name="Entitlement">DayPass</field>
                     <field name="Custom20">xyz</field>
                     <field name="Entitlement">DayPassPlus</field>
                     </subscriber>]]]]>
                    <![CDATA[>
                   </rv>
                </row>
              </rset>
        </req>]]>
        </ns1:message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of a successful SOAP response when *validateProvResponse* flag is set as TRUE.

BillingDay field is changed to *billingday* for subscriber. Provisioning Read/Get request is validated as defined in the SEC configuration while retrieving subscriber. See the 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/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
    <ns1:message error="0">
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
    <req name="select" resonly="y">
    <res affected="1" error="0"/>
       <rset>
              <row>
                <rv>
                   <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
                   <subscriber>
                     <field name="MSISDN">9876543210</field>
                     <field name="BillingDay">23</field>
                     <field name="Entitlement">DayPass</field>
                     <field name="Custom20">xyz</field>
```

```
                    <field name="Entitlement">DayPassPlus</field>
                    </subscriber>]]]]>
                   <![CDATA[>
                  </rv>
                </row>
              </rset>
        </req>]]>
        </ns1:message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

C.6 Enable AE Key Already Exists Error

In ProvOptions table, if *enableAEKeyAlreadyExistsErrCode* is set as TRUE (default is set as FALSE) then error AE_KEY_EXISTS (70055) is returned when these criterias are met:

- A create subscriber request encounters key exists failure
- The key which exists belongs to an AE subscriber

Example of a SOAP response when *enableAEKeyAlreadyExistsErrCode* flag set to TRUE and create subscriber request fails when the key exists:

```
<?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/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
     <ns1:message error="0">
     <![CDATA[<?xml version="1.0" encoding="UTF-8"?><req name="insert" resonly="y">
              <res error="70055" affected="0"/>
              </req>]]>
     </ns1:message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

APPENDIX D. MY ORACLE SUPPORT

My Oracle Support (*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 My Oracle Support 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 these selections in this sequence 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 these options:
    - For Technical issues such as creating a New Service Request (SR), select **1**
    - For Non-technical issues such as registration or assistance with My Oracle Support, select **2**

You are connected to a live agent who can assist you with My Oracle Support registration and opening a support ticket.

My Oracle Support is available 24 hours a day, 7 days a week, 365 days a year.

APPENDIX E. LOCATE PRODUCT DOCUMENTATION ON ORACLE HELP CENTER

Oracle Communications customer documentation is available on the web at the Oracle Help Center 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 *http://www.adobe.com*.

1. Access the Oracle Help Center site at *http://docs.oracle.com*
2. Click **Industries**.
3. Under the Oracle Communications subheading, click the **Oracle Communications documentation** link.
4. The Communications Documentation page opens. Most products covered by these documentation sets appear under the headings Network Session Delivery and Control Infrastructure or Platforms.
5. Click your product and then the release number.
6. A list of the documentation set for the selected product and release displays.
7. To download a file to your location, right-click the **PDF** link, select **Save target as** (or similar command based on your browser), and save to a local folder.