

Oracle® Communications
User Data Repository
Bulk Import/Export File Specification
Release 12.1
E68359-01

March 2016

ORACLE®

Oracle Communications User Data Repository Bulk Import/Export File Specification, Release 12.1

E68359-01

Copyright ©2014, 2016, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.



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

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

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

Call the CAS main number at 1-800-223-1711 (toll-free in the US), or call the Oracle Support hotline for your local country from the list at <http://www.oracle.com/us/support/contact/index.html>.

See more information on MOS in the Appendix section.

Table of Contents

1	INTRODUCTION	6
1.1	Purpose and Scope	6
1.2	References.....	6
1.3	Glossary.....	6
2	SYSTEM ARCHITECTURE	8
2.1	Overview	8
2.2	Database Transactions	10
2.2.1	Block Transaction Mode	10
2.2.1.1	Request Format	10
2.2.1.2	Examples	11
2.2.1.3	Import File Processing Sequencing	12
2.2.2	ACID-Compliance	13
2.2.2.1	Atomicity.....	13
2.2.2.2	Consistency.....	13
2.2.2.3	Isolation.....	13
2.2.2.4	Durability	13
2.3	Behavior During Low Free System Memory	14
2.4	Multiple Subscriber Key Processing	14
2.5	PNR Generation with Import	15
3	BULK OPERATIONS	16
3.1	Message Conventions	16
3.2	Import.....	16
3.2.1	Configuring Import Options	16
3.2.2	Import Files	16
3.2.3	Import File Format.....	17
3.2.3.1	Basic Import File Request Format	18
3.2.3.2	Case Sensitivity.....	20
3.2.4	Import File Comments.....	21
3.2.5	Import Log Files	21
3.2.6	Import Status.....	23
3.2.6.1	Import Status Table.....	24
3.3	Export.....	24
3.3.1	XMLExport	25
3.3.2	Export File and Format	26
3.3.2.1	Basic Export File Format.....	26
3.3.2.2	Subscriber Record	26
3.3.2.3	Pool Record	27
3.3.2.4	Examples	27
3.3.3	Export Conversion Tool (xmlconverter)	30
3.3.4	Configuring Export Options.....	30
3.3.5	Scheduling Exports	30
3.3.5.1	Display	31
3.3.5.2	Insert	31
3.3.5.3	Edit.....	31
3.3.5.4	Delete.....	32
3.3.5.5	Export Status.....	32
4	UDR DATA MODEL	34
4.1	Subscriber Data	36
4.1.1	Subscriber Profile.....	36
4.1.2	Quota	38
4.1.3	State.....	39
4.1.4	Dynamic Quota	39

4.2	Pool Data	40
4.2.1	Pool Profile	40
4.2.2	Pool Quota	41
4.2.3	Pool State	42
4.2.4	Pool Dynamic Quota	42
4.3	Date/Timestamp Format	42
5	SUBSCRIBER PROVISIONING	44
5.1	Subscriber Profile Commands	44
5.1.1	Create Subscriber	45
5.1.2	Update Subscriber	48
5.1.3	Delete Subscriber	52
6	POOL PROVISIONING.....	54
6.1	Pool Profile Commands	54
6.1.1	Create Pool	54
6.1.2	Delete Pool.....	56
6.2	Additional Pool Commands.....	57
6.2.1	Add Member to Pool	58
6.2.2	Remove Member from Pool	61
7	GENERAL PROVISIONING	64
7.1	General Editing Commands.....	64
7.1.1	Create Data.....	64
7.1.2	Update Field.....	69
7.1.3	Update FieldSet	78
7.1.4	Delete Field	86
7.1.5	Delete FieldSet	94
8	SPECIAL OPERATIONS.....	100
8.1.1	Reset.....	100
9	RESTORE COMMANDS	104
9.1.1	Restore Subscriber	104
9.1.2	Restore Pool	107
APPENDIX A.	ERROR CODES	114
APPENDIX B.	BULK IMPORT/EXPORT VARIABLES	117
APPENDIX C.	MY ORACLE SUPPORT (MOS)	118
APPENDIX D.	CUSTOMER TRAINING	119
APPENDIX E.	LOCATE PRODUCT DOCUMENTATION ON THE ORACLE TECHNOLOGY NETWORK SITE	120

List of Figures

Figure 1: User Data Repository High Level Architecture.....	9
Figure 3: Import Log File – Import Successfully Completed Example	22
Figure 4: Import Log File – Import Interrupted Example.....	23
Figure 5: Import Status	24
Figure 5: Generating Output File	25
Figure 6: Export Schedule (Display).....	31
Figure 7: Export Schedule (Insert).....	31
Figure 8: Export Schedule (Edit)	32
Figure 9: Export Schedule (Delete)	32
Figure 10: Export Status.....	33
Figure 11: Data Model	36

List of Tables

Table 1: Glossary.....	6
Table 2: Message Conventions	16
Table 3: Import Request Operations	17
Table 4: Import File Comment Format.....	21
Table 5: Import Log File Format	22
Table 6: Import Log File Parameters	22
Table 7: Import Status Table	24
Table 8: Subscriber Profile Entity Definition	37
Table 9: Quota Entity Definition	38
Table 10: State Entity Definition	39
Table 11: Dynamic Quota Entity Definition.....	40
Table 12: Pool Profile Entity Definition	41
Table 13: Summary of Subscriber Profile Commands	44
Table 14: Summary of Pool Profile Commands	54
Table 15: Summary of Additional Pool Commands.....	57
Table 16: Summary of General Editing Commands.....	64
Table 17: Summary of Special Operation Commands	100
Table 18: Summary of Restore Commands	104
Table 19: Error Codes	114
Table 20: Bulk Import/Export variables.....	117

1 Introduction

1.1 Purpose and Scope

This document presents the bulk import/export file interface to be used by provisioning client applications to administer the Provisioning Database of the Oracle Communications User Data Repository (UDR) system. Through bulk import/export files, an external provisioning system supplied and maintained by the network operator may 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 have a need to use the bulk import/export file interface.

1.2 References

The following external document references capture the source material 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
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)
OAMP	Operations, Administration, Maintenance, and Provisioning
NOAM&P	Network OAM&Provisioning
PCRF	Policy Charging & Rules Function
PS	Provisioning System
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
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 an SPR, which is a database system that acts as a single logical repository that stores subscriber data. The subscriber data that traditionally has been stored into the HSS /HLR/AuC, Application Servers, etc., is now stored in UDR as specified in 3GPP UDC information model [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 know to perform the service. User identities (e.g. MSISDN, IMSI, NAI and AccountId), service data (e.g. service profile) and authentication data are examples of the subscription data. This kind of user data has a lifetime as long as the user is permitted to use the service and may be modified by administration means. Temporary subscriber data is dynamic data which may be changed as a result of normal operation of the system or traffic conditions (e.g. transparent data stored by Application Servers for service execution, user status, usage, etc.).

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

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

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

As shown in the figure, Oracle Communications 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 will reside on the N-OAMP servers. The initial release will focus on the development of the Sh messaging interface for use with the UDR application.

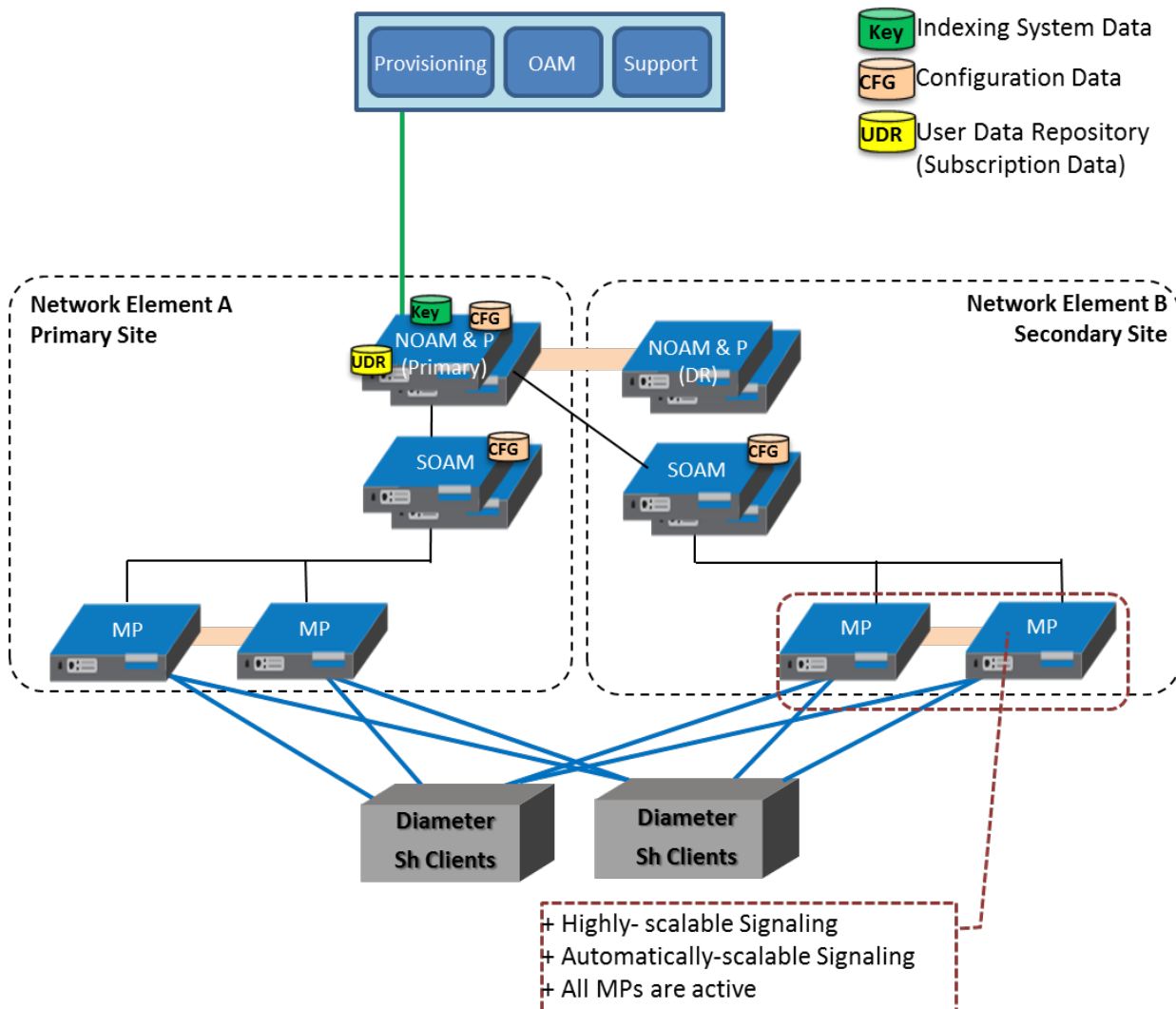
As the product evolves forward, the subscriber profiles in UDR can be expanded to support data associated with additional applications. Along with that, the MPs can be expanded to support additional Diameter interfaces associated with these applications. The IPFE can be integrated with the product to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

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

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

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

Figure 1: User Data Repository High Level Architecture



The bulk import provisioning interface provides following data manipulation commands:

Subscriber:

- Subscriber Profile create/modify/delete
- Subscriber Profile field create/modify/delete
- Subscriber opaque data create/modify/delete
 - Quota, State and Dynamic Quota
- Subscriber data row and/or field create/modify/delete
 - Quota, State and Dynamic Quota
- Reset of Subscriber data row data
 - Quota
- Restore of complete Subscriber data

Pool:

- Pool Profile create/delete
- Pool Profile field create/modify/delete
- Pool opaque data create/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
- Reset of Pool data row data
 - Pool Quota
- Pool subscriber membership operations
 - Add/remove from pool
- Restore of complete Pool and member Subscriber data

The bulk export interface exports the following information:

Subscriber:

- Subscriber Profile
- Subscriber opaque data (if present)
 - Quota, State and Dynamic Quota
- If a subscriber is a member of a pool :
 - Subscriber pool membership (i.e. PoolID)

Pool:

- Pool Profile
- Pool opaque data (if present)
 - Pool Quota, Pool State and Pool Dynamic Quota

2.2 Database Transactions

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

2.2.1 Block Transaction Mode

The block database transaction mode requires explicit `<transaction>` XML tags around all of the requests within a transaction. Within a `<transaction>` request, all requests must be contained *on a single line* in the import file.

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

If a block transaction fails, the error code for the request within the block that encountered an error will be returned.

All block transactions must also satisfy limits indicated by the `Maximum Provisioning Backend Response Timeout` and `Transaction Durability Timeout` system variables, which are defined in Appendix B. If any of those limits are exceeded, the transaction will be aborted and automatically rolled back.

Note: The relevant transaction related measurement(s) will be incremented once per `<transaction>` request (i.e. by +1), i.e. `ProvTxnCommitted`, `TxProvTxnFailed` or `TxProvTxnAborted`.

Note: The `<restoreSubscriber>` and `<restorePool>` requests cannot be used within a `<transaction>`.

2.2.1.1 Request Format

```
<transaction>
  <txRequest id="id1">
    request1
  </txRequest>
[
  <txRequest id="id2">
    request2
  </txRequest>
```

```

:
<txRequest id="idN">
  requestN
</txRequest>
]
</transaction>

```

- **idX**: Identifier for the request within the transaction
 - Values: 0–4294967295
- **requestX**: Bulk import request contained in the transaction
 - **Note**: The maximum number of requests that can be included in a <transaction> transaction is 50
 - **Note**: The maximum number of <addPoolMember> requests that can be included in a <transaction> is 12, if each <addPoolMember> request adds 25 members.
 - **Note**: The maximum number of <deletePoolMember> requests that can be included in a <transaction> is 12, if each <deletePoolMember> request deletes 25 members.
 - **Note**: On a Low Capacity Server Configuration, the maximum number of <addPoolMember> requests that can be included in a <transaction> is 3, if each <addPoolMember> request adds 25 members.
 - **Note**: On a Low Capacity Server Configuration, the maximum number of <deletePoolMember> requests that can be included in a <transaction> is 3, if each <deletePoolMember> request deletes 25 members.
 - **Note**: The maximum number of <addPoolMember> requests that can be included in a <transaction> is 50, if each <addPoolMember> request adds 1 member.
 - **Note**: The Database used by UDR allows only 1000 operations to be performed within a single transaction.

2.2.1.2 Examples

Request

This request creates 2 subscribers, and updates another subscriber.

```

<transaction>
  <txRequest id="1">
    <createSubscriber>
      <key>
        <MSISDN>15141234567</MSISDN>
        <IMSI>302370123456789</IMSI>
      </key>
      <entity>
        <data>
          <name>Subscriber</name>
          <interface>XMLIMPORT</interface>
        </data>
        <content>
<![CDATA[
<subscriber>
  <field name="MSISDN">15141234567</field>
  <field name="IMSI">302370123456789</field>
  <field name="BillingDay">5</field>
  <field name="Tier">Gold</field>
  <field name="Entitlement">DayPass</field>
</subscriber>
]]>
          </content>
        </entity>
      </createSubscriber>
    </txRequest>
  <txRequest id="2">

```

```

    <createSubscriber>
      <key>
        <MSISDN>14505551234</MSISDN>
        <IMSI>302370999999999</IMSI>
      </key>
      <entity>
        <data>
          <name>Subscriber</name>
          <interface>XMLIMPORT</interface>
        </data>
        <content>
<![CDATA[
<subscriber>
  <field name="MSISDN">14505551234</field>
  <field name="IMSI">302370999999999</field>
  <field name="BillingDay">1</field>
  <field name="Tier">Silver</field>
  <field name="Entitlement">DayPass</field>
</subscriber>
]]>
          </content>gg
        </entity>
      </createSubscriber>
    </txRequest>
    <txRequest id="3">
      <updateField clearAll="true">
        <key>
          <MSISDN>14165555555</MSISDN>
        </key>
        <entity>
          <data>
            <name>Subscriber</name>
            <interface>XMLIMPORT</interface>
            <xpath>/subscriber</xpath>
          </data>
          <fields>
            <field name="BillingDay">23</field>
            <field name="Tier">Gold</field>
          </fields>
        </entity>
      </updateField>
    </txRequest>
  </transaction>

```

2.2.1.3 Import File Processing Sequencing

In order to improve the performance of bulk import operations, requests read from an import file are *not* guaranteed to be processed in the order they are specified in the import file.

- If multiple requests for a subscriber or pool must be performed in order, it is necessary to use a block transaction (see section 2.2.1), and sequence the requests within the transaction in the order in which they must be performed
- Within a transaction, the request processing is guaranteed to be the order in which the requests are specified
- For example :
 - If the import file contained requests to do the following :
 1. Create subscriber #1
 2. Update subscriber #1
 3. Update subscriber #1
 - Then it could happen that the first command executed was (3) to update the subscriber, which would fail because the subscriber did not exist. Then (2) could occur next, and the update would fail because the

subscriber still did not exist, and then finally (1) would create the subscriber, but at the end of the import operation the subscriber data would NOT contain the updates made in (2) and (3).

- By creating a transaction such as :
 - `<transaction>`
 - `Create Subscriber #1`
 - `Update Subscriber #1`
 - `Update Subscriber #1`
 - `</transaction>`
- Then the request would work as expected

2.2.2 ACID-Compliance

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

2.2.2.1 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 within that transaction which were executed successfully. If any database errors are encountered while committing the transaction, all updates are rolled back and the database is restored to its previous state.

2.2.2.2 Consistency

Data across all requests performed inside a transaction is consistent.

2.2.2.3 Isolation

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

2.2.2.4 Durability

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

Note: For bulk import, requests are considered if the database transaction is committed successfully. The bulk import tool does NOT wait until the transaction has become durable before moving onto the next request. But, requests should eventually become durable a short time after.

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

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

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

2.3 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 `MemThresholdReached`. Before this happens, memory threshold alarms will be raised indicating the impending behavior if the critical level is reached.

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

```
[error 61 errorText : line lineNumber]
```

2.4 Multiple Subscriber Key Processing

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

When multiple keys are supplied in a request (such as an IMSI *and* an MSISDN), UDR will look up all supplied keys, and only consider the subscriber record “found” if all supplied keys correspond to the same subscriber.

If any key value does not exist, then `KeyNotFound` will be returned. If multiple keys are provided, and all keys exist, but do not correspond to the same subscriber, then the error `MultipleKeysNotMatch` will be returned.

Example request :

```
<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
    <IMSI>302370123456789</IMSI>
    <AccountId>10404723525</AccountId>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="Tier"/>
      <field name="Custom1"/>
    </fields>
  </entity>
</deleteField>
```

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

```
<deleteField>
  <key>
    <IMSI>302370123456789</IMSI>
    <IMSI>184569547984229</IMSI>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
  :
```

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. It is not allowed to mix `PoolID` and subscriber key values in the same request, and doing so will result in an `InvalidInputXML` error response. For example, the following is NOT allowed:

```
<deleteField>
  <key>
    <IMSI>302370123456789</IMSI>
```

```
<PoolID>100000</PoolID>  
</key>  
<entity>  
:
```

2.5 PNR Generation with Import

UDR has a configuration option `PNR Generation with Import` (refer to Appendix B for Provisioning Options) that, when enabled, results in Sh PNR messages being generated for subscribers that have active subscriptions (i.e. an SNR for the subscriber has been received) if import requests:

- Update, create, or delete data associated with that subscriber
- Update, create, or delete data associated with the pool that the subscriber is a member of
- Add the subscriber, or remove the subscriber from a pool

3 Bulk Operations

3.1 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
Bold	Text that appears literally in the message (e.g. keywords, commas, parentheses, and error text are bold)
<i>italics</i>	Parameter values that will be replaced by an actual parameter name or numeric value.
spaces	Spaces (i.e. zero or more space characters, “ ”) may be inserted anywhere except within a single name or number. At least one space is required to separate adjacent names or numbers.
...	Variable number of repeated entries. For example: dn DN1, dn DN2, ...,dn DN7, dn DN8
< >	Angle brackets are used to enclose parameter values that are choices or names. For example, in parameter1 <1/2/3> , the numbers represent specific value choices. In parameter2 <ServerName> , the <i>ServerName</i> 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.

3.2 Import

UDR supports mechanisms to support file-based bulk loading of Subscriber data. Subscriber data records can be entered by reading files which contain the set of transactions to be processed. The import process is non-blocking, it runs together with Provisioning updates as well as network (Sh) updates.

3.2.1 Configuring Import Options

The user can configure Import options by using the UDR GUI's Main Menu: *UDR* → *Configuration* → *Provisioning Options* screen. Refer to Appendix B for Provisioning Options.

3.2.2 Import Files

Files from a remote directory can be imported and the values within the files are used to populate the database. The import file is an ASCII text file that contains a series of database manipulation requests. Each request must be formatted on a single line. Import files are placed in the **Remote Import Directory** on the remote server specified in **Remote Host IP Address** field on the Provisioning Options (see Appendix B). They are detected within five minutes and automatically resynched over SSH File Transfer Protocol (SFTP) to the file management storage area on the active server.

This local directory is always `/var/TKLc/db/filemgmt/provimport` and is displayed in Provisioning Options (see Appendix B). For a file to be imported it must:

- be properly named
 - XML import files must have `*.ixml` file extensions.
- have been placed in the remote directory after the time when the import last ran
- must not have been previously imported
 - A file that has already been imported into the local directory is not imported again, even if its status is failed.
 - To import a previously failed file, correct the file as necessary, rename the file, and then place the renamed file in the remote directory.
 - **Note:** The renamed file must have a *file* update timestamp later than the last imported file, or UDR will not process it

Once fully downloaded, each file is automatically imported into the Provisioning Database sequentially in the order in which their download completed. The order of imported files is also indicated on the UDR GUI's *Main Menu: UDR* → *Maintenance* → *Import Status* screen. The import file is validated one command at a time and the import will continue if a command fails. Failed commands will appear in the import log file (see section 3.2.5).

An Import file may contain as many requests as the storage media used to hold the import file allows.

3.2.3 Import File Format

Import files contain data in XML format. An XML import file is an ASCII text file that contains a series of database manipulation requests in native XML format as specified in section 5. The data used in the imports is defined in section 4.

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

Table 3: Import Request Operations

Operation	Description	Section
<code><createSubscriber></code>	Create a subscriber	5.1.1
<code><updateSubscriber></code>	Update a subscriber	5.1.2
<code><deleteSubscriber></code>	Delete a subscriber	5.1.3
<code><create></code>	Create a FieldSet	7.1.1
<code><updateField></code>	Update a Field	7.1.2
<code><updateFieldSet></code>	Update a FieldSet	7.1.3
<code><deleteField></code>	Delete a Field	7.1.4
<code><deleteFieldSet></code>	Delete a FieldSet	7.1.5
<code><createPool></code>	Create a Pool	6.1.1
<code><deletePool></code>	Delete a Pool	6.1.2
<code><addPoolMember></code>	Add a pool member	6.2.1
<code><deletePoolMember></code>	Delete a pool member	6.2.2
<code><reset></code>	Reset an Element	8.1.1
<code><transaction></code>	Transaction container	2.2.1
<code><restoreSubscriber></code>	Restore a subscriber	9.1.1
<code><restorePool></code>	Restore a pool	9.1.2

3.2.3.1 Basic Import File Request Format

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

```

<requestName [create="create"]
              [createEntityIfNotExist="createEntityIfNotExist"]
              [clearAll="clearAll"]
              [inTx="inTx"]>

  <key>
    <keyName>keyValue</keyName>
  </key>

  <entity>

    <data>
      <name>dataName</name>
      <interface>dataInterface</interface>
      <xpath>dataXpath</xpath>
      <version>
        <name>versionName</name>
        <value>versionValue</value>
      </version>
    </data>

    <fields>
      <field name="fieldName" [delete="deleteField"]>fieldValue</field>
    </fields>

    <content>
      entityContent
    </content>

  </entity>

  <members>
    <member>
      <keyName>keyValue</keyName>
    </member>
  </members>

</requestName>

```

Note: Each request is formatted on a *single line only*. It is shown above expanded for readability.

The *requestName* element indicates the request type, such as **<createSubscriber>**, **<updateField>** etc. See Table 3 for the full list of request name values.

The **create** attribute is used to indicate whether the row within an entity should be created if it does not already exist. Possible values of *create* are **true** or **false**. Note: this attribute is only applicable to the **<updateFieldSet>** request.

The **createEntityIfNotExist** attribute is used to indicate that the entire entity being updated should be created if it does not already exist before applying the update to create the entity/row. Possible values of *createEntityIfNotExist* are **true** or **false**. Note: this attribute is only applicable to the **<create>** and **<updateFieldSet>** requests.

The **clearAll** attribute is used to indicate that when a field is being updated, if all existing values within the field should first be cleared before applying the update. Possible values of *clearAll* are **true** or **false**. Note: this attribute is only applicable to the **<updateField>** request. Note: For fields that are not multi-value (i.e. single value), the value of

clearAll must be set to **true** else the request will attempt to add a second instance of the field, and the request will fail.

The **inTx** attribute is used to indicate that the request is in a transaction. Possible values of *inTx* are **true** or **false**. Note: this optional attribute is present on every operation, but currently has no use for requests *in import files*, and this attribute should always be omitted. Note: because this attribute has no effect in any requests, it is not listed in the subsequent sections describing each request.

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

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

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

Note: Multiple subscriber key values may be supplied. If UDR is configured to allow this, when attempting to find the subscriber, all key values supplied must be valid for the single subscriber in order to match the subscriber. For more details see section 2.4.

The *dataName* element identifies the provisioning entity type on which the request is being performed on. Values are either **Subscriber**, **Quota**, **State**, **DynamicQuota**, **Pool**, **PoolQuota**, **PoolState**, or **PoolDynamicQuota** depending on the request, which should match the configured Entity values in the SEC for the XML import interface.

The *dataInterface* element must be set to **XMLIMPORT** for bulk import requests.

When a request is performing an action on a specific field or row in an entity (such as updating a field value in a specific quota row), the XML XPath expression which references the row to be created/updated must be specified in *dataXPath*. The *dataXPath* value can indicate the base element, or row name optionally including a particular instance (i.e. the **<cid>** field in a Quota row, or the **<InstanceId>** field in a DynamicQuota row) when the row name is the same etc.

The **<version>** element is only used when creating a new entity when creating a new row instance (using the **<updateFieldSet>** request). The **<name>** and **<version>** elements within are used to indicate (if required) which version of the transparent entity (if more than one are defined) to create the entity with by default. The *versionName* and *versionValue* values indicate entry defined in the SEC to use.

When a field value is included to be set (for example in an insert/update request), a **<fields>** element is present. Within this, zero, one, or many **<field name="fieldName">fieldValue</field>** element(s) are present. The *fieldName* indicates the name of the field being set, and the *fieldValue* is the value to set it to. A **<field>** element may contain the optional **deleteField** attribute, that when set to **true** indicates that during an **<updateField>** request that the specific field is to be deleted not updated. It is also possible to specify a specific field value to indicate that the field should only be deleted if the current value matched.

Note: When specifying fields in a **<fields>** 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 is a list type (such as Entitlement in Profile), multiple instances of the field element should be specified.

When *entityContent* is to be set as an XML data “blob”, the blob data should be included within the constructs of an XML CDATA section. The CDATA section starts with “<![CDATA[”, then the *entityContent* containing the XML data “blob”, and the CDATA section ends with “]]>”.

The **<members>** element is used to contain the list of subscribers when adding or removing a subscriber to/from a pool using the **<addPoolMember>** and **<deletePoolMember>** requests. A single key (identified by *keyName* and *keyValue*) for each individual subscriber is contained within individual **<member>** elements.

3.2.3.2 Case Sensitivity

The constructs that bulk import requests are made up of (such as **<updateField>**, **<key>**, **<entity>**, **<xpath>** etc.) *are* case-sensitive. Exact case must be followed for all the commands described in this document, or the request will fail.

For example, the following is valid:

```
<addPoolMember>
  <key>
    <PoolID>100000</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>15141234567</MSISDN>
    </member>
  </members>
</addPoolMember>
```

But the following is NOT:

```
<addPoolMember>
  <KEY>
    <PoolID>100000</PoolID>
  </KEY>
  <members>
    <member>
      <MSISDN>15141234567</MSISDN>
    </member>
  </members>
</addPoolMember>
```

Request names defined in *requestName* *are* case-sensitive, for example “createSubscriber”, “updateField”, and “addPoolMember”.

Entity names defined in *dataName* are *not* case-sensitive.

Entity field or key names/attributes in *fieldName*, *keyName* or *versionName* *are not* case-sensitive.

Entity field/key values *are* case-sensitive, for example *fieldValue*, *keyValue*, and *versionValue*.

Examples:

- When accessing a *fieldName* defined as “inputVolume” in the SEC, then “inputvolume”, “INPUTVOLUME” or “inputVolume” *are* valid field names. Field names do not have to be specified in a request as they are defined in the SEC

- When a *fieldValue* is used to find a field (such as when using the `<deleteField>` command), the field value *is* case-sensitive. If a multi-value field contained the values “DayPass,Weekend,Evening” and the `<deleteField>` command was used to delete the value “WEEKEND”, then this would fail
- When an XPath value is specified in *dataXPath*, such as when accessing a row in an entity (for example in Quota), then everything contained in the *dataXPath* *is* case-sensitive, and must be specified as defined
 - For example, for Quota the following is valid :
 - `/usage/quota[@name='Q1']`
 - But the following is NOT valid :
 - `/usage/quota[@NAME='Q1']`
- When a *keyName* is specified in a `<key>` element (such as “MSISDN”), the name is *not* case-sensitive
- When a *keyValue* is specified in the `<key>` element (such as for an NAI), the value *is* case-sensitive. For example, for a subscriber with an NAI of “mum@foo.com”, then “Mum@foo.com” or “MUM@FOO.COM” will *not* find the subscriber
- When a *versionName* is specified in a `<version>` element (such as “version”), the name is *not* case-sensitive
- When a *versionValue* is specified in the `<version>` element (such as “v1”), the value *is* case-sensitive. For example, for a transparent entity with a single version defined of “v1”, then “V1” will *not* match the version defined

3.2.4 Import File Comments

Import files in XML format can contain blank lines and comment lines. UDR ignores these particular lines. Comment lines in XML files have the following format:

Table 4: Import File Comment Format

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

If the comment is contained with the XML blob for an *opaque* entity, within the CDATA constraint, then the comment *will* get stored in the XML blob.

If the comment is contained with the XML blob for a *transparent* entity, within the CDATA constraint, then the comment will *not* get stored in the XML blob.

3.2.5 Import Log Files

An import log file is created for each file that is imported and a copy is automatically uploaded to the same location the import file was downloaded from on the remote server. The log file has the same name as its corresponding import file with “.log” appended. Import log files on the local system are viewable for up to 7 days or until manually removed via either the UDR GUI’s *Main Menu: Status & Manage* → *Files* screen, or the UDR GUI’s *Main Menu: UDR* → *Maintenance* → *Import Status* screen.

The import log file contains:

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

Table 5: Import Log File Format

```

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

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

mm/dd/yy hh:mm:ss <Completed|Interrupted> (linesImported of linesToImport) percentCplt%
complete
Successful: successfulCmds Failures: failedCmds Total: totalCmds

```

Table 6: Import Log File Parameters

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

Below are examples of import log files for successfully completed and interrupted import files:

Figure 2: Import Log File – Import Successfully Completed Example

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

```

<removeSubscriber><imsi>310910421000102</imsi></removeSubscriber>
[error 6 Invalid XML: 100 Line:1, Column:19 error: no declaration found for element 'removeSubscriber' : line 1]

<updateSubscriber><key><MSISDN>33123654862</MSISDN></key><subscriber><AccountId>10404723525</AccountId><MSISDN>33123654862</MSISDN><IMSI>184569547984229</IMSI></subscriber><entity><data>
<name>Subscriber</name><interface>XMLIMPORT</interface></data><content><![CDATA[<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">DayPass</field></subscriber>]]></content></entity></updateSubscriber
>
[error 39 Key not found: [MSISDN:33123654862] : line 1]

<deleteSubscriber><key><MSISDN>33123654862</MSISDN></key></deleteSubscriber>
[error 39 Key not found: [MSISDN:33123654862] : line 1]

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

Successful: successfulCmds Failures: failedCmds Total: totalCmds

```

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

Figure 3: Import Log File – Import Interrupted Example

```

02/06/13 13:28:01 Started (0 of 200) 0% complete
02/06/13 13:28:03 Connection terminated
02/06/13 13:28:03 Interrupted (100 of 200) 50% complete
Successful: 100 Failures: 0 Total: 100

```

3.2.6 Import Status

The Import Status GUI is used to view and monitor the status of import operations.

The user can view the status of all imported files by using the UDR GUI's *Main Menu: UDR → Maintenance → Import Status* screen. This screen displays the import file and result file names, the current progress (percentage) and status of the import, number of import commands that succeeded and failed, and time stamps for when the import was queued, started and completed. The Import Status screen also provides hyperlinks so that the user can view the import and result files as text or save them locally.

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

Figure 4: Import Status

Import File	Time Queued	Time Started	Time Completed	Progress	Result Log	Pass Count	Fail Count	Status
import_5_delete_imsi.c...	2013-04-18 10:43:08	2013-04-18 10:43:09	2013-04-18 10:43:15	100%	import_5_del_ete_imsi.c...	999	0	Completed
import_6_delete_imsi.c...	2013-04-18 10:45:04	2013-04-18 10:45:05	2013-04-18 10:45:10	100%	import_6_del_ete_imsi.c...	978	21	Completed
import_7_delete_imsi.c...	2013-04-18 10:48:09	2013-04-18 10:48:10	2013-04-18 10:48:15	100%	import_7_del_ete_imsi.c...	0	999	Completed
import_8_delete_imsi.c...	2013-04-18 10:55:00	2013-04-18 10:55:01	2013-04-18 10:55:06	100%	import_8_del_ete_imsi.c...	950	49	Completed

Delete

3.2.6.1 Import Status Table

Import Status table will contain an entry for each XML file imported from Remote Server. The Status will be changed on the basis of events occurred.

Table 7: Import Status Table

Current State	Event	Action	Next State
-	XML file (*.ixml) found on Remote server	Start downloading the XML file	Transferring
		Add an entry to ProvImports table for that XML file	
Transferring	File successfully downloaded to NOAMP server	Import the file into the Provisioning Database	Transfer Completed
	File downloading failed in between (any reason)	-	Transfer Failed
Transfer Completed	-	Parse the XML file	In Progress
		Send the Internal XML Commands to UDRBE	
In Progress	Responses received from UDRBE	Update the Failed responses to a log file <XML file>.log	Completed
		Update the ProvImports Table with the execution status	
		Send the Result log file back to the remote server at the same location	

3.3 Export

UDR export will generate XML output to align with the output produced by the Oracle Communications Subscriber Data Management v9.3. The export feature allows a text export of the database based on a range (I.E MSISDN/IMSI range). UDR operators can schedule repeat exports. Exported data may also be offloaded to a remote server. The exported text file is also available to be downloaded from the 'file transfer area'. Customers may use exported records to do data manipulation of subscriber data or as an import file. The export process is non-blocking, it runs together with Provisioning updates as well as network (Sh) updates.

3.3.1 XMLExport

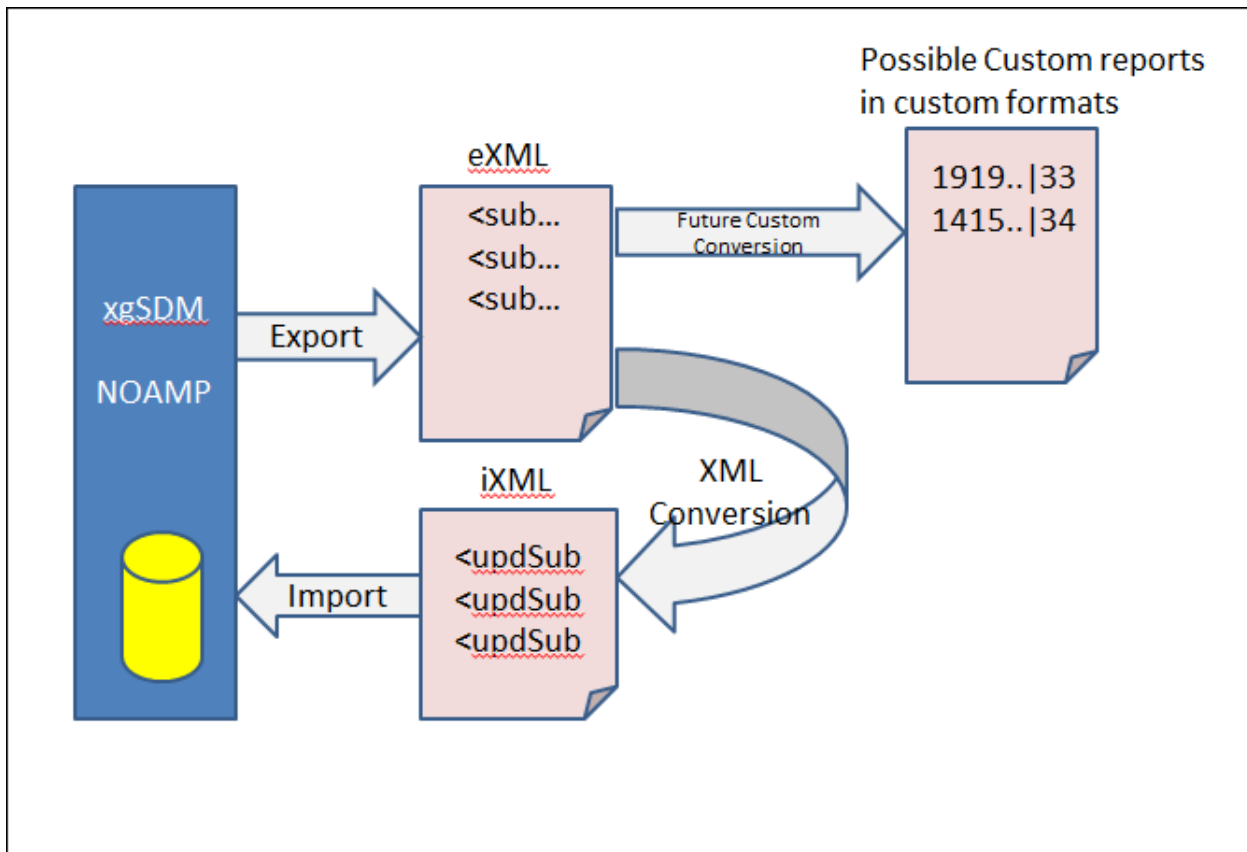
To have started the process, the user will have selected a range of MSISDN or IMSI and scheduled an export via the GUI.

The XMLExport process performs the following:

- Export process creates an output file
- Export process will lookup subscribers sequentially, including Auto-Enrolled subscribers, and output lines as follows:
 - Produce `<subscriberRecord>` line with all subscriber SDO entries
 - If a subscriber is part of a pool, `<subscriberRecord>` line will include `<poolId>` tag
 - If a subscriber is part of pool, `<poolRecord>` line is produced with all pool SDO entries.
 - Auto-Enrolled subscribers are exported with the `autoEnrolled="true"` attribute
- XML Declaration `<?xml version="1.0" encoding="UTF-8"?>` is stripped out of the retrieved data for each register.
- Entity/Service Indication name, Sequence Number and Last Update Time are not exported for each entity per subscriber.

A maximum of 30 million subscribers can be exported from the range specified. The following picture displays export in more detail:

Figure 5: Generating Output File



3.3.2 Export File and Format

Export files are created in a local directory and are transferred to a remote export host if one is configured. The local directory is always `/var/TKLC/db/filemgmt/provexport` and the remote export directory can be configured in the Provisioning Options (see Appendix B). The export file format contains the following information:

1. The exported file contains one line per subscriber with each XML entity appended to the same line (carriage returns are removed from entity value).
2. `<subscriberRecord>` and `<poolRecord>` will each take a single line.

3.3.2.1 Basic Export File Format

The following describes the basic layout of a bulk export file, with all different options and parameters included.

```
<subscriberRecord [autoEnrolled="autoEnrolled"]>
  <poolId>poolId</poolId>
  <subscriber>
    <field name="profileFieldName1">profileFieldValue1</field>
    :
    <field name="profileFieldNameN">profileFieldValueN</field>
  </subscriber>
  <entityName>
    entityData
  </entityName>
</subscriberRecord>
<poolRecord>
  <pool>
    <field name="poolProfileFieldName1">poolProfileFieldValue1</field>
    :
    <field name="poolProfileFieldNameN">poolProfileFieldValueN</field>
  </pool>
  <poolEntityName>
    poolEntityData
  </poolEntityName>
</poolRecord>
```

Note: Each `<subscriberRecord>` or `<poolRecord>` is formatted on a single line only. It is shown above expanded for readability.

3.3.2.2 Subscriber Record

One `<subscriberRecord>` is present for every subscriber that is exported, and all data for that subscriber is contained within it.

If the subscriber was auto enrolled, then the `autoEnrolled` attribute will be set to **true**. If the subscriber was NOT auto enrolled, then the `autoEnrolled` attribute is omitted.

If the subscriber is a member of a pool, then a `<poolId>` element will be present, and the PoolID of the pool to which the subscriber is a member will be set in `poolId`. If the subscriber is a member of a pool, then the corresponding `<poolRecord>` for the pool for which the subscriber is a member will also be contained in the export file.

A subscriber's Profile is stored in the **<subscriber>** element. This contains all **<field>** elements defined within the subscriber's Profile XML blob. Each defined Profile field is set in *profileFieldNameX/profileFieldValueX*.

An element exists for each entity defined for the subscriber, such as Quota, State, or DynamicQuota. All XML blob data for that entity is contained within in. For example, the element **<usage>** will be present for the Quota entity, the element **<state>** for the State entity, and **<definition>** for the DynamicQuota entity. The XML blob contents within the root element are in *entityData*.

3.3.2.3 Pool Record

One **<poolRecord>** is present for every pool that is exported, and all data for that pool is contained within it.

A pool's Profile is stored in the **<pool>** element. This contains all **<field>** elements defined within the pool's PoolProfile XML blob. Each defined PoolProfile field is set in *poolProfileFieldNameX/poolProfileFieldValueX*.

An element exists for each entity defined for the pool, such as PoolQuota, PoolState, or PoolDynamicQuota. For each entity, the *entityName* contains the root element name of the XML blob. All XML blob data for that entity is contained within in. For example, the element **<usage>** will be present for the PoolQuota entity, the element **<state>** for the PoolState entity, and **<definition>** for the PoolDynamicQuota entity. The XML blob contents within the root element are in *poolEntityData*.

3.3.2.4 Examples

Sample File Formats (lines are expanded to improve readability):

Provisioned Subscriber Record

```
<subscriberRecord>
  <poolId>1000</poolId>
  <subscriber>
    :
  </subscriber>
  <usage>
    :
  </usage>
</subscriberRecord>
```

Note: **<poolId>** tag is only present if the subscriber is a pool member.

Auto-Enrolled Subscriber Record

```
<subscriberRecord autoEnrolled="true">
  <subscriber>
    :
  </subscriber>
  <usage>
    :
  </usage>
</subscriberRecord>
```

Pool Record

```
<poolRecord>
  <pool>
    :
  </pool>
```

```

    <usage>
    :
    </usage>
</poolRecord>

```

Example Export Outputs: (lines are expanded to improve readability):

Subscriber with only Profile Entity

```

<subscriberRecord>
  <subscriber>
    <field name="MSISDN">6542896514</field>
    <field name="BillingDay">1</field>
    <field name="Tier"/>
    <field name="Entitlement">DayPass</field>
  </subscriber>
</subscriberRecord>

```

Subscriber with State and Profile Entities

```

<subscriberRecord>
  <subscriber>
    <field name="MSISDN">6542896515</field>
    <field name="BillingDay">1</field>
    <field name="Tier"/>
    <field name="Entitlement">DayPass</field>
  </subscriber>
  <state>
    <version>3</version>
    <property>
      <name>mcc</name>
      <value>315</value>
    </property>
    <property>
      <name>expire</name>
      <value>2014-02-09T11:20:32</value>
    </property>
    <property>
      <name>approved</name>
      <value>yes</value>
    </property>
  </state>
</subscriberRecord>

```

Subscriber which is a member of a Pool

```

<subscriberRecord>
  <poolId>1234</poolId>
  <subscriber>
    <field name="MSISDN">6542896515</field>
    <field name="BillingDay">1</field>
    <field name="Tier"/>
    <field name="Entitlement">DayPass</field>
  </subscriber>
  <state>
    <version>3</version>
    <property>
      <name>mcc</name>
      <value>315</value>
    </property>
    <property>
      <name>expire</name>
      <value>2010-02-09T11:20:32</value>
    </property>
  </state>

```

```

    <property>
      <name>approved</name>
      <value>yes</value>
    </property>
  </state>
</subscriberRecord>

```

Auto-Enrolled Subscriber

```

<subscriberRecord autoEnrolled="true">
  <subscriber>
    <field name="MSISDN">6542896515</field>
    <field name="BillingDay">1</field>
    <field name="Tier"/>
    <field name="Entitlement">DayPass</field>
  </subscriber>
  <state>
    <version>3</version>
    <property>
      <name>mcc</name>
      <value>315</value>
    </property>
    <property>
      <name>expire</name>
      <value>2010-02-09T11:20:32</value>
    </property>
    <property>
      <name>approved</name>
      <value>yes</value>
    </property>
  </state>
</subscriberRecord>

```

PoolRecord with only Pool Profile

```

<poolRecord>
  <pool>
    <field name="PoolID">206534</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>
</poolRecord>

```

PoolRecord with Pool Profile and Pool Quota

```

<poolRecord>
  <pool>
    <field name="PoolID">206534</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>
  <usage>
    <version>3</version>
    <quota name="DP_QUOTA_PAYG.500MB">
      <cid>5764888998014956049</cid>
      <nextResetTime>2013-04-02T00:00:00+05:00</nextResetTime>
      <totalVolume>19948458</totalVolume>
    </quota>
  </usage>
</poolRecord>

```

```
</quota>
</usage>
</poolRecord>
```

3.3.3 Export Conversion Tool (xmlconverter)

Xmlconverter is responsible for converting the exported .xml files to XML Import .ixml input files. This tool is invoked by the user if they have a need to import the exported data and works as follows:

- xmlconverter will read export file one line at a time and create import file to recreate all subscribers and pool relationships.
- xmlconverter will provide a 'create' or 'update' option

The following is the usage for this tool:

Tool Usage:

```
xmlconverter <exportFileName> <importFileName> <create|update>
```

- `exportFileName` => The file name with the absolute path which would be used as input
- `importFileName` => The file name with the absolute path which would get created as output.
- `create` => The generated import file is expected to be used on an UDR system which does not contain the exported subscribers as it generates create commands.
- `update` => The generated import file is expected to be used on an UDR system which contains the exported subscribers as it generates update commands.

E.g.:

```
./usr/TKLC/udr/bin/xmlconverter /var/tmp/ExportFile.xml /var/tmp/ImportFile.ixml create
```

Note: For Auto-Enrolled Subscribers, internal XML commands would not be generated for the profile entity. In this case, the updated internal XML commands would be generated for non-profile entities only.

Note: For an Enterprise Pool, a single `<transaction>` would be generated containing `<addPoolMember>` for each of its members. The user of the generated file is expected to modify the file as mentioned in Section 2.2.1.1

3.3.4 Configuring Export Options

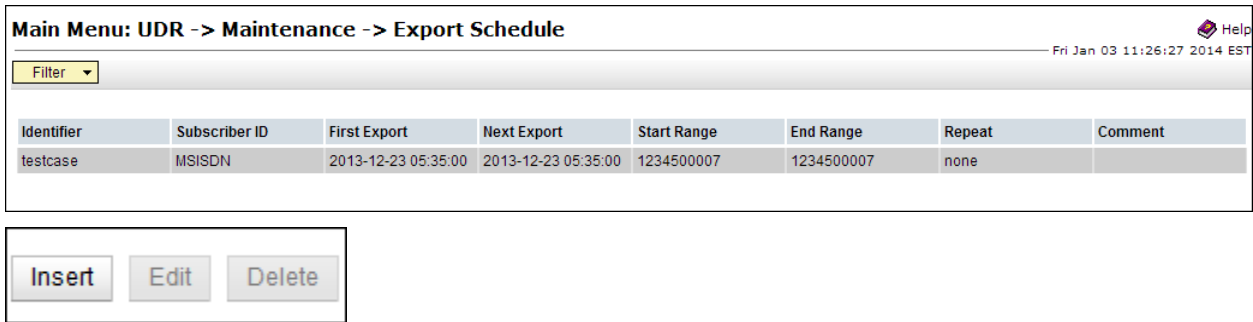
The user can configure Export options by using the UDR GUI's *Main Menu: UDR* → *Configuration* → *Provisioning Options* screen. Refer to Appendix B for Provisioning Options.

3.3.5 Scheduling Exports

The user can view the export schedule by using the UDR GUI's *Main Menu: UDR* → *Maintenance* → *Export Schedule* screen.

3.3.5.1 Display

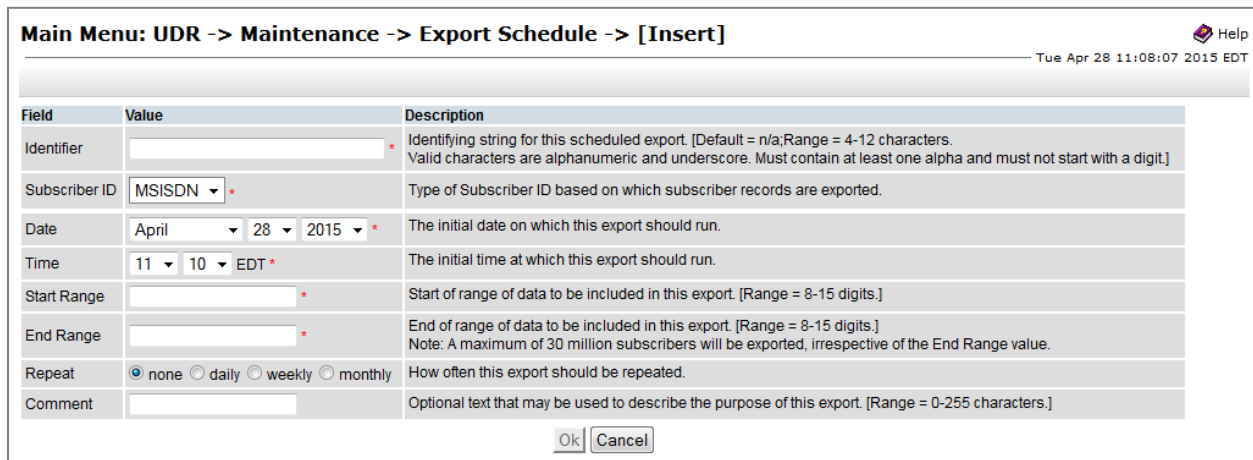
Figure 6: Export Schedule (Display)



3.3.5.2 Insert

Operators schedule exports using the *Main Menu: UDR → Maintenance → Export → Schedule->[insert]* GUI screen. On this screen, users can add a scheduled export. The user can schedule an export from GUI by specifying a range of MSISDNs or IMSIs as shown below. A maximum of 30 million subscribers would be exported from the range specified. If a range larger than 30 million subscribers is specified, the export will stop once it reaches 30 million subscribers. Exporting pools by specifying a range of PoolIDs is not supported.

Figure 7: Export Schedule (Insert)



3.3.5.3 Edit

The user can modify a scheduled export from GUI by specifying range of MSISDNs or IMSIs as shown below. A maximum of 30 million subscribers would be exported from the range specified. Exporting pools by specifying a range of PoolIDs is not supported.

Figure 8: Export Schedule (Edit)

Main Menu: UDR -> Maintenance -> Export Schedule -> [Edit] Help
Tue Apr 28 11:11:26 2015 EDT

Field	Value	Description
Identifier	ExportMsisdn *	Identifying string for this scheduled export. [Default = n/a; Range = 4-12 characters. Valid characters are alphanumeric and underscore. Must contain at least one alpha and must not start with a digit.]
Subscriber ID	MSISDN *	Type of Subscriber ID based on which subscriber records are exported.
Date	April 28 2015 *	The initial date on which this export should run.
Time	11 14 EDT *	The initial time at which this export should run.
Start Range	988888880 *	Start of range of data to be included in this export. [Range = 8-15 digits.]
End Range	9888888899 *	End of range of data to be included in this export. [Range = 8-15 digits.] Note: A maximum of 30 million subscribers will be exported, irrespective of the End Range value.
Repeat	<input checked="" type="radio"/> none <input type="radio"/> daily <input type="radio"/> weekly <input type="radio"/> monthly	How often this export should be repeated.
Comment		Optional text that may be used to describe the purpose of this export. [Range = 0-255 characters.]

Ok Cancel

3.3.5.4 Delete

Figure 9: Export Schedule (Delete)

Main Menu: UDR -> Maintenance -> Export Schedule Help
Mon Nov 25 11:28:11 2013 EST

Filter

Identifier	Export Data	First Export	Next Export	Start Range	End Range	Repeat	Comment
exportmsi	MSI	2013-11-21 06:20:00	1970-01-01 19:00:00	123456789	987654321	daily	
exportmsisdn	MSISDN	2013-11-25 12:15:00	1970-01-01 19:00:00	123456789	987654321	daily	
testExport	MSISDN	2013-11-29 10:50:00	2013-11-25 10:05:00	1111111111	5555555555	none	

The page at https://10.240.208.179 says:

Delete Export Row: exportmsi ?

OK Cancel

3.3.5.5 Export Status

The user can view the status of all in progress and completed requested exports by using the UDR GUI's *Main Menu: UDR -> Maintenance -> Export Status* screen. This screen displays the export file name, status of the export, number of export commands that succeeded and failed, comment and time stamps for when the export was queued, started and completed. The Export Status screen also provides hyperlinks so that the user can view the exported file as text or save the file locally.

Figure 10: Export Status

Main Menu: UDR -> Maintenance -> Export Status Help
Fri Apr 10 11:35:27 2015 EDT

Filter

Export File	Time Queued	Time Started	Time Completed	Subscriber Count	Pool Count	Status	Comment
export_Subst_MSIIM SI...	2015-04-10 11:35:15	2015-04-10 11:35:15	2015-04-10 11:35:16	11	0	Transferring	

Pause updates
There is 1 record matching your request.

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

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

The pool-related data includes:

- **Pool Profile:** pre-provisioned information that describes a pool
- **Pool Quota:** information that represents the pool's use of managed resources (quota, pass, top-up, roll-over)
- **Pool State:** pool-specific properties
- **Pool Dynamic Quota:** dynamically configured information related to managed resources (pass, top-up)
- **Pool Membership:** list of subscribers that are associated with a pool

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

The Index contains information on the following:

- **Subscription**
 - A subscription exists for every individual subscriber
 - 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 within an entity can be defined as mandatory, or optional. A mandatory field must exist, and cannot be deleted.

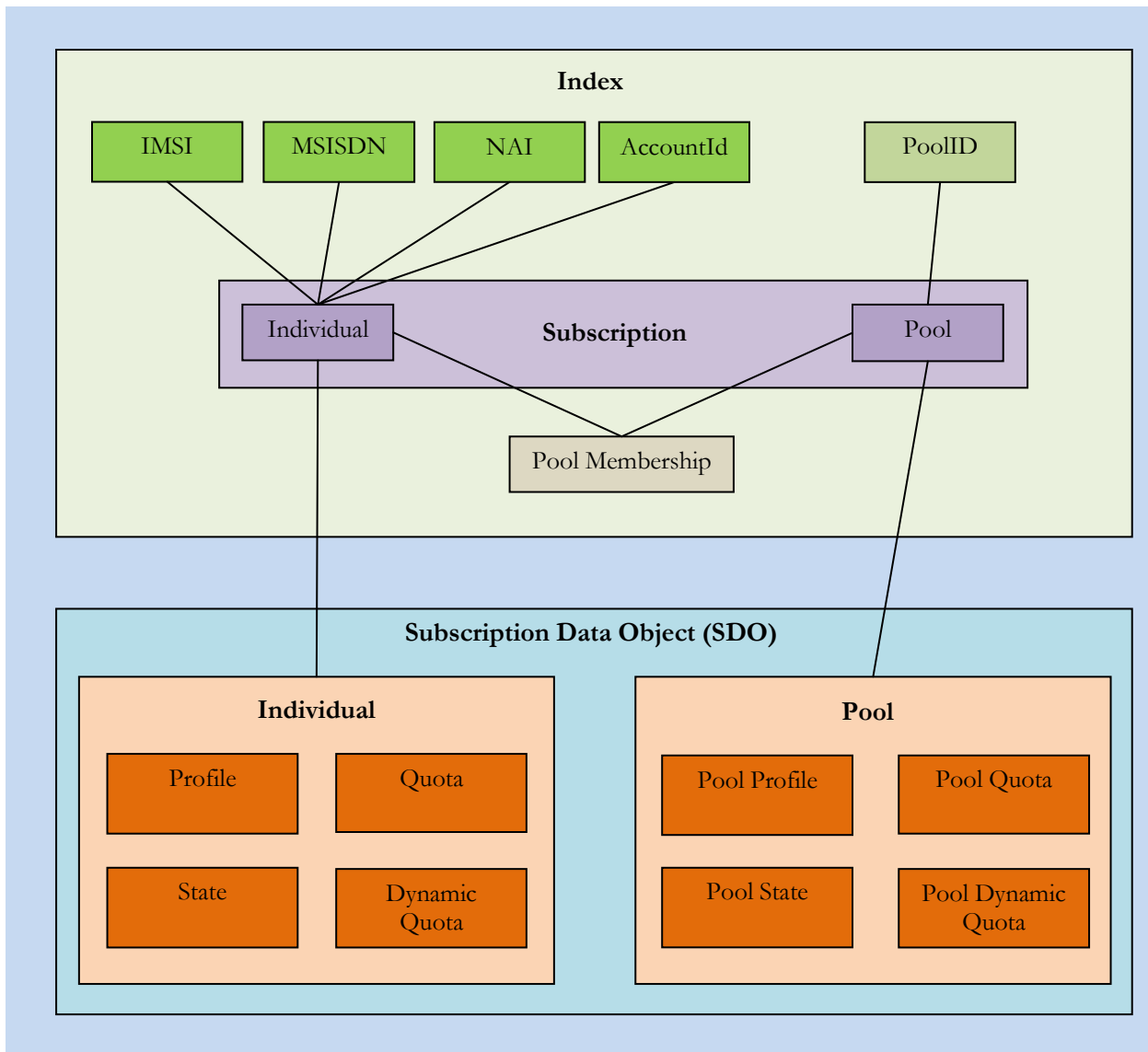
A field within an entity can have a default value. If an entity is created, and the field is not specified, it will be created with the default value.

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

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

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 11: Data Model



4.1 Subscriber Data

4.1.1 Subscriber Profile

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

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

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

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

Table 8: Subscriber Profile Entity Definition

Name (XML tag)	Type	Description
subscriber	---	Sequence (multiplicity = 1)
MSISDN	String	List of MSISDNs (8-15 numeric digits). A separate entry is included for each MSISDN associated with the subscriber's profile.
IMSI	String	List of IMSIs (10-15 numeric digits). A separate entry is included for each IMSI associated with the subscriber's profile.
NAI	String	List of NAIs (in format "user@domain", "user", or "@domain"). A separate entry is included for each NAI associated with the subscriber's profile. The user or realm can be empty. Allowed characters for the user: '!', '%', '\$', 'A'-'Z', 'a'-'z', '0'-'9', '!', '-', '_' Allowed characters for the host: 'A'-'Z', 'a'-'z', '0'-'9', '!', '-', '_' Example NAI Formats: bob, @privatecorp.example.net, fred\$@example.com, eng.example.net!nancy@example.net , eng%nancy@example.net
AccountId	String	Any string that can be used to identify the account for the subscriber (1-255 characters). Allowed values are any ASCII printable character, values x20 to x7e.
BillingDay	String	Allowed values are [0-31]. The day of the month [1-31] on which the subscriber's associated quota should be reset. [0] indicates that the default value configured at the PCRF level should be used. This is automatically set in any record where BillingDay is not specified.
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the subscriber's profile.
Tier	String	Subscriber's tier.
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.
Custom3	String	Fields used to store customer-specific data.
Custom4	String	Fields used to store customer-specific data.
Custom5	String	Fields used to store customer-specific data.
Custom6	String	Fields used to store customer-specific data.
Custom7	String	Fields used to store customer-specific data.
Custom8	String	Fields used to store customer-specific data.
Custom9	String	Fields used to store customer-specific data.
Custom10	String	Fields used to store customer-specific data.
Custom11	String	Fields used to store customer-specific data.
Custom12	String	Fields used to store customer-specific data.
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.

Name (XML tag)	Type	Description
Custom20	String	Fields used to store customer-specific data.

4.1.2 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 shall be associated with a subscriber record and supports the following sequence of attributes.

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

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

- When a Quota instance is created, and no value is supplied for the field. In this case, the field is created with the value indicated
- When a Quota instance is reset using the “Reset” command. If a field is defined as resettable, and the field currently exists, then it is set to the value indicated. If the field does not currently 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 will get created with the value indicated

Table 9: 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
cid	String	---	Internal identifier used to identify a quota within 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 4.3 for format details.	1/2/3
Type	String	Empty string ""	Type of the resource in use.	2/3
grantedTotalVolume	String	"0"	Granted Total Volume, will represent the granted total volume of all the subscribers in the pool, in case of pool quota. In case of individual quota, it	2/3

Name (XML tag)	Type	Default Value	Description	Quota Versions
			will represent the granted volume to all the PDN connections for that subscriber.	
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

4.1.3 State

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

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

Note: The default fields configured are not :

- Resettable
- Defaultable

The State entity shall support the following sequence of attributes:

Table 10: 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.

4.1.4 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 customer’s OSS system.

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

Note: The default fields configured are not :

- Resettable
- Defaultable

The DynamicQuota entity shall support the following sequence of attributes:

Table 11: 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 will be used to match top-ups to quota definitions on the PCRF. This name will be used in policy conditions and actions on the PCRF.
InstanceId	String	A unique identifier to identify this instance of a dynamic quota object.
Priority	String	An integer represented as a string. This number allows service providers to specify when one pass or top-up should be used before another pass or top-up.
InitialTime	String	An integer represented as a string. The number of seconds initially granted for the pass/top-up.
InitialTotalVolume	String	An integer represented as a string. The number of bytes of total volume initially granted for the pass/top-up.
InitialInputVolume	String	An integer represented as a string. The number of bytes of input volume initially granted for the pass/top-up.
InitialOutputVolume	String	An integer represented as a string. The number of bytes of output volume initially granted for the pass/top-up.
InitialServiceSpecific	String	An integer represented as a string. The number of service specific units initially granted for the pass/top-up.
activationdatetime	String	The date/time after which the pass or top-up may be active. See section 4.3 for format details.
expirationdatetime	String	The date/time after which the pass or top-up is considered to be exhausted See section 4.3 for format details.
purchasedatetime	String	The date/time when a pass was purchased See section 4.3 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.

4.2 Pool Data

4.2.1 Pool Profile

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

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

BillingDay must be defined with a default value if another value is not specified. The remaining fields are only included in the record if they are specified when the record is created/updated.

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

Table 12: Pool Profile Entity Definition

Name (XML tag)	Type	Description
pool	---	Sequence (multiplicity = 1)
PoolID	String	Pool identifier (1-22 numeric digits, minimum value 1)
BillingDay	Uint8	The day of the month [1-31] on which the pool's associated quota should be reset. [0] indicates that the default value configured at the PCRf level should be used.
BillingType	String	The billing frequency, monthly, weekly, daily
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the pool's profile.
Tier	String	Pool's tier.
Type	String	Field used to identify an Enterprise Pool. Allowed value is "enterprise" and is <i>not</i> 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.
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.

4.2.2 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 customer's OSS system.

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

The PoolQuota entity attributes are the same as defined for the Quota entity in section 4.1.2.

4.2.3 Pool State

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

The PoolState entity contains a version number. Different attributes may 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 4.1.3.

4.2.4 Pool Dynamic Quota

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

The PoolDynamicQuota entity contains a version number. Different attributes may 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 4.1.4.

4.3 Date/Timestamp Format

The Date/Timestamp format used by many fields is:

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

This corresponds to either:

1. *CCYY-MM-DDThh:mm:ss* (local time)
2. *CCYY-MM-DDThh:mm:ssZ* (UTC time)
3. *CCYY-MM-DDThh:mm:ss+hh:mm* (positive offset from UTC)
4. *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

The following 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)

5 Subscriber Provisioning

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

5.1 Subscriber Profile Commands

Table 13: Summary of Subscriber Profile Commands

Command	Description	Key(s)	Command Syntax
Create Subscriber	Create a new subscriber/ subscriber Profile	MSISDN, IMSI, NAI and/or AccountId	<createSubscriber>
Update Subscriber	Update subscriber Profile data		<updateSubscriber>
Delete Subscriber	Delete all subscriber Profile data and all opaque data associated with the subscriber		<deleteSubscriber>

5.1.1 Create Subscriber

Description

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

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

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

Prerequisites

A subscriber with any of the keys supplied in the <key> section must not exist.

Request

```
<createSubscriber>
  <key>
    [
      <IMSI>IMSI1</IMSI>
    [
      <IMSI>IMSI2</IMSI> ]
    [
      <IMSI>IMSI3</IMSI> ]
    ]

    [
      <MSISDN>MSISDN1</MSISDN>
    [
      <MSISDN>MSISDN2</MSISDN> ]
    [
      <MSISDN>MSISDN3</MSISDN> ]
    ]

    [
      <NAI>NAI1</NAI>
    [
      <NAI>NAI2</NAI> ]
    [
      <NAI>NAI3</NAI> ]
    ]

    [
      <AccountId>accountId</AccountId> ]
  </key>
  <entity>
    <data>
      <name>dataName</name>
      <interface>dataInterface</interface>
    </data>
    <content>
      <![CDATA[cdataProfile]]>
    </content>
  </entity>
</createSubscriber>
```

Where:

- **IMSI**: IMSI value(s) corresponding to the subscriber. No values will be present if an IMSI is not provisioned for the subscriber
 - Values: A string with 10 to 15 digits (if value is set)
- **MSISDN**: MSISDN value(s) corresponding to the subscriber. No values will be present if an MSISDN is not provisioned for the subscriber
 - Values: A string with 8 to 15 digits (if value is set)
- **NAI**: NAI value(s) corresponding to the subscriber. No values will be present if an NAI is not provisioned for the subscriber
 - Values: A string with 1 to 255 characters (if value is set)
 - **Note**: NAI is in format “user@domain”
- **accountId**: AccountId corresponding to the subscriber. This value will not be present if an AccountId is not provisioned for the subscriber
 - Values: A string with 1 to 255 characters (if value is set)
- **dataName**: A user defined entity type/name for the subscriber Profile
 - Value is Subscriber
- **dataInterface**: The interface type used to identify the bulk import/export interface
 - Value is XMLIMPORT
- **cdataProfile**: Contents of the XML data “blob” for the subscriber Profile

Note: Within <key> at least one key type is mandatory. Any combination of key types are allowed. Up to 3 occurrences of each repeatable key type (i.e. IMSI/MSISDN/NAI) is supported.

Note: Key order in the request is not important.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
ElementNotDefined	An XML Element is not defined
FieldValueNotValid	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
InvalidInputXml	Invalid Input XML
OccurrenceConstraintViolation	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
FieldDefinitionNotFound	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KeyAlreadyExists	Key Already Exists. A subscriber already exists with the given key

Examples

Request #1

A subscriber is created, with an AccountId, MSISDN and IMSI keys. The *BillingDay*, *Tier*, and *Entitlement* fields are set.

```
<createSubscriber>
  <key>
```

```

    <AccountId>10404723525</AccountId>
    <MSISDN>33123654862</MSISDN>
    <IMSI>184569547984229</IMSI>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
    </data>
    <content>
<![CDATA[
<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">DayPass</field>
  <field name="Entitlement">DayPassPlus</field>
</subscriber>
]]>
    </content>
  </entity>
</createSubscriber>

```

Response #1

The request is successful, and the subscriber was created.

Request #2

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

```

<createSubscriber>
  <key>
    <AccountId>10404723525</AccountId>
    <MSISDN>33123654862</MSISDN>
    <IMSI>184569547984229</IMSI>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
    </data>
    <content>
<![CDATA[
<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">DayPass</field>
  <field name="Entitlement">DayPassPlus</field>
</subscriber>
]]>
    </content>
  </entity>
</createSubscriber>

```

Response #2

The request fails. The *errorValue* indicates a subscriber already exists with the given IMSI.

```
[error 40 errorText : line lineNumber]
```

Request #3

A subscriber is created, with an AccountId, MSISDN and IMSI keys. The *BillingDay*, *Tier*, and *Entitlement* fields are set. **Provisioning has been disabled.**

```
<createSubscriber>
  <key>
    <MSISDN>33123654862</MSISDN>
    <IMSI>184569547984229</IMSI>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
    </data>
    <content>
<![CDATA[
<subscriber>
  <field name="MSISDN">33123654862</field>
  <field name="IMSI">184569547984229</field>
  <field name="BillingDay">1</field>
  <field name="Tier"></field>
  <field name="Entitlement">DayPass</field>
</subscriber>
]]>
    </content>
  </entity>
</createSubscriber>
```

Response #3

The request fails. The *errorValue* indicates that provisioning has been disabled.

```
[error 5 errorText : line lineNumber]
```

5.1.2 Update Subscriber

Description

This operation replaces an existing subscriber profile, for the subscriber identified by the specified key(s).

All existing data for the subscriber is completely removed and replaced by the request content.

All other subscriber keys that exist for the subscriber, apart from the one specified in `<key>`, will be replaced by those specified in `<subscriber>`.

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

Prerequisites

A subscriber with a key(s) of the *keyNameX/keyValueX* supplied must exist.

All supplied keys must reference the same subscriber.

Request


```

<updateSubscriber>
  <key>
    <keyName1>keyValue1</keyName1>
  [
    <keyName2>keyValue2</keyName2>
    :
    <keyNameN>keyValueN</keyNameN>
  ]
</key>

<subscriber>
  [
    <IMSI>IMSI1</IMSI>
  [
    <IMSI>IMSI2</IMSI> ]
  [
    <IMSI>IMSI3</IMSI> ]
  ]

  [
    <MSISDN>MSISDN1</MSISDN>
  [
    <MSISDN>MSISDN2</MSISDN> ]
  [
    <MSISDN>MSISDN3</MSISDN> ]
  ]

  [
    <NAI>NAI1</NAI>
  [
    <NAI>NAI2</NAI> ]
  [
    <NAI>NAI3</NAI> ]
  ]

  [
    <AccountId>accountId</AccountId> ]

</subscriber>

<entity>
  <data>
    <name>dataName</name>
    <interface>dataInterface</interface>
  </data>

  <content>
    <![CDATA[cdataProfile]]>
  </content>

</entity>

</updateSubscriber>

```

Where :

- **keyNameX**: A key field within the subscriber Profile
 - Value is either IMSI, MSISDN, NAI, or Account Id
- **keyValueX**: Corresponding key field value assigned to *keyName*
- **IMSIX**: IMSI value(s) corresponding to the subscriber. No values will be present if an IMSI is not provisioned for the subscriber
 - Values: A string with 10 to 15 digits (if value is set)
- **MSISDNX**: MSISDN value(s) corresponding to the subscriber. No values will be present if an MSISDN is not provisioned for the subscriber

- Values: A string with 8 to 15 digits (if value is set)
- **NAIX**: NAI value(s) corresponding to the subscriber. No values will be present if an NAI is not provisioned for the subscriber
 - Values: A string with 1 to 255 characters (if value is set)
 - Note: NAI is in format “user@domain”
- **accountId**: AccountId corresponding to the subscriber. This value will not be present if an AccountId is not provisioned for the subscriber
 - Values: A string with 1 to 255 characters (if value is set)
- **dataName**: A user defined entity type/name for the subscriber Profile
 - Value is Subscriber
- **dataInterface**: The interface type used to identify the bulk import/export interface
 - Value is XMLIMPORT
- **cdataProfile**: Contents of the XML data “blob” for the subscriber Profile

Note: In <key>, one single key value is mandatory.

Note: Multiple subscriber key values can be supplied. See section 2.4 for details.

Note: In <subscriber>, any combination of key types is allowed. Up to 3 occurrences of each repeatable key type (i.e. IMSI/MSISDN/NAI) is supported. Key values are checked to match those from the Profile XML blob supplied.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
ElementNotDefined	An XML Element is not defined
FieldValueNotValid	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OccurenceConstraintViolation	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
InvalidInputXml	Invalid Input XML
FieldDefinitionNotFound	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KeyNotFound	Key Not Found. A subscriber with the given key cannot be found
KeyAlreadyExists	Key Already Exists. A subscriber already exists with the given key
MultipleKeysNotMatch	All supplied keys do not correspond to the same subscriber

Examples

Request #1

A subscriber is updated using MSISDN. The *AccountId*, *IMSI*, *BillingDay*, *Tier*, and *Entitlement* fields are set. The subscriber exists.

```

<updateSubscriber>
  <key>
    <MSISDN>33123654862</MSISDN>
  </key>
  <subscriber>
    <AccountId>10404723525</AccountId>
    <MSISDN>33123654862</MSISDN>
    <IMSI>184569547984229</IMSI>
  </subscriber>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
    </data>
    <content>
<![CDATA[
<subscriber>
  <field name="AccountId">10404723525</field>
  <field name="MSISDN">33123654862</field>
  <field name="IMSI">184569547984229</field>
  <field name="BillingDay">6</field>
  <field name="Tier">Silver</field>
  <field name="Entitlement">DayPass</field>
</subscriber>
]]>
    </content>
  </entity>
</updateSubscriber>

```

Response #1

The request is successful, and the subscriber was updated.

Request #2

A subscriber is updated using IMSI. The *AccountId*, *IMSI*, *BillingDay*, *Tier*, and *Entitlement* fields are set. The subscriber does NOT exist.

```

<updateSubscriber>
  <key>
    <IMSI>302370123456789</IMSI>
  </key>
  <subscriber>
    <IMSI>302370123456789</IMSI>
  </subscriber>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
    </data>
    <content>
<![CDATA[
<subscriber>
  <field name="IMSI">302370123456789</field>
  <field name="BillingDay">4</field>
  <field name="Tier">Gold</field>
  <field name="Entitlement">DayPass</field>
</subscriber>
]]>
    </content>
  </entity>
</updateSubscriber>

```

Response #2

The request fails. The *errorValue* indicates a subscriber with the given IMSI does not exist.

```
[error 39 errorText : line lineNumber]
```

5.1.3 Delete Subscriber

Description

This operation deletes all profile data (field-value pairs) and opaque data for the subscriber that is identified by the specified key(s).

Prerequisites

A subscriber with a key(s) of the *keyNameX/keyValueX* supplied must exist.

All supplied keys must reference the same subscriber.

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

Request

```
<deleteSubscriber>
```

```
  <key>
```

```
    <keyName1>keyValue1</keyName1>
```

```
  [
```

```
    <keyName2>keyValue2</keyName2>
```

```
    :
```

```
    <keyNameN>keyValueN</keyNameN>
```

```
  ]
```

```
  </key>
```

```
</deleteSubscriber>
```

- **keyNameX**: A key field within the subscriber Profile
 - Value is either IMSI, MSISDN, NAI, or Account Id
- **keyValueX**: Corresponding key field value assigned to *keyName*

Note: Multiple subscriber key values can be supplied. See section 2.4 for details.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
KeyNotFound	Key Not Found. A subscriber with the given key cannot be found
SubscriberIsPoolMember	Subscriber is Pool Member. The subscriber is a member of a pool. A subscriber cannot be deleted if they are a pool member
MultipleKeysNotMatch	All supplied keys do not correspond to the same subscriber

Examples

Request #1

The subscriber with the given MSISDN is deleted. The subscriber exists.

```
<deleteSubscriber>
  <key>
    <MSISDN>33123654862</MSISDN>
  </key>
</deleteSubscriber>
```

Response #1

The request is successful, and the subscriber was deleted.

Request #2

The subscriber with the given MSISDN is deleted. The subscriber does NOT exist.

```
<deleteSubscriber>
  <key>
    <MSISDN>33123655555</MSISDN>
  </key>
</deleteSubscriber>
```

Response #2

The request fails. The *errorValue* indicates a subscriber with the given MSISDN does not exist.

```
[error 39 errorText : line lineNumber]
```

6 Pool Provisioning

Pools are used to group subscribers that share common data. Subscribers in a pool share all the entities of that pool.

Via bulk import, provisioning clients can create, modify, and delete pool data. Pool data is accessed via the PoolID value associated with the pool.

Note: Modifying a Pool is done by using the `<updateFieldSet>` command as described in section 7.1.3.

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

6.1 Pool Profile Commands

Table 14: Summary of Pool Profile Commands

Command	Description	Key(s)	Command Syntax
Create Pool	Creates a new pool profile using the field-value pairs that are specified in the request content.	PoolID	<code><createPool></code>
Delete Pool	Delete pool profile data and all opaque data associated with the pool		<code><deletePool></code>

6.1.1 Create Pool

Description

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

Note: The PoolID key value should be specified identically in BOTH the `<key>` section AND in the PoolProfile XML blob. The value specified in the `<key>` section is used to create the pool and define what value is used in the `<key>` section for subsequent requests. The value in the PoolProfile XML blob is simply stored and returned if requested.

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

Prerequisites

A pool with a key of *poolId* in the `<key>` section must not exist.

Request

```
<createPool>

  <key>
    <PoolID>poolId</PoolID>
  </key>

  <entity>
    <data>
      <name>dataName</name>
      <interface>dataInterface</interface>
    </data>

    <content>
      <![CDATA[cdataPoolProfile]]>
    </content>
  </entity>
```

```
</createPool>
```

Where:

- **poolId**: PoolID value of the pool. Numeric value, 1-22 digits in length
 - Values: 1-9999999999999999999999
- **dataName**: A user defined entity type/name for the pool Profile
 - Value is Pool
- **dataInterface**: The interface type used to identify the bulk import/export interface
 - Value is XMLIMPORT
- **cdataPoolProfile**: Contents of the XML data “blob” for the pool Profile

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
ElementNotDefined	An XML Element is not defined
FieldValueNotValid	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
InvalidInputXml	Invalid Input XML.
OccurrenceConstraintViolation	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
FieldDefinitionNotFound	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KeyAlreadyExists	Key Already Exists. A pool already exists with the given key

Examples

Request #1

A pool is created, with PoolID. The *BillingDay* and *Entitlement* fields are set.

```
<createPool>
  <key>
    <PoolID>100000</PoolID>
  </key>
  <entity>
    <data>
      <name>Pool</name>
      <interface>XMLIMPORT</interface>
    </data>
    <content>
<![CDATA[
<pool>
  <field name="PoolID">100000</field>
  <field name="BillingDay">1</field>
  <field name="Entitlement">DayPass</field>
  <field name="Entitlement">DayPassPlus</field>
</pool>
]]>
```

```

    </content>
  </entity>
</createPool>

```

Response #1

The request is successful, and the subscriber was created.

Request #2

A pool is created. Another pool already exists with the given *PoolID*.

```

<createPool>
  <key>
    <PoolID>200000</PoolID>
  </key>
  <entity>
    <data>
      <name>Pool</name>
      <interface>XMLIMPORT</interface>
    </data>
    <content>
<![CDATA[
<pool>
  <field name="PoolID">200000</field>
  <field name="BillingDay">7</field>
  <field name="Entitlement">DayPass</field>
</pool>
]]>
    </content>
  </entity>
</createPool>

```

Response #2

The request fails. The *errorValue* indicates a pool already exists with the given *PoolID*.

```
[error 40 errorText : line lineNumber]
```

6.1.2 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 will fail.

Request

```

<deletePool>
  <key>
    <PoolID>poolId</PoolID>
  </key>

```


</deletePool>

- **poolId:** PoolID value of the pool. Numeric value, 1-22 digits in length
 - Values: 1-99999999999999999999

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
KeyNotFound	Key Not Found. A subscriber with the given key cannot be found
PoolNotEmpty	Has Pool Members. A pool cannot be deleted when it has member subscribers

Examples

Request #1

The pool with the given PoolID is deleted. The pool exists.

```
<deletePool>
  <key>
    <PoolID>100000</PoolID>
  </key>
</deletePool>
```

Response #1

The request is successful, and the pool was deleted.

Request #2

The pool with the given PoolID is deleted. The pool does NOT exist.

```
<deletePool>
  <key>
    <PoolID>200000</PoolID>
  </key>
</deletePool>
```

Response #2

The request fails. The *errorValue* indicates a pool with the given PoolID does not exist.

```
[error 39 errorText : line lineNumber]
```

6.2 Additional Pool Commands

Table 15: Summary of Additional Pool Commands

Command	Description	Key(s)	Command Syntax
---------	-------------	--------	----------------

Command	Description	Key(s)	Command Syntax
Add Member to Pool	Add subscriber to a Pool	PoolID and (MSISDN, IMSI, NAI or AccountId)	<addPoolMember>
Remove Member from Pool	Remove subscriber from a Pool		<deletePoolMember>

6.2.1 Add Member to Pool

Description

This operation adds one or more subscribers to a Pool.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

Separate subscriber(s) with the key(s) of the *keyNameX/keyValueX* supplied must exist.

Each subscriber must not already be a member of a pool.

The pool must have less than the maximum number of member subscribers allowed.

Request

<addPoolMember>

<key>

<PoolID>*poolId*</PoolID>

</key>

<members>

<member>

<*subKeyName1*>*subKeyValue1*</*subKeyName1*>

</member>

[

<member>

<*subKeyName2*>*subKeyValue2*</*subKeyName2*>

</member>

:

<member>

<*subKeyName10*>*subKeyValue10*</*subKeyName10*>

</member>

]

</members>

</addPoolMember>

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
 - Values: 1-99999999999999999999999999999999
- *subKeyNameX*: A key field within the subscriber Profile
 - Value is either IMSI, MSISDN, NAI, or Account Id
- *subKeyValueX*: Corresponding key field value assigned to *subKeyNameX*

Note: Up to 25 subscribers can be added in one request.

Note: The number of subscribers being added must not cause the number of members in a basic pool to exceed the maximum allowed value, else the request will fail.

Note: If any subscriber specified is currently a member of a pool, the request will fail.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
KeyNotFound	Key Not Found. A subscriber with the given key cannot be found
MemberAlreadyExists	Already a Pool Member. The subscriber is already a member of a pool
PoolNotFound	Pool does not exist. A subscriber cannot be added or removed from a pool that does not exist
MaxMembersBasicPool	Basic Pool Member List Maximum Limit Reached

Examples

Request #1

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is not already a member of a pool.

```
<addPoolMember>
  <key>
    <PoolID>100000</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>33123654862</MSISDN>
    </member>
  </members>
</addPoolMember>
```

Response #1

The request is successful, and the subscriber is added to the pool.

Request #2

A request is made to add a subscriber to a pool. The pool exists, but the subscriber does not.

```
<addPoolMember>
  <key>
    <PoolID>200002</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>15141234567</MSISDN>
    </member>
  </members>
</addPoolMember>
```

Response #2

The request fails. The *errorValue* indicates that the subscriber does not exist.

```
[error 39 errorText : line lineNumber]
```

Request #3

A request is made to add a subscriber to a pool. The subscriber exists, but the pool does not.

```
<addPoolMember>
  <key>
    <PoolID>300003</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>33123654862</MSISDN>
    </member>
  </members>
</addPoolMember>
```

Response #3

The request fails. The *errorValue* indicates that the pool does not exist.

```
[error 53 errorText : line lineNumber]
```

Request #4

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is already a member of a pool.

```
<addPoolMember>
  <key>
    <PoolID>200000</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>33123654862</MSISDN>
    </member>
  </members>
</addPoolMember>
```

Response #4

The request fails. The *errorValue* indicates the subscriber is already a member of a pool.

```
[error 43 errorText : line lineNumber]
```

Request #5

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is not a member of a pool. The basic pool has the maximum number of members allowed.

```
<addPoolMember>
  <key>
    <PoolID>400000</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>33123654862</MSISDN>
    </member>
  </members>
</addPoolMember>
```

Response #5

The request fails. The *errorValue* indicates the basic pool has the maximum number of members allowed.

```
[error 68 errorText : line lineNumber]
```

Request #6

A request is made to add 3 subscribers to a pool. The pool and all subscribers exist. No subscribers are already a member of a pool.

```
<addPoolMember>
  <key>
    <PoolID>800000</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>15145551234</MSISDN>
    </member>
    <member>
      <IMSI>302370123456789</IMSI>
    </member>
    <member>
      <MSISDN>14162221234</MSISDN>
    </member>
  </members>
</addPoolMember>
```

Response #6

The request is successful, and the 3 subscribers are added to the pool.

6.2.2 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 subscriber(s) with the key(s) of the *keyNameX/keyValueX* supplied must exist.

Each subscriber must be a member of the specified pool.

Request

```
<deletePoolMember>
  <key>
    <PoolID>poolId</PoolID>
  </key>
  <members>
    <member>
      <subKeyName1>subKeyValue1</subKeyName1>
    </member>
  </members>
```

```
[
  <member>
    <subKeyName2>subKeyValue2</subKeyName2>
  </member>
  :
  <member>
    <subKeyName10>subKeyValue10</subKeyName10>
  </member>
]

</members>
</deletePoolMember>
```

- **poolId:** PoolID value of the pool. Numeric value, 1-22 digits in length
 - Values: 1-9999999999999999999999
- **subKeyNameX:** A key field within the subscriber Profile
 - Value is either IMSI, MSISDN, NAI, or AccountId
- **subKeyValueX:** Corresponding key field value assigned to *subKeyNameX*

Note: Up to 25 subscribers can be removed in one request.

Note: If any subscriber specified is not a member of the pool, the request will fail.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
KeyNotFound	Key Not Found. A subscriber with the given key cannot be found
NotAPoolMember	Not A Pool Member
PoolNotFound	Pool does not exist. A subscriber cannot be added 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.

```
<deletePoolMember>
  <key>
    <PoolID>100000</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>33123654862</MSISDN>
    </member>
  </members>
</deletePoolMember>
```

Response #1

The request is successful, and the subscriber is removed from the pool.

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.

```
<deletePoolMember>
  <key>
    <PoolID>200000</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>33123654862</MSISDN>
    </member>
  </members>
</deletePoolMember>
```

Response #2

The request fails. The *errorValue* indicates the subscriber is not a member of the pool.

```
[error 45 errorText : line lineNumber]
```

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.

```
<deletePoolMember>
  <key>
    <PoolID>800000</PoolID>
  </key>
  <members>
    <member>
      <MSISDN>15145551234</MSISDN>
    </member>
    <member>
      <IMSI>302370123456789</IMSI>
    </member>
    <member>
      <MSISDN>14162221234</MSISDN>
    </member>
  </members>
</deletePoolMember>
```

Response #3

The request is successful, and the 3 subscribers are removed from the pool.

7 General Provisioning

Note: for command responses, the error code values described are listed in section Appendix A.

7.1 General Editing Commands

Note: Data row/field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a row based command on an entity defined as opaque will result in a **NonEmptyXPathForOpaqueData** error being returned. Attempting to perform a field based command on an entity defined as opaque will result in an **EntityDefinitionNotFound** error being returned.

Table 16: Summary of General Editing Commands

Command	Description	Key(s)	Command Syntax
Create Data	Create data of the specified type	MSISDN, IMSI, NAI, AccountId, or PoolID	<create>
Update Field	Update field(s) to the specified value(s)		<updateField>
Update FieldSet	Update row or entire entity		<updateFieldSet>
Delete Field	Delete instance(s) of the specified field(s)		<deleteField>
Delete FieldSet	Delete row or entire entity		<deleteFieldSet>

7.1.1 Create Data

Description

This operation creates an entity or row for the subscriber/pool identified by the specified key(s).

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

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

Prerequisites

A subscriber/pool with the key(s) of the *keyNameX/keyValueX* values supplied must exist.

The supplied *dataName* must be a valid interface entity name for a subscriber/pool.

When creating an entity, no entity of the *dataName* must already exist for the subscriber/pool.

Any supplied *dataXPath* must reference a valid field set within the entity/row for the subscriber/pool.

For subscriber based requests, all supplied keys must reference the same subscriber.

Request

```
<create createEntityIfNotExist="createEntityIfNotExist">
```

```
  <key>
```

```
    <keyName1>keyValue1</keyName1>
```

```
  [
```

```
    <keyName2>keyValue2</keyName2>
```

```
    :
```

```
    <keyNameN>keyValueN</keyNameN>
```

```
  ]
```

```
  </key>
```

```
  <entity>
```



```

<data>
  <name>dataName</name>
  <interface>dataInterface</interface>
  <xpath>dataXPath</xpath>
</data>

<content> <![CDATA[
  entityContent
]]></content>

</entity>

</create>

```

- ***createEntityIfNotExist***: Indicates whether the entity should be created if it does not already exist before creating the entity/row (for example if a Quota row is being created, and the Quota entity does not currently exist for the subscriber)
 - Value is either true or false
- ***keyNameX***: A key field within the subscriber Profile or pool Profile
 - Value is either IMSI, MSISDN, NAI, AccountId, or PoolID
- ***keyValueX***: Corresponding key field value assigned to *keyNameX*
- ***dataName***: A user defined entity type/name for the transparent entity being updated
 - Value is either Subscriber, Quota, State, DynamicQuota, Pool, PoolQuota, PoolState, or PoolDynamicQuota
- ***dataInterface***: The interface type used to identify the bulk import/export interface
 - Value is XMLIMPORT
- ***dataXPath***: XML XPath value which corresponds to the root element within the entity for which the row element will be created, or empty when creating an entire entity
 - Value is ***"/usage"*** for a Quota or PoolQuota row
 - Value is ***"/definition"*** for a DynamicQuota or PoolDynamicQuota row
 - Value is ***"/state"*** for a State or PoolState row
- ***entityContent***: Content of entity/row being created

Note: For subscriber based requests, multiple subscriber key values can be supplied. See section 2.4 for details.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
InterfaceEntityNameNotFound	Interface Entity Not Found
ElementNotDefined	An XML Element is not defined
FieldValueNotValid	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OccurrenceConstraintViolation	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
InvalidInputXml	Invalid Input XML

Error Code	Description
FieldDefinitionNotFound	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
KeyNotFound	Key Not Found. A subscriber/pool with the given key cannot be found
NonEmptyXPathForOpaqueData	XPath cannot be non-empty for an Opaque-data operation
RegisterDataNotFound	Register Data Not Found
OperationNotAllowed	Operation Not Allowed
MultipleKeysNotMatch	All supplied keys do not correspond to the same subscriber

Examples

Request #1

A request is made to create the *Quota* opaque data. The Quota XML blob is supplied whole.

```
<create createEntityIfNotExist="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath/>
    </data>
    <content>
<![CDATA[
<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>
    <Type>pass</Type>
    <RefInstanceId>184569547984765</RefInstanceId>
  </quota>
</usage>
]]>
      </content>
    </entity>
  </create>
```

Response #1

The request is successful, and the Quota opaque data was created.

Request #2

A request is made to create the *State* opaque data. The State XML blob is supplied whole. Two valid keys are supplied for the subscriber.

```
<create createEntityIfNotExist="true">
  <key>
```

```

    <MSISDN>15141234567</MSISDN>
    <IMSI>302370123456789</IMSI>
  </key>
  <entity>
    <data>
      <name>State</name>
      <interface>XMLIMPORT</interface>
      <xpath/>
    </data>
    <content>
<![CDATA[
<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>
]]>
    </content>
  </entity>
</create>

```

Response #2

The request is successful, and the State opaque data was created.

Request #3

A request is made to create a row in the *Quota* opaque data. The Quota opaque data exists for the subscriber.

```

<create createEntityIfNotExist="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage</xpath>
    </data>
    <content>
<![CDATA[
<quota name="NewQuota">
  <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>
  <Type>pass</Type>
  <RefInstanceId>184569547984765</RefInstanceId>
</quota>
]]>
    </content>

```

```

  </entity>
</create>

```

Response #3

The request is successful, and the Quota row data was created.

Request #4

A request is made to create a row in the *Quota* opaque data. The *Quota* opaque data does NOT exist for the subscriber. The request indicates that the Quota entity should NOT be created if it does not exist.

```

<create createEntityIfNotExist="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage</xpath>
    </data>
    <content>
<![CDATA[
<quota name="NewQuota">
  <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>
  <Type>pass</Type>
  <RefInstanceId>184569547984765</RefInstanceId>
</quota>
]]>
    </content>
  </entity>
</create>

```

Response #4

The request fails. The *errorValue* indicates the opaque data type does not exist.

```
[error 47 errorText : line lineNumber]
```

Request #5

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

```

<create createEntityIfNotExist="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Location</name>
      <interface>XMLIMPORT</interface>
      <xpath/>
    </data>
    <content>
<![CDATA[

```

```

<location>
  <town>Montreal</town>
  <province>Quebec</province>
  <country>Canada</country>
</location>
]]>
  </content>
</entity>
</create>

```

Response #5

The request fails. The *errorValue* indicates the opaque data type is invalid.

```
[error 11 errorText : line lineNumber]
```

Request #6

A request is made to create a row in the *PoolDynamicQuota* opaque data. The *PoolDynamicQuota* opaque data exists for the pool.

```

<create createEntityIfNotExist="false">
  <key>
    <PoolID>100000</PoolID>
  </key>
  <entity>
    <data>
      <name>PoolDynamicQuota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/definition</xpath>
    </data>
    <content>
<![CDATA[
<DynamicQuota name="NewDynamicQuota">
  <Type>Roll-over</Type>
  <InstanceId>15678</InstanceId>
  <Priority>4</Priority>
  <InitialTime>135</InitialTime>
  <InitialTotalVolume>2000</InitialTotalVolume>
  <InitialInputVolume>1500</InitialInputVolume>
  <InitialOutputVolume>500</InitialOutputVolume>
  <InitialServiceSpecific>4</InitialServiceSpecific>
  <activationdatetime>2015-03-09T11:20:32</activationdatetime>
  <expirationdatetime>2015-04-09T11:20:32</expirationdatetime>
  <InterimReportingInterval>100</InterimReportingInterval>
  <Duration>10</Duration>
</DynamicQuota>
]]>
    </content>
  </entity>
</create>

```

Response #6

The request is successful, and the *PoolDynamicQuota* row data was created.

7.1.2 Update Field**Description**

This operation updates a field(s) to the specified values within an entity, or row within an entity, for the subscriber/pool identified by the specified key(s), in the specified transparent entity.

For multiple value fields:

- Multiple values are specified by repeating the appropriate element, one instance per value.
- If the `clearAll` attribute is set to `true`, then all existing values are removed, and only the new values(s) specified are inserted. For example, if the current value of a field was “a,b,c”, and this command was used with value “d”, after the update the field would have the value “d” (it would NOT be “a,b,c,d”)
- If the `clearAll` attribute is set to `false`, then all existing values are retained, and the new values(s) specified are inserted. For example, if the current value of a field was “a,b,c”, and this command was used with value “d”, after the update the field would have the value “a,b,c,d”)

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

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

Note: It is possible to also delete a field(s) in the update request by specifying the `delete="true"` attribute. A specific value can also be specified so that the field (or value) will only be deleted if it matches the value supplied.

Note: If one or more key (i.e. IMSI/MSISDN/NAI/AccountId) field (or values) are deleted for a subscriber, then afterwards, the subscriber must still have at least one key type/value remaining or the request will fail.

Prerequisites

A subscriber/pool with the key(s) of the `keyNameX/keyValueX` values supplied must exist.

Each requested field `fieldName` must be a valid field in the transparent entity being updated.

The supplied `dataName` must be a valid interface entity name for a subscriber/pool.

The supplied `dataXPath` must reference a valid XML XPath where the specified fields in `<fields>` exist within the transparent entity for the subscriber/pool.

For subscriber based requests, all supplied keys must reference the same subscriber.

Request

```
<updateField clearAll="clearAll">
```

```
  <key>
    <keyName1>keyValue1</keyName1>
  [
    <keyName2>keyValue2</keyName2>
    :
    <keyNameN>keyValueN</keyNameN>
  ]
</key>

  <entity>
    <data>
      <name>dataName</name>
      <interface>dataInterface</interface>
      <xpath>dataXPath</xpath>
    </data>

    <fields>
```

```

    <field name="fieldName1" [delete="deleteField1"]>[fieldValue1]</field>
  [
    <field name="fieldName2" [delete="deleteField2"]>[fieldValue2]</field>
    :
    <field name="fieldName250" [delete="deleteField250"]>[fieldValue250]</field>
  ]
</fields>

</entity>

</updateField>

```

- **clearAll:** Indicates whether all existing value(s) in the field(s) being updated should first be removed before adding the newly specified field value(s)
 - Value is either `true` or `false`
 - **Note:** For fields that are not multi-value (i.e. single value), the value of *clearAll* must be set to `true` else the request will attempt to add a second instance of the field, and the request will fail
- **keyNameX:** A key field within the subscriber Profile or pool Profile
 - Value is either IMSI, MSISDN, NAI, AccountId, or PoolID
- **keyValueX:** Corresponding key field value assigned to *keyNameX*
- **dataName:** A user defined entity type/name for the transparent entity being updated
 - Value is either Subscriber, Quota, Pool, or PoolQuota
- **dataInterface:** The interface type used to identify the bulk import/export interface
 - Value is XMLIMPORT
- **dataXPath:** XML XPath expression identifying the base element containing the fields to be updated
 - Value is `"/usage/quota[@name='quotaName']"` for a Quota or PoolQuota row without an instance specified
 - Value is `"/usage/quota[@name='quotaName' and cid='quotaCid']"` for a Quota or PoolQuota row with an instance specified
 - Value is `"/usage/quota[@name='quotaName' and Type='quotaType']"` for a Quota or PoolQuota row with a type specified
 - Value is `"/definition/DynamicQuota[@name='dynamicQuotaName']"` for a DynamicQuota or PoolDynamicQuota row with the name specified
 - Value is `"/definition/DynamicQuota[@name='dynamicQuotaName' and InstanceId='dynamicQuotaInstanceId']"` for a DynamicQuota or PoolDynamicQuota row with an instance specified
 - Value is `"/definition/DynamicQuota[@name='dynamicQuotaName' and Type='dynamicQuotaType']"` for a DynamicQuota or PoolDynamicQuota row with a type specified
 - Value is `"/state/property[name='propertyName']"` for a State or PoolState row with the property name specified
- **fieldNameX:** A user defined field within the transparent entity being updated
- **fieldValueX:** (Optional) Corresponding field value assigned to *fieldNameX*
 - **Note:** This can be omitted if entire field is to be deleted, or can also be used to delete a field with the specified value
- **deleteFieldX:** (Optional) Indicates that field should be deleted, not updated
 - Value is either `true` or `false`
 - Default value is `false` if attribute is omitted

- **quotaName:** (See *dataXpath*) The name that identifies the required quota row within the Quota/PoolQuota entity
- **quotaCid:** (See *dataXpath*) The cid value that identifies the specific required quota row within the Quota/PoolQuota entity
- **quotaType:** (See *dataXpath*) The type value that identifies the specific required quota row within the Quota/PoolQuota entity
- **dynamicQuotaName:** (See *dataXpath*) The name that identifies the required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **dynamicQuotaInstanceId:** (See *dataXpath*) The instance value that identifies the specific required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **dynamicQuotaType:** (See *dataXpath*) The type that identifies the required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **propertyName:** (See *dataXpath*) The name that identifies the required state property within the State/PoolState entity

Note: A maximum of 250 fields can be updated in a single <updateField> request.

Note: For subscriber based requests, multiple subscriber key values can be supplied. See section 2.4 for details.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
InterfaceEntityNameNotFound	Interface Entity Not Found
ElementNotDefined	An XML Element is not defined
FieldValueNotValid	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
FieldSetNotFound	Field Set Not Found
FieldAlreadyExists	Field Already Exists
FieldNotMultiValued	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
FieldSetDefinitionNotFound	Field Set Not Defined
FieldDefinitionNotFound	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FieldNotUpdatable	Field Cannot be Updated. The field is defined in the SEC as not be updatable
MultipleRowsFound	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
KeyNotFound	Key Not Found. A subscriber/pool with the given key cannot be found
KeyAlreadyExists	Key Already Exists. A subscriber/pool already exists with the given key
RegisterDataNotFound	Register Data Not Found

Error Code	Description
OperationNotAllowed	Operation Not Allowed
OneKeyRequired	A subscriber must have at least one key value
MultipleKeysNotMatch	All supplied keys do not correspond to the same subscriber

Examples

Request #1

A request is made to update a subscriber Profile, and set the value of the *BillingDay* field to 23, and the *Tier* field to *Gold*.

```
<updateField clearAll="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="BillingDay">23</field>
      <field name="Tier">Gold</field>
    </fields>
  </entity>
</updateField>
```

Response #1

The request is successful, and the *BillingDay* and *Tier* values were updated.

Request #2

A request is made to update a subscriber Profile, and set the value of the *BillingDay* field to 55.

```
<updateField clearAll="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="BillingDay">55</field>
    </fields>
  </entity>
</updateField>
```

Response #2

The request fails. The *errorValue* indicates the value of *BillingDay* was invalid.

```
[error 18 errorText : line lineNumber]
```

Request #3

A request is made to update the *inputVolume* and the *outputVolume* fields within the *Q1* Quota row within the Quota entity.

```
<updateField clearAll="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1']</xpath>
    </data>
    <fields>
      <field name="inputVolume">3000</field>
      <field name="outputVolume">2500</field>
    </fields>
  </entity>
</updateField>
```

Response #3

The request is successful, and the *inputVolume* and *outputVolume* values were updated.

Request #4

A request is made to update the *inputVolume* and the *outputVolume* fields within the *Q1* Quota row within the Quota entity. Two rows called *Q1* exist, one with a *cid* of 111 and another with a *cid* of 222. The request is to update the instance with the *cid* of 111.

```
<updateField clearAll="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1' and cid='111']</xpath>
    </data>
    <fields>
      <field name="inputVolume">3000</field>
      <field name="outputVolume">2500</field>
    </fields>
  </entity>
</updateField>
```

Response #4

The request is successful, and the *inputVolume* and *outputVolume* values were updated in the *Q1* row containing a *cid* of 111.

Request #5

A request is made to update a subscriber Profile, and add the value *EveningPass* to the multi-value field *Entitlement* retaining all existing values. The current value of the field is “*DayPass,Weekend*”.

```
<updateField clearAll="false">
  <key>
    <MSISDN>15141234567</MSISDN>
```

```

</key>
<entity>
  <data>
    <name>Subscriber</name>
    <interface>XMLIMPORT</interface>
    <xpath>/subscriber</xpath>
  </data>
  <fields>
    <field name="Entitlement">EveningPass</field>
  </fields>
</entity>
</updateField>

```

Response #5

The request is successful, and the *Entitlement* field was updated. The value of the field is now “*DayPass,Weekend,EveningPass*”.

Request #6

A request is made to update a subscriber Profile, and set the multi-value field *Entitlement* to be only *Weekend*, removing all other existing values. The current value of the field is “*DayPass,Weekend,EveningPass*”.

```

<updateField clearAll="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="Entitlement">Weekend</field>
    </fields>
  </entity>
</updateField>

```

Response #6

The request is successful, and the *Entitlement* field was updated. The value of the field is now “*Weekend*”.

Request #7

A request is made to update a subscriber Profile, and add two additional MSISDN values. Currently, the subscriber only has the MSISDN “*15141234567*”.

```

<updateField clearAll="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="MSISDN">14161112222</field>
      <field name="MSISDN">14505556666</field>
    </fields>
  </entity>
</updateField>

```

```

</entity>
</updateField>

```

Response #7

The request is successful, and the two additional MSISDNs were added. The subscriber now has three MSISDNs, “15141234567”, “14161112222”, and “14505556666”.

Request #8

A request is made to update a subscriber Profile, and replace the 3 existing IMSI values “302370123456789”, “302370999888777”, and “302370555555555” with a single value of “302370111111111”.

```

<updateField clearAll="true">
  <key>
    <IMSI>302370123456789</IMSI>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="IMSI">302370111111111</field>
    </fields>
  </entity>
</updateField>

```

Response #8

The request is successful, and subscriber now has a single IMSI, “302370111111111”.

Request #9

A request is made to update a subscriber Profile. The request will replace all exist IMSI value(s) with a single value, delete the specific MSISDN “14161112222”, delete all instance(s) of the NAI, update the *BillingDay* field, delete the *Tier* field if the existing value is set to “Gold”, and delete the *Custom5* field.

```

<updateField clearAll="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="IMSI">302370111111111</field>
      <field name="MSISDN" delete="true">14161112222</field>
      <field name="NAI" delete="true"/>
      <field name="BillingDay">11</field>
      <field name="Tier" delete="true">Gold</field>
      <field name="Custom5" delete="true"/>
    </fields>
  </entity>
</updateField>

```

Response #9

The request is successful, and subscriber now has a single IMSI, “302370111111111”.

Request #10

A request is made to update the *InitialInputVolume* and the *InitialOutputVolume* fields within the *DQ1* DynamicQuota row within the DynamicQuota entity. Two rows called *DQ1* exist, one with an *InstanceId* of 11111 and another with an *InstanceId* of 22222. The request is to update the instance with the *InstanceId* of 11111.

```
<updateField clearAll="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>DynamicQuota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/definition/DynamicQuota[@name='DQ1' and InstanceId='11111']</xpath>
    </data>
    <fields>
      <field name="InitialInputVolume">3000</field>
      <field name="InitialOutputVolume">2500</field>
    </fields>
  </entity>
</updateField>
```

Response #10

The request is successful, and the *InitialInputVolume* and *InitialOutputVolume* values were updated in the *DQ1* row containing an *InstanceId* of 11111.

Request #11

A request is made to update a property value within the PoolState entity. Two properties exist, one with a property *name* of *mcc* and another with *name* of *approved*. The request is to update the property value with the *name* of *mcc*.

```
<updateField clearAll="true">
  <key>
    <PoolID>10000</PoolID>
  </key>
  <entity>
    <data>
      <name>PoolState</name>
      <interface>XMLIMPORT</interface>
      <xpath>/state/property[name='mcc']</xpath>
    </data>
    <fields>
      <field name="value">3000</field>
    </fields>
  </entity>
</updateField>
```

Response #11

The request is successful, and the property *value* was updated in the PoolState containing a property *name* of *mcc*.

7.1.3 Update FieldSet

Description

This operation updates or creates an entity or row within an entity for the subscriber or pool identified by the specified key(s), for the specified transparent entity. This operation replaces ("sets") the entire content of the entity/row, which means that any existing values are deleted first.

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

Note: When an entire entity is created during a request to update a row, if the transparent entity is versioned, then it is necessary for UDR to know *which* version of the transparent entity should be created.

- If no `<version>` element is supplied in the request, then :
 - If an entity is not versioned, then the non versioned definition will be used
 - If only one version definition exists in the SEC, then that version will be used
 - If multiple version definitions exists in the SEC, then the version with the alphabetically greater value will be used (i.e. "v3" is greater than "v2", "3" is greater than "2" etc.)
- If a `<version>` element is supplied in the request, then the specified version `<name>` and `<value>` are searched for. If the version is found in the SEC, then it is used. If the version is not found, then the request will fail

Prerequisites

A subscriber/pool with the key(s) of the `keyNameX/keyValueX` values supplied must exist.

The supplied `dataName` must be a valid interface entity name for a subscriber/pool.

Any supplied `dataXpath` must reference a valid field set within the entity/row for the subscriber/pool.

Any supplied `<version>` `versionName/versionValue` must be a valid transparent entity version defined in the SEC for the specified entity.

For subscriber based requests, all supplied keys must reference the same subscriber.

Request

```
<updateFieldSet [create="create"]
    [createEntityIfNotExist="createEntityIfNotExist"]>

  <key>

    <keyName1>keyValue1</keyName1>
  [
    <keyName2>keyValue2</keyName2>
    :
    <keyNameN>keyValueN</keyNameN>
  ]

</key>

<entity>

  <data>
    <name>dataName</name>
    <interface>dataInterface</interface>
    <xpath>dataXpath</xpath>
  [
    <version>
      <name>versionName</name>
      <value>versionValue</value>
```

```

    </version>
]
  </data>

  <content>
    entityContent
  </content>

</entity>
</updateFieldSet>

```

- **create:** (Optional) Indicates whether the row should be created if it does not already exists
 - Value is either `true` or `false`
 - **Note:** If the entity does not exist, and the value of `createEntityIfNotExist` is set to `true`, the value of `create` is ignored and the row will be created in the new entity
- **createEntityIfNotExist:** (Optional) Indicates whether the entity should be created if it does not already exist before creating the entity/row (for example if a Quota row is being created, and the Quota entity does not currently exist for the subscriber)
 - Value is either `true` or `false`
- **keyNameX:** A key field within the subscriber Profile or pool Profile
 - Value is either `IMSI`, `MSISDN`, `NAI`, `Account Id`, or `PoolID`
- **keyValueX:** Corresponding key field value assigned to `keyNameX`
- **dataName:** A user defined entity type/name for the transparent entity being updated
 - Value is either `Subscriber`, `Quota`, `State`, `DynamicQuota`, `Pool`, `PoolQuota`, `PoolState`, or `PoolDynamicQuota`
- **dataInterface:** The interface type used to identify the bulk import/export interface
 - Value is `XMLIMPORT`
- **dataXPath:** XML XPath expression identifying the field set to be updated
 - **Note:** To update the entire entity (i.e. a complete opaque data replacement) the `dataXPath` value should be empty
 - Value is `"/usage/"` for a Quota or PoolQuota row
 - Value is `"/definition/"` for a DynamicQuota or PoolDynamicQuota row
 - Value is `"/state/"` for a State or PoolState property
- **versionName:** (Optional) The name of the versioning element for the entity, used to specify the default version number when creating an entity
- **versionValue:** (Optional) The version value for the entity, used to specify the default version number when creating an entity
- **entityContent:** Content of entity/row being updated

Note: For subscriber based requests, multiple subscriber key values can be supplied. See section 2.4 for details.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of `errorValue` in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
InterfaceEntityNameNotFound	Interface Entity Not Found
ElementNotDefined	An XML Element is not defined
FieldValueNotValid	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OccurrenceConstraintViolation	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
InvalidInputXml	Invalid Input XML.
FieldDefinitionNotFound	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
MultipleRowsFound	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
KeyNotFound	Key Not Found. A subscriber/pool with the given key cannot be found
NonEmptyXPathForOpaqueData	XPath cannot be non-empty for an Opaque-data operation
RegisterDataNotFound	Register Data Not Found
OperationNotAllowed	Operation Not Allowed
MultipleKeysNotMatch	All supplied keys do not correspond to the same subscriber

Examples

Request #1

A request is made to update the entire Quota entity. The subscriber currently has a Quota entity.

```
<updateFieldSet createEntityIfNotExist="false" create="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath/>
    </data>
    <content>
<![CDATA[
<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>
]]>
    </content>
```



```

    </entity>
</updateFieldSet>

```

Response #1

The request is successful, and the Quota entity was updated.

Request #2

A request is made to update the entire State entity. The subscriber currently does NOT have a State entity. The request indicates that the entity should NOT be created if it does not exist.

```

<updateFieldSet createEntityIfNotExist="false" create="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>State</name>
      <interface>XMLIMPORT</interface>
      <xpath/>
    </data>
    <content>
<![CDATA[
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2014-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>no</value>
  </property>
</state>
]]>
    </content>
  </entity>
</updateFieldSet>

```

Response #2

The request fails. The *errorValue* indicates the opaque State entity does not exist.

```
[error 47 errorText : line lineNumber]
```

Request #3

A request is made to update the *Q1* row in the Quota entity. The subscriber currently has a Quota entity, but the *Q1* row does not exist. The request indicates that the row should NOT be created if it does not exist.

```

<updateFieldSet createEntityIfNotExist="false" create="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>

```

```

    <interface>XMLIMPORT</interface>
    <xpath>/usage</xpath>
  </data>
  <content>
<![CDATA[
<quota name="Q1">
  <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>
]]>
  </content>
</entity>
</updateFieldSet>

```

Response #3

The request fails. The *errorValue* indicates the row does not exist.

```
[error 23 errorText : line lineNumber]
```

Request #4

A request is made to update the *Q1* row in the Quota entity. The subscriber currently has a Quota entity, but the Q1 row does not exist. The request indicates that the row should be created if it does not exist.

```

<updateFieldSet createEntityIfNotExist="false" create="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage</xpath>
    </data>
    <content>
<![CDATA[
<quota name="Q1">
  <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>
]]>
    </content>
  </entity>
</updateFieldSet>

```

Response #4

The request is successful, and the Quota row was created.

Request #5

A request is made to update the *Q1* row in the Quota entity. The subscriber currently does NOT have a Quota entity. The request indicates that the entity should be created if it does not exist. No version number is specified, so the latest version of the Quota entity is used to create Quota.

```
<updateFieldSet createEntityIfNotExist="true" create="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage</xpath>
    </data>
    <content>
<![CDATA[
<quota name="Q1">
  <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>
]]>
    </content>
  </entity>
</updateFieldSet>
```

Response #5

The request is successful, and the Quota row was created.

Request #6

A request is made to update the *Q5* row in the Quota entity. The subscriber currently does NOT have a Quota entity. The request indicates that the entity should be created if it does not exist. The request specifies that the *version 3* of the Quota entity is used to create Quota.

```
<updateFieldSet createEntityIfNotExist="true" create="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage</xpath>
      <version>
        <name>version</name>
        <value>3</value>
      </version>
    </data>
    <content>
<![CDATA[
<quota name="Q5">
  <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>
]]>
  </content>
</entity>
</updateFieldSet>

```

Response #6

The request is successful, and the Quota row was created.

Request #7

A request is made to update the *Q7* row in the Quota entity. The subscriber currently does NOT have a Quota entity. The request indicates that the entity should be created if it does not exist. The request specifies that the “*version 4*” of the Quota entity is used to create Quota. The “*version 4*” of Quota does NOT exist.

```

<updateFieldSet createEntityIfNotExist="true" create="false">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage</xpath>
      <version>
        <name>version</name>
        <value>4</value>
      </version>
    </data>
    <content>
<![CDATA[
<quota name="Q7">
  <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>
]]>
    </content>
  </entity>
</updateFieldSet>

```

Response #7

The request fails. The *errorValue* indicates that the “*version 4*” does not exist.

```
[error 22 errorText : line lineNumber]
```

Request #8

A request is made to update the *PDQ1* row in the PoolDynamicQuota entity. The pool currently has a PoolDynamicQuota entity, but the PDQ1 row does not exist. The request indicates that the row should be created if it does not exist.

```

<updateFieldSet createEntityIfNotExist="false" create="true">
  <key>

```

```

    <PoolID>10000</PoolID>
  </key>
  <entity>
    <data>
      <name>PoolDynamicQuota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/definition</xpath>
    </data>
    <content>
<![CDATA[
<DynamicQuota name="PDQ1">
  <Type>Roll-over</Type>
  <InstanceId>15678</InstanceId>
  <Priority>4</Priority>
  <InitialTime>135</InitialTime>
  <InitialTotalVolume>2000</InitialTotalVolume>
  <InitialInputVolume>1500</InitialInputVolume>
  <InitialOutputVolume>500</InitialOutputVolume>
  <InitialServiceSpecific>4</InitialServiceSpecific>
  <activationdatetime>2015-03-09T11:20:32</activationdatetime>
  <expirationdatetime>2015-04-09T11:20:32</expirationdatetime>
  <InterimReportingInterval>100</InterimReportingInterval>
  <Duration>10</Duration>
</DynamicQuota>
]]>
    </content>
  </entity>
</updateFieldSet>

```

Response #8

The request is successful, and the PoolDynamicQuota row was created.

Request #9

A request is made to update the *mcc* property in the State entity. The subscriber currently has a State entity with the *mcc* property. The request indicates that the row should be updated if it already exists.

```

<updateFieldSet createEntityIfNotExist="false" create="true">
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>State</name>
      <interface>XMLIMPORT</interface>
      <xpath>/state</xpath>
    </data>
    <content>
<![CDATA[
<property>
  <name>mcc</name>
  <value>315</value>
</property>
]]>
    </content>
  </entity>
</updateFieldSet>

```

Response #9

The request is successful, and the State property `mcc` was updated.

7.1.4 Delete Field

Description

This operation deletes the specified field(s) for the subscriber/pool identified by the specified key(s) in the request, in the specified transparent entity.

A field with a specific value can be deleted the value matches what is supplied in *fieldValueX*.

If the field is multi-value field then all values are deleted, unless specific values are supplied in *fieldValueX*, when only the matching field values are deleted.

Deletion of a complete field results removal of the entire field from the entity. I.e. the field is not present, not just the value is empty.

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

Note: If a field value is supplied for a field, and the supplied value does not match the existing value, the request will still succeed.

Note: If a field is deleted that has a default value defined in the SEC, then the field will be set to the default instead of being deleted.

Note: If one or more key (i.e. IMSI/MSISDN/NAI/AccountId) field (or values) are deleted for a subscriber, then afterwards, the subscriber must still have at least one key type/value remaining or the request will fail.

Prerequisites

A subscriber/pool with the key(s) of the *keyNameX/keyValueX* values supplied must exist.

The supplied *dataName* must be a valid interface entity name for a subscriber/pool.

The supplied *dataXPath* must reference a valid XML XPath where the specified fields in `<fields>` exist within the transparent entity for the subscriber/pool.

Each requested field *fieldNameX* must be a valid field in the specified transparent entity.

For subscriber based requests, all supplied keys must reference the same subscriber.

Request

```
<deleteField>
```

```
  <key>
```

```
    <keyName1>keyValue1</keyName1>
```

```
  [
```

```
    <keyName2>keyValue2</keyName2>
```

```
    :
```

```
    <keyNameN>keyValueN</keyNameN>
```

```
  ]
```

```
</key>
```

```
<entity>
```

```
  <data>
```

```
    <name>dataName</name>
```

```
    <interface>dataInterface</interface>
```

```
    <xpath>dataXPath</xpath>
```

```

</data>

<fields>

  <field name="fieldName1">[fieldValue1]</field>
[
  <field name="fieldName2">[fieldValue2]</field>
  :
  <field name="fieldName250">[fieldValue250]</field>
]
</fields>

</entity>

</deleteField>

```

- **keyNameX**: A key field within the subscriber Profile or pool Profile
 - Value is either IMSI, MSISDN, NAI, AccountId, or PoolID
- **keyValueX**: Corresponding key field value assigned to *keyNameX*
- **dataName**: A user defined entity type/name for the transparent entity being updated
 - Value is either Subscriber, Quota, State, DynamicQuota, Pool, PoolQuota, PoolState, or PoolDynamicQuota
- **dataInterface**: The interface type used to identify the bulk import/export interface
 - Value is XMLIMPORT
- **dataXPath**: XML XPath value which corresponds to the root element, or a row element in which the fields being deleted are contained
 - Value is `"/usage/quota[@name='quotaName']"` for a Quota or PoolQuota row without an instance specified
 - Value is `"/usage/quota[@name='quotaName' and cid='quotaCid']"` for a Quota or PoolQuota row with an instance specified
 - Value is `"/usage/quota[@name='quotaName' and Type='quotaType']"` for a Quota or PoolQuota row with a type specified
 - Value is `"/definition/DynamicQuota[@name='dynamicQuotaName']"` for a DynamicQuota or PoolDynamicQuota row with the name specified
 - Value is `"/definition/DynamicQuota[@name='dynamicQuotaName' and InstanceId='dynamicQuotaInstanceId']"` for a DynamicQuota or PoolDynamicQuota row with an instance specified
 - Value is `"/definition/DynamicQuota[@name='dynamicQuotaName' and Type='dynamicQuotaType']"` for a DynamicQuota or PoolDynamicQuota row with a type specified
 - Value is `"/state/property[name='propertyName']"` for a State or PoolState with the name specified
- **fieldNameX**: A user defined field within the transparent entity being updated
- **fieldValueX**: (Optional) Corresponding field value assigned to *fieldNameX*. Used when deleting a field only if set to the supplied field value. If no field value is supplied, the supplied field is deleted regardless of current value
 - **Note**: for multi-value fields, individual *fieldNameX* elements must be specified for each instance/value being deleted
- **quotaName**: (See *dataXPath*) The name that identifies the required quota row within the Quota/PoolQuota entity
- **quotaCid**: (See *dataXPath*) The instance value that identifies the specific required quota row within the Quota/PoolQuota entity

- **quotaType:** (See *dataXpath*) The type value that identifies the specific required quota row within the Quota/PoolQuota entity
- **dynamicQuotaName:** (See *dataXpath*) The name that identifies the required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **dynamicQuotaInstanceId:** (See *dataXpath*) The instance value that identifies the specific required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **dynamicQuotaType:** (See *dataXpath*) The type that identifies the required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **propertyName:** (See *dataXpath*) The name that identifies the required state property within the State/PoolState entity

Note: A maximum of 250 fields can be deleted in a single <deleteField> request.

Note: For subscriber based requests, multiple subscriber key values can be supplied. See section 2.4 for details.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
InterfaceEntityNameNotFound	Interface Entity Not Found
ElementNotDefined	An XML Element is not defined
FieldNotMultiValued	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
FieldDefinitionNotFound	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
MultipleRowsFound	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
KeyNotFound	Key Not Found. A subscriber/pool with the given key cannot be found
RegisterDataNotFound	Register Data Not Found
OperationNotAllowed	Operation Not Allowed
KeyValueInvalid	The key value supplied is invalid, due to invalid characters/format etc.
OneKeyRequired	A subscriber must have at least one key value
MultipleKeysNotMatch	All supplied keys do not correspond to the same subscriber
EnterpriseToBasicPoolFailed	Enterprise to Basic Pool Conversion Failed Threshold Exceeded. An enterprise pool cannot be converted to a basic pool by deleting the "Type" field, if the pool has greater than the allowed number of members for a basic pool.

Examples

Request #1

A request is made to delete the *Tier* and *Custom1* fields. Both fields are valid subscriber Profile fields.


```

<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="Tier" />
      <field name="Custom1" />
    </fields>
  </entity>
</deleteField>

```

Response #1

The request is successful, and the two fields were deleted.

Request #2

A request is made to delete the *message* field. The field *message* is not a valid subscriber Profile field.

```

<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="message" />
    </fields>
  </entity>
</deleteField>

```

Response #2

The request fails. The *errorValue* indicates the *message* field was invalid.

```
[error 30 errorText : line lineNumber]
```

Request #3

A request is made to delete the *EveningPass* value from the multi-value field *Entitlement* retaining all other values. The current value of the field is “*DayPass,Weekend,EveningPass*”.

```

<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>

```

```

    <fields>
      <field name="Entitlement">EveningPass</field>
    </fields>
  </entity>
</deleteField>

```

Response #3

The request is successful, and the *Entitlement* field was updated. The value of the field is now “*DayPass,Weekend*”.

Request #4

A request is made to delete the *inputVolume* and *outputVolume* fields from the *Q1* Quota row.

```

<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1']</xpath>
    </data>
    <fields>
      <field name="inputVolume"/>
      <field name="outputVolume"/>
    </fields>
  </entity>
</deleteField>

```

Response #4

The request is successful, and the two fields were deleted.

Request #5

A request is made to delete the *totalVolume* field with a value of *500* from the *Q1* Quota row. The value of *outputVolume* is currently *500*.

```

<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1']</xpath>
    </data>
    <fields>
      <field name="totalVolume">500</field>
    </fields>
  </entity>
</deleteField>

```

Response #5

The request is successful, and the field is deleted.

Request #6

A request is made to delete the *totalVolume* field with a value of *500* from the *Q1* Quota row. The value of *outputVolume* is currently *600* (i.e. it does not match the request).

```
<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1']</xpath>
    </data>
    <fields>
      <field name="totalVolume">500</field>
    </fields>
  </entity>
</deleteField>
```

Response #6

The request is successful, but the field is NOT deleted and still contains the value *600*.

Request #7

A request is made to delete the *Custom3* field. Two key values are supplied, but these keys correspond to two different subscribers.

```
<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
    <IMSI>302370111111111</IMSI>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="Custom3"/>
    </fields>
  </entity>
</deleteField>
```

Response #7

The request fails. The *errorValue* indicates that the subscriber with the two supplied keys is not found.

[error 39 *errorText* : line *lineNumber*]

Request #8

A request is made to delete the *MSISDN* field with a value of “14161112222” for the subscriber. The subscriber currently has 3 *MSISDN* values, “15141234567”, “14161112222”, and “15145556666”.

```

<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="MSISDN">14161112222</field>
    </fields>
  </entity>
</deleteField>

```

Response #8

The request is successful, and the MSISDN value “14161112222” is deleted. The subscriber now has 2 MSISDNs, “15141234567” and “15145556666”.

Request #9

A request is made to delete the *MSISDN* field for the subscriber. The subscriber currently has 2 MSISDN values, “15141234567” and “15145556666”. The subscriber also has an IMSI value.

```

<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="MSISDN" />
    </fields>
  </entity>
</deleteField>

```

Response #9

The request is successful, and the MSISDN field is deleted. The subscriber now does not have any MSISDN values.

Request #10

A request is made to delete the *MSISDN* field for the subscriber. The subscriber currently has only a single key, an MSISDN value “15141234567”.

```

<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
  </entity>
</deleteField>

```

```

    </data>
    <fields>
      <field name="MSISDN" />
    </fields>
  </entity>
</deleteField>

```

Response #10

The request fails. The *errorValue* indicates that the subscriber must have at least one key value.

```
[error 62 errorText : line lineNumber]
```

Request #11

A request is made to delete the *Custom5* field for a subscriber. Multiple key lookup is enabled. The IMSI and MSISDN values provided both exist, but correspond to different subscribers.

```

<deleteField>
  <key>
    <IMSI>3023701111111111</IMSI>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
    <fields>
      <field name="Custom5" />
    </fields>
  </entity>
</deleteField>

```

Response #11

The request fails. The *errorValue* indicates that all supplied keys do not correspond to the same subscriber.

```
[error 63 errorText : line lineNumber]
```

Request #12

A request is made to delete the *InitialTotalVolume* field with a value of 500 from the *PDQ1* PoolDynamicQuota row. The value of *InitialTotalVolume* is currently 500.

```

<deleteField>
  <key>
    <PoolID>10000</PoolID>
  </key>
  <entity>
    <data>
      <name>PoolDynamicQuota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/definition/DynamicQuota[@name='PDQ1']</xpath>
    </data>
    <fields>
      <field name="InitialTotalVolume">500</field>
    </fields>
  </entity>
</deleteField>

```

Response #12

The request is successful, and the field is deleted.

Request #13

A request is made to delete the *totalVolume* field with a value of 500 from the *Q1* Quota row having *cid* of value 45678. The value of *totalVolume* is currently 500.

```
<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1' and cid='45678']</xpath>
    </data>
    <fields>
      <field name="totalVolume">500</field>
    </fields>
  </entity>
</deleteField>
```

Response #13

The request is successful, and the field is deleted.

Request #14

A request is made to delete the *mcc* property with a value of 315 from the State entity. The value of *mcc* is currently 315.

```
<deleteField>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>State</name>
      <interface>XMLIMPORT</interface>
      <xpath>/state/property[name='mcc']</xpath>
    </data>
    <fields>
      <field name="value">315</field>
    </fields>
  </entity>
</deleteField>
```

Response #14

The request is successful, and the field is deleted.

7.1.5 Delete FieldSet**Description**

This operation deletes an entity, or row within an entity for the subscriber/pool identified by the specified key(s) in the request.

Note: If an entity or the row within the entity is being deleted, and it does not exist, the request will fail.

Prerequisites

A subscriber/pool with the key(s) of the *keyNameX/keyValueX* values supplied must exist.

The supplied *dataName* must be a valid interface entity name for a subscriber/pool.

The supplied *dataXpath* must reference a valid field set within the entity for the subscriber/pool.

For subscriber based requests, all supplied keys must reference the same subscriber.

Request

<deleteFieldSet>

```

<key>
  <keyName1>keyValue1</keyName1>
[
  <keyName2>keyValue2</keyName2>
  :
  <keyNameN>keyValueN</keyNameN>
]
</key>
<entity>
  <data>
    <name>dataName</name>
    <interface>dataInterface</interface>
    <xpath>dataXpath</xpath>
  </data>
</entity>
</deleteFieldSet>

```

- **keyNameX:** A key field within the subscriber Profile or pool Profile
 - Value is either IMSI, MSISDN, NAI, AccountId, or PoolID
- **keyValueX:** Corresponding key field value assigned to *keyNameX*
- **dataName:** A user defined entity type/name for the transparent entity being updated
 - Value is either Subscriber, Quota, State, DynamicQuota, Pool, PoolQuota, PoolState, or PoolDynamicQuota
- **dataInterface:** The interface type used to identify the bulk import/export interface
 - Value is XMLIMPORT
- **dataXpath:** XML XPath value which corresponds to the root element, or row being deleted
 - Value is `"/usage/quota[@name='quotaName']"` for a Quota or PoolQuota row without an instance specified
 - Value is `"/usage/quota[@name='quotaName' and cid='quotaCid']"` for a Quota or PoolQuota row with an instance specified
 - Value is `"/usage/quota[@name='quotaName' and Type='quotaType']"` for a Quota or PoolQuota row with a type specified
 - Value is `"/definition/DynamicQuota[@name='dynamicQuotaName']"` for a DynamicQuota or PoolDynamicQuota row with the name specified

- Value is `"/definition/DynamicQuota[@name='dynamicQuotaName' and InstanceId='dynamicQuotaInstanceId']"` for a DynamicQuota or PoolDynamicQuota row with an instance specified
- Value is `"/definition/DynamicQuota[@name='dynamicQuotaName' and Type='dynamicQuotaType']"` for a DynamicQuota or PoolDynamicQuota row with a type specified
- Value is `"/state/property[name='propertyName']"` for a State or PoolState row with the property name specified
- **quotaName:** (See *dataXpath*) The name that identifies the required quota row within the Quota/PoolQuota entity
- **quotaCid:** (See *dataXpath*) The instance value that identifies the specific required quota row within the Quota/PoolQuota entity
- **quotaType:** (See *dataXpath*) The type value that identifies the specific required quota row within the Quota/PoolQuota entity
- **dynamicQuotaName:** (See *dataXpath*) The name that identifies the required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **dynamicQuotaInstanceId:** (See *dataXpath*) The instance value that identifies the specific required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **dynamicQuotaType:** (See *dataXpath*) The type that identifies the required dynamic quota row within the DynamicQuota/PoolDynamicQuota entity
- **propertyName:** (See *dataXpath*) The name that identifies the required state property within the State/PoolState entity

Note: For subscriber based requests, multiple subscriber key values can be supplied. See section 2.4 for details.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
InterfaceEntityNameNotFound	Interface Entity Not Found
OccurrenceConstraintViolation	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
KeyNotFound	Key Not Found. A subscriber/pool with the given key cannot be found
NonEmptyXPathForOpaqueData	XPath cannot be non-empty for an Opaque-data operation
RegisterDataNotFound	Register Data Not Found
OperationNotAllowed	Operation Not Allowed
KeyValueInvalid	The key value supplied is invalid, due to invalid characters/format etc.
MultipleKeysNotMatch	All supplied keys do not correspond to the same subscriber
FieldSetNotFound	Field Set Not Found

Examples

Request #1

A request is made to delete the Quota entity for a subscriber. The subscriber currently has a Quota entity.

```
<deleteFieldSet>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath/>
    </data>
  </entity>
</deleteFieldSet>
```

Response #1

The request is successful, and the Quota entity was deleted for the subscriber.

Request #2

A request is made to delete the State entity for a subscriber. The subscriber currently does NOT have a State entity.

```
<deleteFieldSet>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>State</name>
      <interface>XMLIMPORT</interface>
      <xpath/>
    </data>
  </entity>
</deleteFieldSet>
```

Response #2

The request fails. The *errorValue* indicates the State entity does not exist.

```
[error 47 errorText : line lineNumber]
```

Request #3

A request is made to delete the *Q1* row within the Quota entity for a subscriber. The subscriber currently has a Quota entity with a row called *Q1*.

```
<deleteFieldSet>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1']</xpath>
    </data>
```

```

</entity>
</deleteFieldSet>

```

Response #3

The request is successful, and the *Q1* row within the Quota entity was deleted for the subscriber.

Request #4

A request is made to delete the *Q2* row within the Quota entity for a subscriber. The subscriber currently has a Quota entity, but it does not contain a row called *Q2*.

```

<deleteFieldSet>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q2']</xpath>
    </data>
  </entity>
</deleteFieldSet>

```

Response #4

The request fails. The *errorValue* indicates the Quota row *Q2* does not exist.

```
[error 23 errorText : line lineNumber]
```

Request #5

A request is made to delete the *PDQ1* row within the PoolDynamicQuota entity for a pool. The pool currently has a PoolDynamicQuota entity with a row called *PDQ1*.

```

<deleteFieldSet>
  <key>
    <PoolID>10000</PoolID>
  </key>
  <entity>
    <data>
      <name>PoolDynamicQuota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/definition/DynamicQuota[@name='PDQ1']</xpath>
    </data>
  </entity>
</deleteFieldSet>

```

Response #5

The request is successful, and the *PDQ1* row within the PoolDynamicQuota entity was deleted for the pool.

Request #6

A request is made to delete the *mcv* property within the State entity for a subscriber. The subscriber currently has a State entity with a property called *mcc*.

```
<deleteFieldSet>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>State</name>
      <interface>XMLIMPORT</interface>
      <xpath>/state/property[name='mcc']</xpath>
    </data>
  </entity>
</deleteFieldSet>
```

Response #6

The request is successful, and the *mcc* row within the State entity was deleted for the subscriber.

Request #7

A request is made to delete the *Q1* row having *cid* with value *23456* within the Quota entity for a subscriber. The subscriber currently has a Quota entity with a row called *Q1* having *cid* *23456*.

```
<deleteFieldSet>
  <key>
    <MSISDN>15141234567</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1' and cid='23456']</xpath>
    </data>
  </entity>
</deleteFieldSet>
```

Response #7

The request is successful, and the *Q1* row having *cid* *23456* within the Quota entity was deleted for the subscriber.

8 Special Operations

Table 17: Summary of Special Operation Commands

Command	Description	Key(s)	Command Syntax
Reset	Reset fields within an Entity/Row	(MSISDN, IMSI, NAI, AccountId or PoolID)	<reset>

8.1.1 Reset

Description

This operation resets the field values in an entity (or specified row within an entity) for a subscriber/pool. The values are reset to the values defined in the SEC.

Prerequisites

A subscriber/pool with the key(s) of the *keyNameX/keyValueX* values supplied must exist.

The supplied *dataName* must be a valid interface entity name for a subscriber/pool.

The supplied *dataXPath* must reference a valid field entity/row for the subscriber/pool.

For subscriber based requests, all supplied keys must reference the same subscriber.

Note: When an entity/row instance is reset using the “Reset” command, each resettable field is set to its defined reset value. If the field does not currently exist, it is *not* created. But, if a resettable field does not exist, and the field has a default value, then the field will then get created with the default value.

Request

```
<reset>
```

```
  <key>
```

```
    <keyName1>keyValue1</keyName1>
```

```
  [
```

```
    <keyName2>keyValue2</keyName2>
```

```
    :
```

```
    <keyNameN>keyValueN</keyNameN>
```

```
  ]
```

```
</key>
```

```
<entity>
```

```
  <data>
```

```
    <name>dataName</name>
```

```
    <interface>dataInterface</interface>
```

```
    <xpath>dataXPath</xpath>
```

```
  </data>
```

```
</entity>
```

```
</reset>
```

- **keyNameX:** A key field within the subscriber Profile or pool Profile
 - Value is either IMSI, MSISDN, NAI, AccountId, or PoolID
- **keyValueX:** Corresponding key field value assigned to *keyNameX*
- **dataName:** A user defined entity type/name for the transparent entity being updated

- Value is either Quota or PoolQuota
- **dataInterface:** The interface type used to identify the bulk import/export interface
 - Value is XMLIMPORT
- **dataXPath:** XML XPath value which corresponds to the row element for which the reset operation needs to be performed
 - Value is `"/usage/quota[@name='quotaName']"` for a Quota or PoolQuota row without an instance specified
 - Value is `"/usage/quota[@name='quotaName' and cid='quotaCid']"` for a Quota or PoolQuota row with an instance specified
- **quotaName:** (See *dataXPath*) The name that identifies the required quota row within the Quota/PoolQuota entity
- **quotaCid:** (See *dataXPath*) The instance value that identifies the specific required quota row within the Quota/PoolQuota entity

Note: For subscriber based requests, multiple subscriber key values can be supplied. See section 2.4 for details.

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
InterfaceEntityNameNotFound	Interface Entity Not Found
FieldSetNotFound	Field Set Not Found
FieldSetDefinitionNotFound	Field Set Not Defined
EntityDefinitionNoReset	Entity Cannot be Reset. The reset command cannot be used on the requested entity
MultipleRowsFound	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
KeyNotFound	Key Not Found. A subscriber/pool with the given key cannot be found
RegisterDataNotFound	Register Data Not Found
OperationNotAllowed	Operation Not Allowed
MultipleKeysNotMatch	All supplied keys do not correspond 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*.

```
<reset>
  <key>
    <MSISDN>33123654862</MSISDN>
  </key>
</entity>
```

```

<data>
  <name>Quota</name>
  <interface>XMLIMPORT</interface>
  <xpath>/usage/quota[@name='Q1']</xpath>
</data>
</entity>
</reset>

```

Response #1

The request is successful, and the specified Quota row was reset.

Request #2

A request is made to reset the *Q1* Quota row. The subscriber does not have Quota data. Two subscriber addresses are supplied, and multiple key lookup is enabled.

```

<reset>
  <key>
    <MSISDN>15141234567</MSISDN>
    <IMSI>302370123456789</IMSI>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q1']</xpath>
    </data>
  </entity>
</reset>

```

Response #2

The request fails. The subscriber with both keys exists, but the *errorValue* indicates the subscriber does not have Quota data.

[error 47 errorText : line lineNumber]

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*.

```

<reset>
  <key>
    <MSISDN>33123654862</MSISDN>
  </key>
  <entity>
    <data>
      <name>Quota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='Q6']</xpath>
    </data>
  </entity>
</reset>

```

Response #3

The request fails. The *errorValue* indicates the *Q6* data row was not present.

[error 29 errorText : line lineNumber]

Request #4

A request is made to reset the *PQ1* PoolQuota row with a *cid* of 77 for a pool. The pool has PoolQuota data, and the PoolQuota data contains a row called *PQ1* with a *cid* of 77.

```
<reset>
  <key>
    <PoolID>100000</PoolID>
  </key>
  <entity>
    <data>
      <name>PoolQuota</name>
      <interface>XMLIMPORT</interface>
      <xpath>/usage/quota[@name='PQ1' and cid='77']</xpath>
    </data>
  </entity>
</reset>
```

Response #4

The request is successful, and the specified PoolQuota row was reset.

9 Restore Commands

The restore commands are used in the cases when it is necessary to restore an entire subscriber or pool.

Note: These commands should only be used to aid in the recovery of subscribers/pools, and are not to be used in the creation of new subscribers/pool.

Table 18: Summary of Restore Commands

Command	Description	Key(s)	Command Syntax
Restore Subscriber	Restore a Subscriber	MSISDN, IMSI, NAI, or AccountId	<restoreSubscriber>
Restore Pool	Restore a Pool	PoolID	<restorePool>

9.1.1 Restore Subscriber

Description

This operation restores a complete subscriber, including all defined entities in a single request.

Note: The request will always contain an <entity> element with a Profile entity.

Note: The <restoreSubscriber> request cannot be used within a <transaction>.

Prerequisites

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

Request

<restoreSubscriber>

<subscriberData>

< poolId>poolId</poolId> | <poolId/> >

<ae>ae</ae>

<subscriberLastUpdateTime>subscriberLUT</subscriberLastUpdateTime>

<notificationSubscriptionLastUpdateTime>nsLUT</notificationSubscriptionLastUpdateTime>

<entity>

<name>Profile</name>

<seqnum>entitySeqnum1</seqnum>

<data> <![CDATA[

entityData1

]]></data>

</entity>

[

<entity>

<name>entityName2</name>

<seqnum>entitySeqnum2</seqnum>

<data> <![CDATA[

entityData2

]]></data>

</entity>

:


```

<entity>
  <name>entityNameN</name>
  <seqnum>entitySeqnumN</seqnum>
  <data> <![CDATA[
entityDataN
]]></data>
</entity>
]
[
<notificationSubscription>
  <client>client1</client>
  <userIdentity>userIdentity1</userIdentity>
  <creationTime>creationTime1</creationTime>
  <expirationTime>expirationTime1</expirationTime>
  <notEff>notEff1</notEff>
</notificationSubscription>
:
<notificationSubscription>
  <client>clientN</client>
  <userIdentity>userIdentityN</userIdentity>
  <creationTime>creationTimeN</creationTime>
  <expirationTime>expirationTimeN</expirationTime>
  <notEff>notEffN</notEff>
</notificationSubscription>
]
</subscriberData>
</restoreSubscriber>

```

- **poolId**: PoolID value of the pool the subscriber is a member of. Numeric value, 1-22 digits in length
 - Values: 1-9999999999999999999999
 - **Note**: If the subscriber is not a member of a pool, then no PoolID value is given
- **ae**: Flag indicating if the subscriber is considered as auto-enrolled or not
 - Value is either `true` or `false`
- **subscriberLUT**: The date/time when the data for the subscriber was last updated. Numeric value indicating number of seconds since midnight on January 1st 1970, in UTC time
 - Values: 0- 4294967295
- **nsLUT**: The most recent date/time when a notification subscription for the subscriber was either added or removed. Numeric value indicating number of seconds since midnight on January 1st 1970, in UTC time
 - Values: 0- 4294967295
- **entityNameX**: A user defined entity type/name for the subscriber entity
 - Value is either `Profile`, `Quota`, `State`, or `DynamicQuota`
- **entitySeqnumX**: The sequence number associated with the subscriber entity (as used on the Diameter Sh interface)
 - Values: 0- 65535
- **entityDataX**: Contents of the subscriber XML data “blob”
- **clientX**: Subscribing client name of the subscribing entity. For example, a Diameter FQDN as used on the Diameter Sh interface

- ***userIdentityX***: The identity value supplied by the subscribing client to identify the subscriber (such as an IMSI, MSISDN, NAI etc.) in network specific format
- ***creationTimeX***: The date/time when the notification subscription was created. Numeric value indicating number of seconds since midnight on January 1st 1970, in UTC time
 - Values: 0– 4294967295
- ***expirationTimeX***: The date/time when the notification subscription will expire. Numeric value indicating number of seconds since midnight on January 1st 1970, in UTC time
 - Values: 0– 4294967295
- ***notEffX***: Flag indicating if the client allows multiple entities to be included in a single notification (for example when sending a PNR request on the Diameter Sh interface)
 - Value is either `true` or `false`

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
InvalidXml	Invalid XML.
KeyNotFound	Key Not Found. A subscriber/pool with the given key cannot be found
KeyValueInvalid	The key value supplied is invalid, due to invalid characters/format etc.
EntityDefinitionNotFound	Entity Definition Not Found
InvalidInputXml	Invalid Input XML.
HwtRetryLimitExceeded	Data could not be committed to database as the total number of retries to commit database transactions exhausted.
Inconsistent	Database is inconsistent
KeyAlreadyExists	Key Already Exists. A subscriber/pool already exists with the given key

Examples

Request #1

A request is made to restore a subscriber, who has Quota data. The subscriber is in a pool which has a *poolId* of 100000. The subscriber has no active notification subscriptions.

```
<restoreSubscriber>
  <subscriberData>
    <poolId>100000</poolId>
    <ae>false</ae>
    <subscriberLastUpdateTime>1436798453</subscriberLastUpdateTime>
    <notificationSubscriptionLastUpdateTime>1436798453</notificationSubscriptionLastUpdateTi
me>
    <entity>
      <name>Profile</name>
      <seqnum>5</seqnum>
      <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><subscriber><field name="AccountId
">10404723525</field><field name="MSISDN">15141234567</field><field name="IMSI">3023701234567
89</field><field name="BillingDay">1</field><field name="Tier">Gold</field><field name="Entit
lement">DayPass</field><field name="Entitlement">DayPassPlus</field></subscriber>]]></data>
```

```

</entity>
<entity>
  <name>Quota</name>
  <seqnum>12</seqnum>
  <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><usage><version>3</version><quota
name="AggregateLimit"><cid>9223372036854775807</cid><time>3422</time><totalVolume>1000</total
Volume><inputVolume>980</inputVolume><outputVolume>20</outputVolume><serviceSpecific>12</serv
iceSpecific><nextResetTime>2015-07-22T00:00:00-05:00</nextResetTime></quota></usage>]]></data
>
  </entity>
</subscriberData>
</restoreSubscriber>

```

Response #1

The request is successful, and the subscriber was restored.

Request #2

A request is made to restore an auto-enrolled subscriber. The subscriber has two active notification subscriptions.

```

<restoreSubscriber>
  <subscriberData>
    <poolId/>
    <ae>true</ae>
    <subscriberLastUpdateTime>1436798453</subscriberLastUpdateTime>
    <notificationSubscriptionLastUpdateTime>1436817543</notificationSubscriptionLastUpdateTi
me>
    <entity>
      <name>Profile</name>
      <seqnum>0</seqnum>
      <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><subscriber><field name="MSISDN">
15145551234</field><field name="BillingDay">0</field></subscriber>]]></data>
    </entity>
    <notificationSubscription>
      <client>mpe6.policy.operator.com</client>
      <userIdentity>tel:15145551234</userIdentity>
      <creationTime>1436798453</creationTime>
      <expirationTime>0</expirationTime>
      <notEff>true</notEff>
    </notificationSubscription>
    <notificationSubscription>
      <client>mpe3.policy.operator.com</client>
      <userIdentity>tel:15145551234</userIdentity>
      <creationTime>1436796002</creationTime>
      <expirationTime>0</expirationTime>
      <notEff>true</notEff>
    </notificationSubscription>
  </subscriberData>
</restoreSubscriber>

```

Response #2

The request is successful, and the subscriber was restored.

9.1.2 Restore Pool**Description**

This operation restores a complete pool, including all defined entities for the pool, and all subscribers within the pool, in a single request.

Note: The request will always contain an `<entity>` element within the `<poolData>` with a `PoolProfile` entity.

Note: A pool can contain zero, one, or more subscribers in the `<subscriberData>` elements.

Note: The `<restorePool>` request cannot be used within a `<transaction>`.

Prerequisites

A pool with `PoolID` supplied in the pool Profile must not exist.

A subscriber with any of the keys supplied in the contained subscriber Profile(s) must not exist.

Request

`<restorePool>`

`<poolData>`

`<entity>`

```
<name>PoolProfile</name>
<seqnum>entitySeqnum1</seqnum>
<data> <![CDATA[
poolEntityData1
]]></data>
```

`</entity>`

[

`<entity>`

```
<name>poolEntityName2</name>
<seqnum>poolEntitySeqnum2</seqnum>
<data> <![CDATA[
poolEntityData2
]]></data>
```

`</entity>`

:

`<entity>`

```
<name>poolEntityNameN</name>
<seqnum>poolEntitySeqnumN</seqnum>
<data> <![CDATA[
poolEntityDataN
]]></data>
```

`</entity>`

]

[

`<subscriberData>`

`<poolId>poolId</poolId> | <poolId/> >`

`<ae>ae</ae>`

`<subscriberLastUpdateTime>subscriberLUT</subscriberLastUpdateTime>`

`<notificationSubscriptionLastUpdateTime>nsLUT</notificationSubscriptionLastUpdateTime>`

```

    <entity>
      <name>Profile</name>
      <seqnum>entitySeqnum1</seqnum>
      <data> <![CDATA[
        entityData1
      ]]></data>
    </entity>
  [
    <entity>
      <name>entityName2</name>
      <seqnum>entitySeqnum2</seqnum>
      <data> <![CDATA[
        entityData2
      ]]></data>
    </entity>
    :
    <entity>
      <name>entityNameN</name>
      <seqnum>entitySeqnumN</seqnum>
      <data> <![CDATA[
        entityDataN
      ]]></data>
    </entity>
  ]
  [
    <notificationSubscription>
      <client>client1</client>
      <userIdentity>userIdentity1</userIdentity>
      <creationTime>creationTime1</creationTime>
      <expirationTime>expirationTime1</expirationTime>
      <notEff>notEff1</notEff>
    </notificationSubscription>
    :
    <notificationSubscription>
      <client>clientN</client>
      <userIdentity>userIdentityN</userIdentity>
      <creationTime>creationTimeN</creationTime>
      <expirationTime>expirationTimeN</expirationTime>
      <notEff>notEffN</notEff>
    </notificationSubscription>
  ]
</subscriberData>
:
<subscriberData>
  // see above for repeated format of <subscriberData>
</subscriberData>
]
</poolData>
</restorePool>

```

- ***poolEntityNameX***: A user defined entity type/name for the pooled entity
 - Value is either `PoolProfile`, `PoolQuota`, `PoolState`, or `PoolDynamicQuota`
- ***poolEntitySeqnumX***: The sequence number associated with the pooled entity (as used on the Diameter Sh interface)
 - Values: 0– 65535
- ***poolEntityDataX***: Contents of the pooled XML data “blob”
- ***poolId***: PoolID value of the pool the subscriber is a member of. Numeric value, 1-22 digits in length
 - Values: 1–999999999999999999999999
 - **Note**: This should always be the same as the PoolID value in the PoolProfile
- ***ae***: Flag indicating if the subscriber is considered as auto-enrolled or not
 - Value is either `true` or `false`
 - **Note**: For pool members, the value of *ae* should always be `false`
- ***subscriberLUT***: The date/time when the data for the subscriber was last updated. Numeric value indicating number of seconds since midnight on January 1st 1970, in UTC time
 - Values: 0– 4294967295
- ***nsLUT***: The most recent date/time when a notification subscription for the subscriber was either added or removed. Numeric value indicating number of seconds since midnight on January 1st 1970, in UTC time
 - Values: 0– 4294967295
- ***entityNameX***: A user defined entity type/name for the subscriber entity
 - Value is either `Profile`, `Quota`, `State`, or `DynamicQuota`
- ***entitySeqnumX***: The sequence number associated with the subscriber entity (as used on the Diameter Sh interface)
 - Values: 0– 65535
- ***entityDataX***: Contents of the subscriber XML data “blob”
- ***clientX***: Subscribing client name of the subscribing entity. For example, a Diameter FQDN as used on the Diameter Sh interface
- ***userIdentityX***: The identity value supplied by the subscribing client to identify the subscriber (such as an IMSI, MSISDN, NAI etc.) in network specific format
- ***creationTimeX***: The date/time when the notification subscription was created. Numeric value indicating number of seconds since midnight on January 1st 1970, in UTC time
 - Values: 0– 4294967295
- ***expirationTimeX***: The date/time when the notification subscription will expire. Numeric value indicating number of seconds since midnight on January 1st 1970, in UTC time
 - Values: 0– 4294967295
- ***notEffX***: Flag indicating if the client allows multiple entities to be included in a single notification (for example when sending a PNR request on the Diameter Sh interface)
 - Value is either `true` or `false`

Response

If the request fails, a failure response will be indicated as described in section 3.2.5. The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

Error Codes

Error Code	Description
------------	-------------

Error Code	Description
InvalidXml	Invalid XML.
KeyNotFound	Key Not Found. A subscriber/pool with the given key cannot be found
KeyValueInvalid	The key value supplied is invalid, due to invalid characters/format etc.
EntityDefinitionNotFound	Entity Definition Not Found
InvalidInputXml	Invalid Input XML.
HwtRetryLimitExceeded	Data could not be committed to database as the total number of retries to commit database transactions exhausted.
Inconsistent	Database is inconsistent
KeyAlreadyExists	Key Already Exists. A subscriber/pool already exists with the given key

Examples

Request #1

A request is made to restore a pool that has PoolQuota and PoolState data. The pool does not contain any subscribers.

```
<restorePool>
  <poolData>
    <entity>
      <name>PoolProfile</name>
      <seqnum>4</seqnum>
      <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><pool><field name="PoolID">100000</field><field name="BillingDay">1</field><field name="Entitlement">DayPass</field><field name="Entitlement">DayPassPlus</field></pool>]]></data>
    </entity>
    <entity>
      <name>PoolQuota</name>
      <seqnum>27</seqnum>
      <data><![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>2015-07-22T00:00:00-05:00</nextResetTime></quota></usage>]]></data>
    </entity>
    <entity>
      <name>PoolState</name>
      <seqnum>11</seqnum>
      <data><![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>2015-08-09T11:20:32</value></property><property><name>approved</name><value>yes</value></property></state>]]></data>
    </entity>
  </poolData>
</restorePool>
```

Response #1

The request is successful, and the pool was restored.

Request #2

A request is made to restore a pool that contains 2 subscribers.

```
<restorePool>
```

```

<poolData>
  <entity>
    <name>PoolProfile</name>
    <seqnum>4</seqnum>
    <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><pool><field name="PoolID">100000</field><field name="BillingDay">1</field><field name="Entitlement">DayPass</field><field name="Entitlement">DayPassPlus</field></pool>]]></data>
  </entity>
  <entity>
    <name>PoolQuota</name>
    <seqnum>27</seqnum>
    <data><![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>2015-07-22T00:00:00-05:00</nextResetTime></quota></usage>]]></data>
  </entity>
  <entity>
    <name>PoolState</name>
    <seqnum>11</seqnum>
    <data><![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>2015-08-09T11:20:32</value></property><property><name>approved</name><value>yes</value></property></state>]]></data>
  </entity>
  <subscriberData>
    <poolId>100000</poolId>
    <ae>false</ae>
    <subscriberLastUpdateTime>1436798453</subscriberLastUpdateTime>
    <notificationSubscriptionLastUpdateTime>1436817543</notificationSubscriptionLastUpdateTime>
  </subscriberData>
  <entity>
    <name>Profile</name>
    <seqnum>0</seqnum>
    <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><subscriber><field name="AccountId">10404723525</field><field name="MSISDN">15141234567</field><field name="IMSI">302370123456789</field><field name="BillingDay">1</field><field name="Tier">Gold</field><field name="Entitlement">DayPass</field><field name="Entitlement">DayPassPlus</field></subscriber>]]></data>
  </entity>
  <entity>
    <name>Quota</name>
    <seqnum>12</seqnum>
    <data><![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>2015-07-22T00:00:00-05:00</nextResetTime></quota></usage>]]></data>
  </entity>
  <notificationSubscription>
    <client>mpe6.policy.operator.com</client>
    <userIdentity>tel:15141234567</userIdentity>
    <creationTime>1436798453</creationTime>
    <expirationTime>0</expirationTime>
    <notEff>true</notEff>
  </notificationSubscription>
</subscriberData>
<subscriberData>
  <poolId>100000</poolId>
  <ae>false</ae>
  <subscriberLastUpdateTime>1436792233</subscriberLastUpdateTime>
  <notificationSubscriptionLastUpdateTime>1436816011</notificationSubscriptionLastUpdateTime>
</subscriberData>
  <entity>
    <name>Profile</name>
    <seqnum>17</seqnum>

```



```

    <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><subscriber><field name="Account
Id">10404712881</field><field name="MSISDN">15145551234</field><field name="IMSI">30237055555
4444</field><field name="BillingDay">3</field><field name="Tier">Silver</field><field name="E
ntitlement">DayPass</field></subscriber>]]></data>
  </entity>
  <entity>
    <name>Quota</name>
    <seqnum>20</seqnum>
    <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><usage><version>3</version><quot
a name="AggregateLimit"><cid>9223372036858772097</cid><time>3122</time><totalVolume>2000</tot
alVolume><inputVolume>9220</inputVolume><outputVolume>30</outputVolume><serviceSpecific>222</
serviceSpecific><nextResetTime>2015-07-22T00:00:00-05:00</nextResetTime></quota></usage>]]></
data>
  </entity>
</subscriberData>
</poolData>
</restorePool>

```

Response #2

The request is successful, and the pool containing two subscribers was restored.

Request #3

A request is made to restore an enterprise pool.

```

<restorePool>
  <poolData>
    <entity>
      <name>PoolProfile</name>
      <seqnum>4</seqnum>
      <data><![CDATA[<?xml version="1.0" encoding="UTF-8"?><pool><field name="PoolID">100000<
/field><field name="BillingDay">1</field><field name="Entitlement">DayPass</field><field name
="Entitlement">DayPassPlus</field><field name="Type">Enterprise</field></pool>]]></data>
    </entity>
  </poolData>
</restorePool>

```

Response #2

The request is successful, and the pool is restored as an enterprise pool.

Appendix A. Error Codes

Error codes are returned in the `errorValue` code of the import log file response when a request fails (see section 3.2.5). The complete set of error codes and their associated values are defined in the following table.

The “Type” column indicates if an error is permanent (“P”) or temporary (“T”), or indicates success (“S”). A request that results in a permanent error should be discarded and not sent again. A request that results in a temporary can be sent again at a different time, and may be successful.

Error codes that are marked with a “*” are permanent errors that can be fixed by means of configuration, such as configuring the entities/fields in the SEC etc.

Table 19: Error Codes

Error Code	Value	Type	Description
Success	0	S	Success
MissingArgument	1	P	An internal error has occurred
ImportFileError	2	T	An internal error has occurred
LogFileError	3	T	An internal error has occurred
InitError	4	T	An internal error has occurred
ProvProhibited	5	T	Service is unavailable. Provisioning has been disabled
InvalidXml	6	P	Invalid XML
SessionTimeOut	7	T	No response received from UDRBE for a provisioning request
TooBigMessage	8	P*	The provisioning request size exceeded the maximum allowed size
CallbackNotRegistered	9	T	An internal error has occurred
InternalError	10	T	An internal error has occurred
InterfaceEntityNameNotFound	11	P*	Interface Entity Not Found
EntityNotFound	12	P*	Entity Not Found
EntityDefinitionNotFound	13	P*	Entity Definition Not Found
VersionBaseFieldSetNotFound	14	P	Versioned Base Field Set for the Transparent Entity Not Found
NonVersionedBaseFieldSetNotFound	15	P	Non Versioned Base Field Set for the Transparent Entity Not Found
MultipleVersionTagsFound	16	P	Multiple Version Tags Found
ElementNotDefined	17	P*	An XML Element is not defined
FieldValueNotValid	18	P*	Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC
OccurrenceConstraintViolation	19	P*	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field
RepeatableFieldSetElementInvalid	20	P	Invalid Repeatable Element
InvalidInputXml	21	P	Invalid Input XML
BaseFieldSetNotFound	22	P*	Base Field Set for Transparent Entity Not Found

Error Code	Value	Type	Description
FieldSetNotFound	23	P	Field Set Not Found
FieldSetAlreadyExists	24	P	Field Set Already Exists
FieldNotFound	25	P	Field Not Found
FieldAlreadyExists	26	P	Field Already Exists
FieldNotMultiValued	27	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
PathNotFound	28	P*	Specified XPath Not Found in XML
FieldSetDefinitionNotFound	29	P	Field Set Not Defined
FieldDefinitionNotFound	30	P*	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FieldNotUpdatable	31	P*	Field Cannot be Updated. The field is defined in the SEC as not be updatable
EntityDefinitionNoReset	32	P*	Entity Cannot be Reset. The reset command cannot be used on the requested entity
MultipleRowsFound	33	P	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
InvalidOperationType	34	P	An internal error has occurred
InvalidResponseType	35	P	An internal error has occurred
XmlBuildError	36	P	An internal error has occurred
XmlParseError	37	P	An internal error has occurred
DbError	38	P	Database Operation Failed
KeyNotFound	39	P	Key Not Found. A subscriber/pool with the given key cannot be found
KeyAlreadyExists	40	P	Key Already Exists. A subscriber/pool already exists with the given key
SubscriberIsPoolMember	41	P	Subscriber is Pool Member. The subscriber is a member of a pool. A subscriber cannot be deleted if they are a pool member
PoolNotEmpty	42	P	Has Pool Members. A pool cannot be deleted when it has member subscribers
MemberAlreadyExists	43	P	Already a Pool Member. The subscriber is already a member of a pool
PoolLimit	44	P	Pool Member List Max Limit Reached
NotAPoolMember	45	P	Not A Pool Member
NonEmptyXPathForOpaqueData	46	P	XPath cannot be non-empty for an Opaque-data operation
RegisterDataNotFound	47	P	Register Data Not Found
RegisterAlreadyExists	48	P	Register Already Exists
NoResultFound	49	T	An internal error has occurred
OperationNotAllowed	50	P	Operation Not Allowed

Error Code	Value	Type	Description
KeyValueInvalid	51	P	The key value supplied is invalid, due to invalid characters/format etc.
InterfaceNotSupported	52	P*	Requested Provisioning Interface Is Not Supported
PoolNotFound	53	P	Pool does not exist. A subscriber cannot be added or removed from a pool that does not exist
OutstandingCookieIdLimitReached	54	T	An internal error has occurred
MessageQueueFull	55	T	An internal error has occurred
RowNotFound	56	P	Data row specified is not found
DbRetryExhausted	57	T	Data could not be committed to database as the total number of retries to commit database transactions exhausted. NOTE: The client shall retry the command again
DurabilityDegraded	58	T	Data could not be committed as Durability is degraded. NOTE: The client shall retry the command again
DurabilityTimeout	59	T	Data could not be made durable within the configured Durability Timeout. NOTE: The client shall retry the command again to get the data sent in the failed request to verify that it was stored by last request
UnattemptedRequest	60	T	Request in transaction could not be attempted because a prior request failed
MemThresholdReached	61	P	Free system memory is low. Request cannot be performed
UdrbeConnectionFailure	62	T	Internal connection to the UDR provisioning back-end has failed
MultipleKeysNotMatch	63	P	All supplied keys do not correspond to the same subscriber
OneKeyRequired	64	P	A subscriber must have at least one key value
CpuCongestion	65	T	Request rejected due to system congestion
HwtRetryLimitExceeded:	66	T	Data could not be committed to database as the total number of retries to commit database transactions exhausted. NOTE: The client shall retry the command again
Inconsistent	67	P	Database is inconsistent
MaxMembersBasicPool	68	P	Basic Pool Member List Maximum Limit Reached
EnterpriseToBasicPoolFailed	69	P	Enterprise to basic Pool Conversion Failed Threshold Exceeded

Appendix B. Bulk Import/Export Variables

The bulk import/export has a set of system variables that affect its operation as it runs. Bulk import/export variables (shown below in Table 20) can be set via the UDR GUI and can be changed at runtime to effect dynamic server reconfiguration.

Table 20: Bulk Import/Export variables

Parameter	Description
Remote Host IP Address	The IP address and username of Remote Import/Export Host.
Remote Export Transfers Enabled	Whether or not to allow export files to be copied to the Remote Export Host. DEFAULT = UNCHECKED
Local Export Directory	The local directory where export files are created. DEFAULT = /var/TKLC/db/filemgmt/provexport; RANGE = 0-255 characters
Remote Export Directory	The directory in the Remote Export Host to which export files are transferred if configured. DEFAULT = ; RANGE = 0-255 characters
Remote Import Enabled	Whether or not import files are imported from a Remote Host. DEFAULT = UNCHECKED
Remote Import Directory	The directory in which import files exist on the Remote Host. DEFAULT = ; RANGE = 0-255 characters
PNR Generation with Import	If checked, PNR(s) will be generated for subscribers with an active subscription if a relevant subscriber or pool is updated or deleted. DEFAULT = UNCHECKED

Appendix C. My Oracle Support (MOS)

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

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

1. Select **2** for New Service Request
2. Select **3** for Hardware, Networking and Solaris Operating System Support
3. Select **2** for Non-technical issue

You will be connected to a live agent who can assist you with MOS registration and provide Support Identifiers. Simply mention you are a Tekelec Customer new to MOS.

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

Appendix D. Customer Training

Oracle University offers expert training on Oracle Communications solutions for service providers and enterprises. Make sure your staff has the skills to configure, customize, administer and operate your communications solutions, so that your business can realize all of the benefits that these rich solutions offer.

Visit the Oracle University web site to view and register for Oracle Communications training:
education.oracle.com/communication.

To reach Oracle University:

- In the US, please dial 800-529-0165.
- In Canada, please dial 866-825-9790.
- In Germany, please dial 0180 2000 526 (toll free) or +49 8914301200 (International).
- In Spain, please dial +34 91 6267 792.
- In the United Kingdom, please dial 0845 777 7 711 (toll free) or +44 11 89 726 500 (International).

For the appropriate country or region contact phone number for the rest of the world, please visit Oracle University's web site at www.oracle.com/education/contacts.

Appendix E. Locate Product Documentation on Oracle Help Center

Oracle 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 www.adobe.com.

1. Go to the Oracle Communications page on Oracle Help Center at: <http://docs.oracle.com/en/industries/communications/index.html>
2. Navigate to your product and release number to see the list of books for that release. You can view or download any document in PDF format.