

**Oracle® Communications  
User Data Repository**

Import/Export File Interface Reference

Release 10.0

**E56966 Revision 01**

November 2014

Oracle® Communications Import/Export File Interface Reference, Release 10.0

Copyright © 2014, 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, 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 on 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. 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.

# Table of Contents

<b>Chapter 1: Introduction.....</b>	<b>9</b>
Overview.....	10
Scope and Audience.....	10
Manual Organization.....	10
Documentation Admonishments.....	10
Related Publications.....	11
Locate Product Documentation on the Oracle Technology Network Site.....	11
Customer Training.....	12
My Oracle Support (MOS).....	12
Emergency Response.....	12
<b>Chapter 2: System Architecture.....</b>	<b>14</b>
System Architecture Overview.....	15
Database Transactions.....	17
Block Transaction Mode.....	17
ACID-Compliance.....	20
Behavior During Low Free System Memory.....	21
<b>Chapter 3: Bulk Operations.....</b>	<b>22</b>
Message Conventions.....	23
Import.....	24
Configuring Import Options.....	25
Import Files.....	25
Import File Format.....	25
Import File Comments.....	29
Import Log Files.....	30
Import Status.....	32
Export.....	33
XMLExport.....	34
Export File and Format.....	35
Export Conversion Tool (xmlconverter).....	39
Configuring Export Options.....	40
Scheduling Exports.....	40

Export Status.....	42
<b>Chapter 4: UDR Data Model.....</b>	<b>43</b>
Data Model Overview.....	44
Subscriber Data.....	46
Subscriber Profile.....	46
Quota.....	48
State.....	51
Dynamic Quota.....	51
Pool Data.....	54
Pool Profile.....	54
Pool Quota.....	55
Pool State.....	57
Pool Dynamic Quota.....	58
<b>Chapter 5: Subscriber Provisioning.....</b>	<b>61</b>
Subscriber Profile Commands.....	62
Create Subscriber.....	62
Update Subscriber.....	66
Delete Subscriber.....	70
<b>Chapter 6: Pool Provisioning.....</b>	<b>73</b>
Pool Profile Commands.....	74
Create Pool.....	74
Delete Pool.....	76
Additional Pool Commands.....	78
Add Member to Pool.....	78
Remove Member from Pool.....	82
<b>Chapter 7: General Provisioning.....</b>	<b>85</b>
General Editing Commands.....	86
Create Data.....	86
Update Field.....	91
Update FieldSet.....	97
Delete Field.....	105
Delete FieldSet.....	110

<b>Chapter 8: Special Operations.....</b>	<b>114</b>
Special Operation Commands.....	115
Reset.....	115
<b>Appendix A: Error Codes.....</b>	<b>119</b>
Error Codes.....	120
<b>Appendix B: Bulk Import/Export Variables.....</b>	<b>124</b>
Bulk Import/Export Variables.....	125
<b>Glossary.....</b>	<b>126</b>

# List of Figures

Figure 1: UDR High Level Architecture.....16

Figure 2: UDR Import Capability.....24

Figure 3: Import Log File Format.....30

Figure 4: Import Log File — Import Successfully Completed Example.....31

Figure 5: Import Log File — Import Interrupted Example.....32

Figure 6: Import Status.....32

Figure 7: UDR Export Capability.....34

Figure 8: Generating Output File.....35

Figure 9: Export Schedule (Display).....40

Figure 10: Export Schedule (Insert).....41

Figure 11: Export Schedule (Edit).....41

Figure 12: Export Schedule (Delete).....42

Figure 13: Export Status.....42

Figure 14: Data Model.....46

# List of Tables

Table 1: Admonishments.....11

Table 2: Request Variable Definitions.....18

Table 3: Message Conventions.....23

Table 4: Import Log File Parameters.....30

Table 5: Import Status Table.....32

Table 6: Subscriber Profile Fields.....47

Table 7: Quota Instance Default Values.....49

Table 8: Supported Attribute Sequences.....51

Table 9: Dynamic Quota Sequence of Attributes.....51

Table 10: Pool Profile Fields.....54

Table 11: Pool Quota Fields.....56

Table 12: Supported Pool State Attribute Sequences.....58

Table 13: Pool Dynamic Quota Sequence of Attributes.....58

Table 14: Summary of Subscriber Profile Commands.....62

Table 15: Request Variable Definitions: Create Subscriber.....63

Table 16: Error Codes: Create Subscriber.....64

Table 17: Request Definitions: Update Subscriber.....67

Table 18: Error Codes: Update Subscriber.....68

Table 19: Request Variable Definitions: Delete Subscriber.....71

Table 20: Response Variable Definitions: Delete Subscriber.....71

Table 21: Summary of Pool Profile Commands.....74

Table 22: Request Variable Definitions: Create Pool.....75

Table 23: Error Codes0.....	75
Table 24: Request Variable Definitions: Delete Pool.....	77
Table 25: Error Codes: Delete Pool.....	77
Table 26: Summary of Additional Pool Commands.....	78
Table 27: Request Variable Definitions: Add Member to Pool.....	79
Table 28: Error Codes: Add Member to Pool.....	79
Table 29: Request Variable Definitions.....	83
Table 30: Error Codes: Add Member to Pool.....	83
Table 31: Summary of General Editing Commands.....	86
Table 32: Request Variable Definitions: Create Data.....	87
Table 33: Error Codes: Create Data.....	88
Table 34: Request Variable Definitions:.....	93
Table 35: Error Codes.....	94
Table 36: Request Variable Definitions: Update Field (Pool).....	98
Table 37: Error Codes.....	99
Table 38: Request Variable Definitions.....	106
Table 39: Error Codes.....	107
Table 40: Request Variable Definitions.....	110
Table 41: Error Codes.....	111
Table 42: Summary of Special Operation Commands.....	115
Table 43: Request Variable Definitions: Get poolId.....	115
Table 44: Error Codes: Get poolId.....	116
Table 45: UDR Error Codes.....	120
Table 46: Bulk Import/Export Variables.....	125



# Chapter 1

## Introduction

---

### Topics:

- *Overview.....10*
- *Scope and Audience.....10*
- *Manual Organization.....10*
- *Documentation Admonishments.....10*
- *Related Publications.....11*
- *Locate Product Documentation on the Oracle Technology Network Site.....11*
- *Customer Training.....12*
- *My Oracle Support (MOS).....12*
- *Emergency Response.....12*

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

## Overview

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

## Scope and Audience

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

## Manual Organization





This document is organized into the following chapters:

1. *Introduction* explains the purpose and organization of this documentation, defines its audience and admonishments, lists related publications and where to find them, and provides information about technical support and training.
2. *System Architecture* describes the UDR system architecture.
3. *Bulk Operations* describes bulk import and export operations.
4. *UDR Data Model* describes the subscriber data and their defined entities and attributes.
5. *Subscriber Provisioning* describes the subscriber provisioning commands and their parameters.
6. *Pool Provisioning* describes the pool provisioning commands and their parameters.
7. *General Provisioning* describes the general editing commands.
8. *Special Operations* describes the Reset command.
9. *Error Codes* describes the error codes that can appear when a request fails.
10. *Bulk Import/Export Variables* lists and describes the bulk import and export variables.

## Documentation Admonishments

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

Table 1: Admonishments

Icon	Description
 DANGER	<b>Danger:</b> (This icon and text indicate the possibility of <i>personal injury</i> .)
 WARNING	<b>Warning:</b> (This icon and text indicate the possibility of <i>equipment damage</i> .)
 CAUTION	<b>Caution:</b> (This icon and text indicate the possibility of <i>service interruption</i> .)
 TOPPLE	<b>Topple:</b> (This icon and text indicate the possibility of <i>personal injury and equipment damage</i> .)

## Related Publications

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

## Locate Product Documentation on the Oracle Technology Network Site

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

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

## Customer Training

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

<http://education.oracle.com/communication>

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

[www.oracle.com/education/contacts](http://www.oracle.com/education/contacts)

## My Oracle Support (MOS)

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

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

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

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

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

## Emergency Response

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

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

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

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

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

# Chapter 2

## System Architecture

---

### Topics:

- *System Architecture Overview.....15*
- *Database Transactions.....17*
- *Behavior During Low Free System Memory.....21*

This chapter provides an overview of Import/Export File Interface system architecture.

## System Architecture Overview

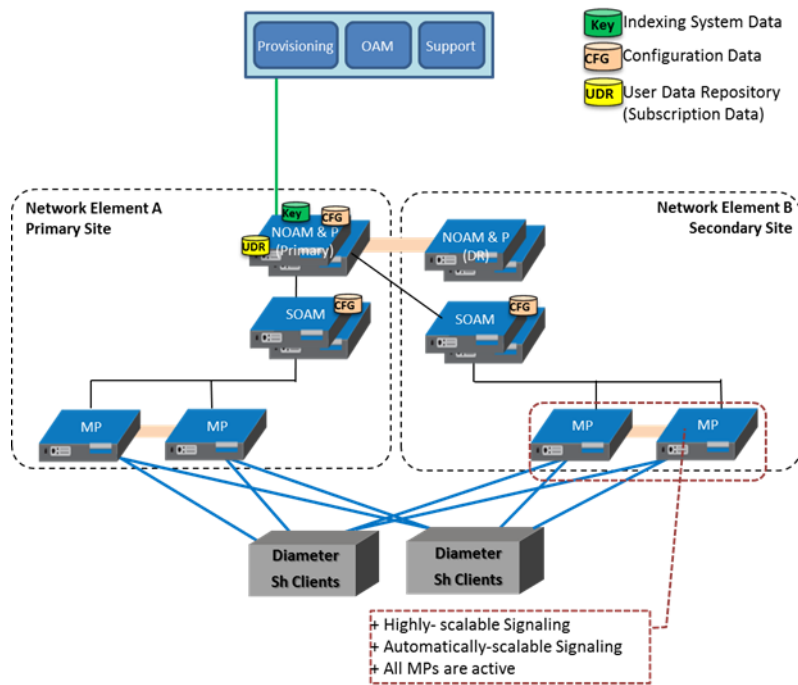
Oracle Communications Subscriber Profile Repository (UDR) performs the function of a subscriber profile repository (SPR), which is a database system that acts as a single logical repository that stores subscriber data. The subscriber data that traditionally has been stored in the HSS /HLR/AuC, Application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

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

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

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

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



**Figure 1: UDR High Level Architecture**

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

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

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

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

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

The bulk import provisioning interface provides following data manipulation commands:

- Subscriber:
  - Subscriber Profile create/delete
  - Subscriber Profile field create/modify/delete
  - Subscriber opaque data create/modify/delete
    - Quota, State and Dynamic Quota
  - Reset of Subscriber Quota opaque 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 subscriber membership operations
  - Add/remove from pool

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

## 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.

### Block Transaction Mode

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

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

If a block transaction fails, the 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 [Bulk Import/Export Variables](#). 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.

### Request Format

```
<transaction>
  <txRequest id="id1">
    request1
  </txRequest>
  [
    <txRequest id="id2">
      request2
    </txRequest>
    :
    <txRequest id="id3">
      requestN
    </txRequest>
  ]
</tx>
```

**Table 2: Request Variable Definitions**

Variable	Definition	Value
id	(Optional) Transaction id value provided in request, and will be passed back in the response	1-4294967295
requestX	Bulk import request contained in the transaction	<b>Note:</b> The maximum number of requests that can be included in a <transaction> transaction is 50.

### Examples

This request creates two 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>
    </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>

```

## 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 in which they are specified in the import file. If multiple requests for a subscriber must be performed in order, it is necessary to use a block transaction (see section [Block Transaction Mode](#)), 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
  Update Subscriber
  Update Subscriber
</transaction>
```

The request would then work as expected.

## ACID-Compliance

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

### Atomicity

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

### Consistency

Data across all requests performed inside a transaction is consistent.

### Isolation

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

### Durability

Once a transaction has been committed and becomes 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)

## 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]
```

# Chapter 3

## Bulk Operations

---

### Topics:

- *Message Conventions.....23*
- *Import.....24*
- *Export.....33*

This chapter explains import and export bulk operations.

## 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 3: Message Conventions**

Symbol	Description
<b>Bold</b>	Text that appears literally in the message (e.g. keywords, commas, parentheses, and error text are <b>bold</b> )
<i>italics</i>	Variable names when provided outside of a code example or value list.
spaces	Spaces (ie, zero or more space characters, " ") may be inserted anywhere except within a single name or number. At least one space is required to separate adjacent names or numbers.
...	Ellipses represent a variable number of repeated entries. For example:  <pre>dn DN1 , dn DN2 , ..., dn DN7 , dn DN8</pre>
< >	Angle brackets are used to enclose parameter values that are choices or names.  In the following example, the numbers represent specific value choices.  <pre>parameter1 &lt;1 2 3&gt;</pre> In the following example, ServerName represents the actual value.  <pre>parameter2 &lt;ServerName&gt;</pre> In the following example, the numbers represent a choice in the range from 0 to 3600.  <pre>parameter3 &lt;0..3600&gt;</pre>
[ ]	Square brackets are used to enclose an optional parameter and its value.  <pre>[ , parameter1 &lt; 1 2 3 &gt; ]</pre> A parameter and its value that are not enclosed in square brackets are mandatory.

Symbol	Description
	The pipe symbol is used in a parameter value list to indicate a choice between available values.  Parameter1 <1 2 3>
,	A literal comma is used in the message to separate each parameter that is specified.

## 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.

The following figure illustrates the capabilities of the UDR import.

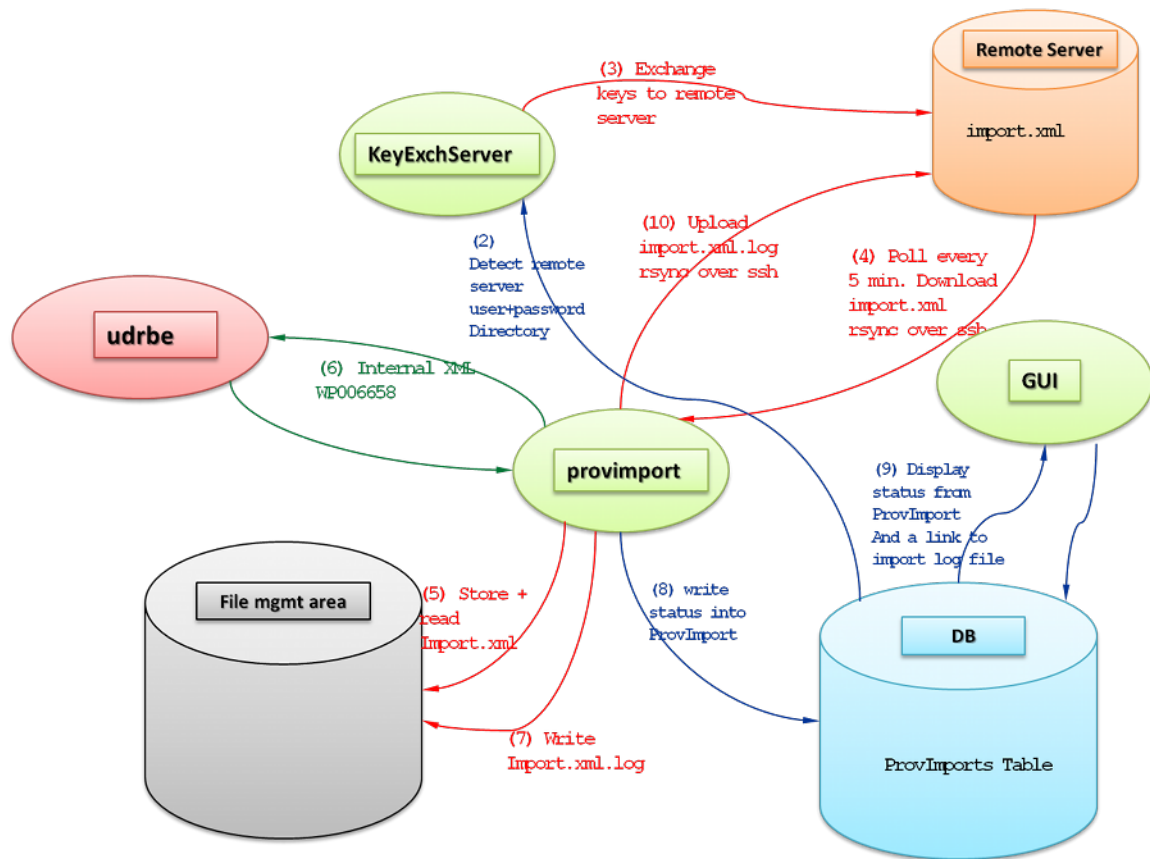


Figure 2: UDR Import Capability



## Configuring Import Options

The user can configure Import options by using the UDR's **Main Menu:UDR > Configuration > Provisioning Options** screen. Refer to [Bulk Import/Export Variables](#) for Provisioning Options.

## 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 the *Remote Host IP Address* field on the Provisioning Options (see [Bulk Import/Export Variables](#)). 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 [Bulk Import/Export Variables](#)). 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.

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 [Import Log Files](#)).

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

## 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 [Subscriber Provisioning](#). The data used in the imports is defined in section [Data Model Overview](#).

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

Operation	Description	Section
<createSubscriber>	Create a subscriber	<a href="#">Create Subscriber</a>
<updateSubscriber>	Update a subscriber	<a href="#">Update Subscriber</a>
<deleteSubscriber>	Delete a subscriber	<a href="#">Delete Subscriber</a>
<create>	Create a FieldSet	<a href="#">Create Data</a>

Operation	Description	Section
<updateField>	Update a Field	<a href="#">Update Field</a>
<updateFieldSet>	Update a FieldSet	<a href="#">Update FieldSet</a>
<deleteField>	Delete a Field	<a href="#">Delete Field</a>
<deleteFieldSet>	Delete a FieldSet	<a href="#">Delete FieldSet</a>
<createPool>	Create a Pool	<a href="#">Create Pool</a>
<deletePool>	Delete a Pool	<a href="#">Delete Pool</a>
<addPoolMember>	Add a pool member	<a href="#">Add Member to Pool</a>
<deletePoolMember>	Delete a pool member	<a href="#">Remove Member from Pool</a>
<reset>	Reset an Element	<a href="#">Reset</a>
<transaction>	Transaction container	<a href="#">Block Transaction Mode</a>

### 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">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* attribute indicates the request type, such as <createSubscriber>, <updateField>, etc.

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:** Since 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 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. <b>Note:</b> 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

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) 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.

**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 "**]]>**".

## 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** case-sensitive, for example "Subscriber", "Quota", and "Pool".

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.

## 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:

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

## 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 in 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 the UDR GUI's **Main Menu: Status & Manage > Files** 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.

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

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

...

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

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

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

Figure 3: Import Log File Format

Table 4: Import Log File Parameters

Parameter	Description
mm/dd/yy	Date the entry was logged. Values: <ul style="list-style-type: none"> <li>• mm = 01-12 (month)</li> <li>• dd = 01-31 (day of month)</li> <li>• yy = 00-99 (last two digits of the year)</li> </ul>
hh:mm:ss	Time the entry was logged. Values: <ul style="list-style-type: none"> <li>• hh = 00-23 (hours)</li> <li>• mm = 00-59 (minutes)</li> <li>• ss = 00-59 (seconds)</li> </ul>
linesImported	Number of lines of the import file that has been processed

Parameter	Description
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

**Note:** Comment lines are included in the counts for `percentCplt`, i.e. if an import file contains one request and one comment, the status may say 1 of 2 completed (i.e., 50% complete) after processing a comment line.

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

```
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 el
ement '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</fie
ld><field name="BillingDay">1</field><field name="Tier"></field><field name="Ent
itlement">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
```

**Figure 4: Import Log File — Import Successfully Completed Example**

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

```
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
```

Figure 5: Import Log File — Import Interrupted Example

### 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.

Main Menu: UDR -> Maintenance -> Import Status								
Import File	Time Queued	Time Started	Time Completed	Progress	Result Log	Pass Count	Fail Count	Status
<a href="#">import_5_delete_imsi.c...</a>	2013-04-18 10:43:08	2013-04-18 10:43:09	2013-04-18 10:43:15	100%	<a href="#">import_5_delete_imsi.c...</a>	999	0	Completed
<a href="#">import_6_delete_imsi.c...</a>	2013-04-18 10:45:04	2013-04-18 10:45:05	2013-04-18 10:45:10	100%	<a href="#">import_6_delete_imsi.c...</a>	978	21	Completed
<a href="#">import_7_delete_imsi.c...</a>	2013-04-18 10:48:09	2013-04-18 10:48:10	2013-04-18 10:48:15	100%	<a href="#">import_7_delete_imsi.c...</a>	0	999	Completed
<a href="#">import_8_delete_imsi.c...</a>	2013-04-18 10:55:00	2013-04-18 10:55:01	2013-04-18 10:55:06	100%	<a href="#">import_8_delete_imsi.c...</a>	950	49	Completed

Figure 6: Import Status

### 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 5: Import Status Table

Current State	Event	Action	Next State
-	XML file (*.ixml) found on Remote server	Start downloading the XML file	Transferring



Current State	Event	Action	Next State
		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	

## Export

UDR export will generate XML output to align with the output produced by 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.

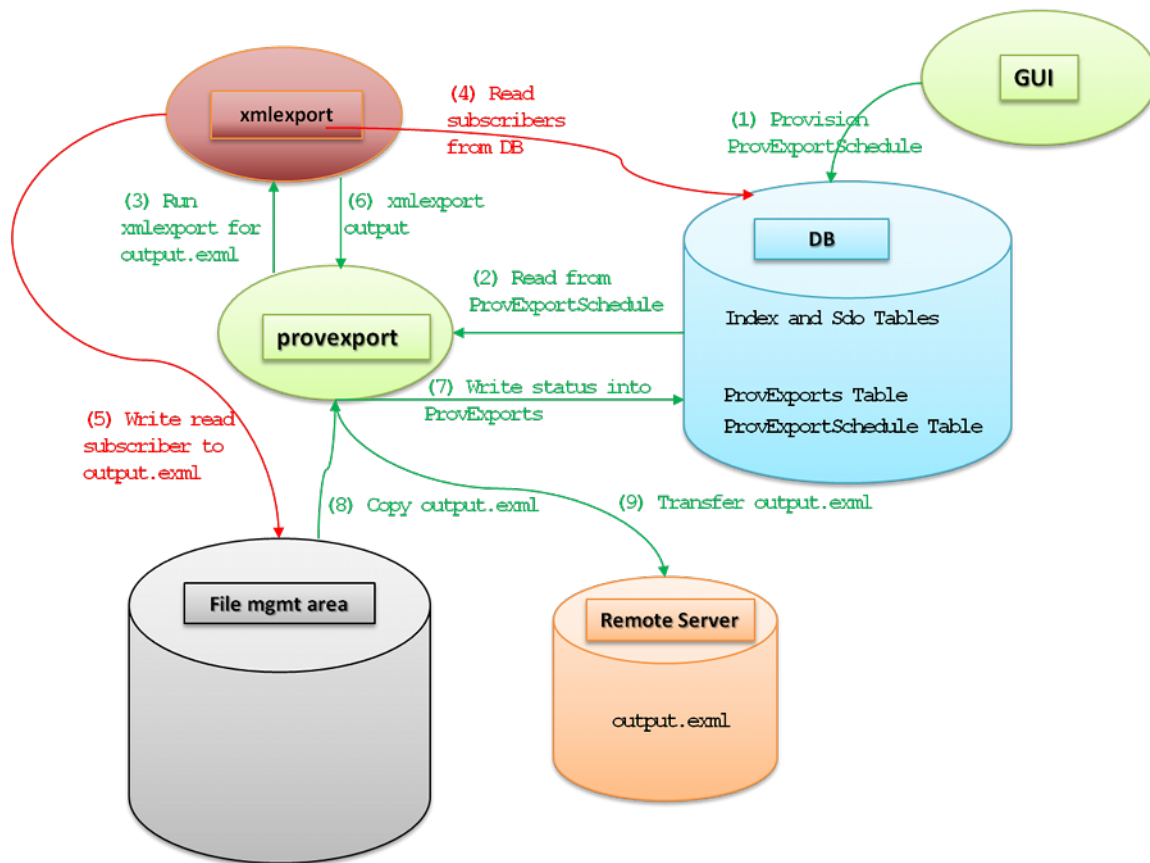


Figure 7: UDR Export Capability

## 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 look up 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 one million subscribers can be exported from the range specified. The following picture displays export in more detail:

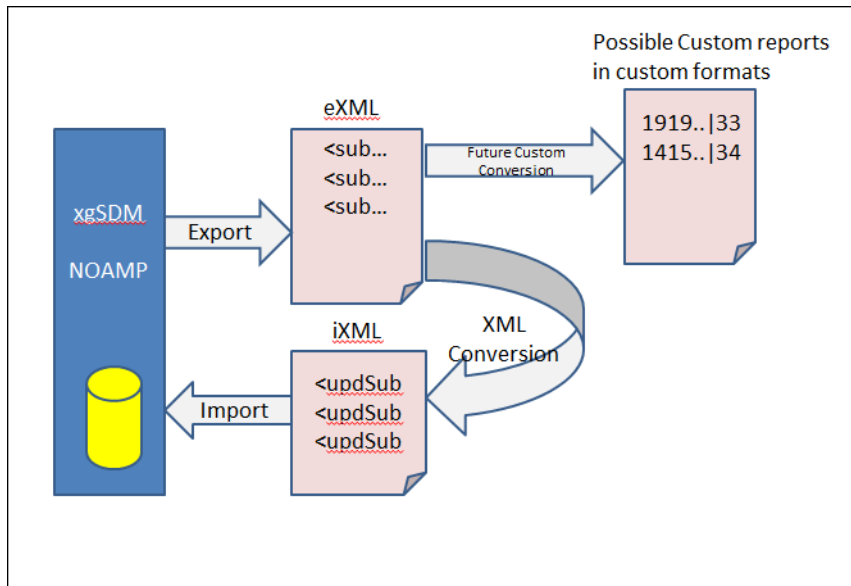


Figure 8: Generating Output File

### 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 [Bulk Import/Export Variables](#)). 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.

### 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.

## 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 "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 it. 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*.

## 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 it. 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*.

## 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">
  <poolId>1000</poolId>
  <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>
    <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>

```

## 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>
```

- 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.
- 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.

Example:

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

### Configuring Export Options

The user can configure Export options by using the UDR GUI's **Main Menu:UDR > Configuration > Provisioning Options**screen. Refer to [Bulk Import/Export Variables](#) for Provisioning Options.

### Scheduling Exports

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

#### Display

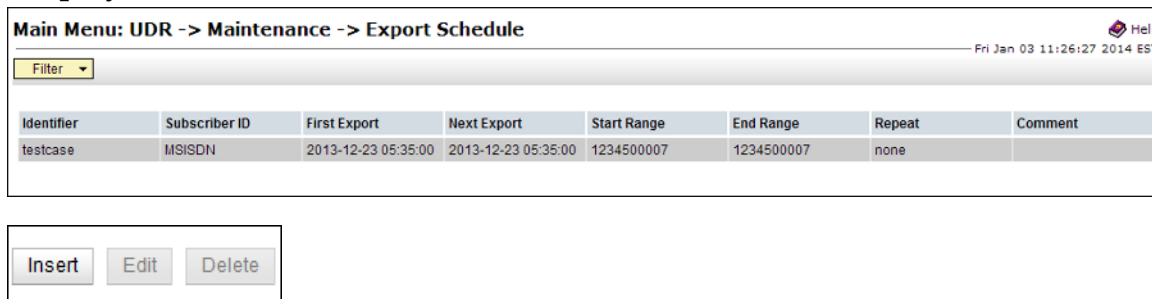


Figure 9: Export Schedule (Display)

#### 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 number of one



million subscribers would be exported from the range specified. If a range larger than one million subscribers is specified, the export will stop once it reaches one million subscribers. Exporting pools by specifying a range of PoolIDs is not supported.

**Main Menu: UDR -> Maintenance -> Export Schedule -> [Insert]**

Field	Value	Description
Identifier	<input type="text"/>	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	June 10 2014	The initial date on which this export should run.
Time	12 20 EDT	The initial time at which this export should run.
Start Range	<input type="text"/>	Start of range of data to be included in this export. [Range = 8-15 digits.]
End Range	<input type="text"/>	End of range of data to be included in this export. [Range = 8-15 digits.] Note: A maximum of 1 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	<input type="text"/>	Optional text that may be used to describe the purpose of this export. [Range = 0-255 characters.]

Ok Cancel

Figure 10: Export Schedule (Insert)

### Edit

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

**Main Menu: UDR -> Maintenance -> Export Schedule -> [Edit]**

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	May 30 2014	The initial date on which this export should run.
Time	16 25 EDT	The initial time at which this export should run.
Start Range	9888888880	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 1 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	<input type="text"/>	Optional text that may be used to describe the purpose of this export. [Range = 0-255 characters.]

Ok Cancel

Figure 11: Export Schedule (Edit)

### Delete

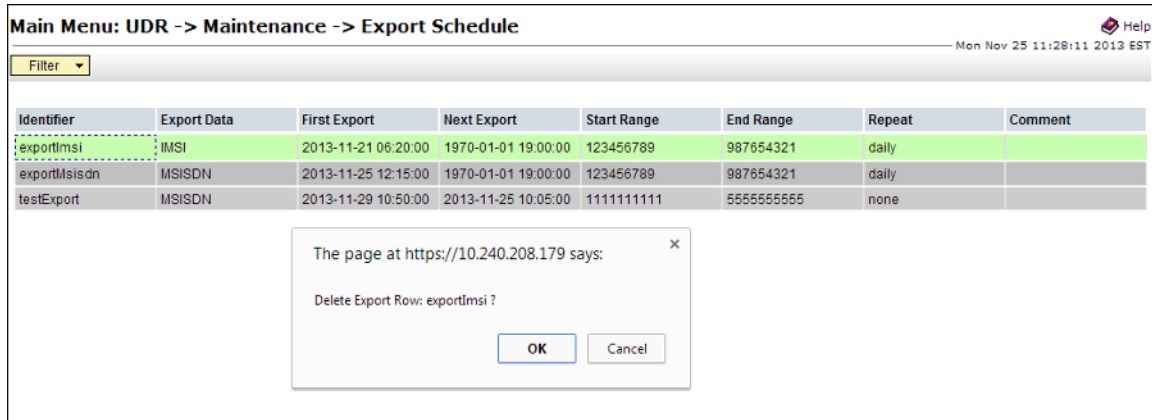


Figure 12: Export Schedule (Delete)

### 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.

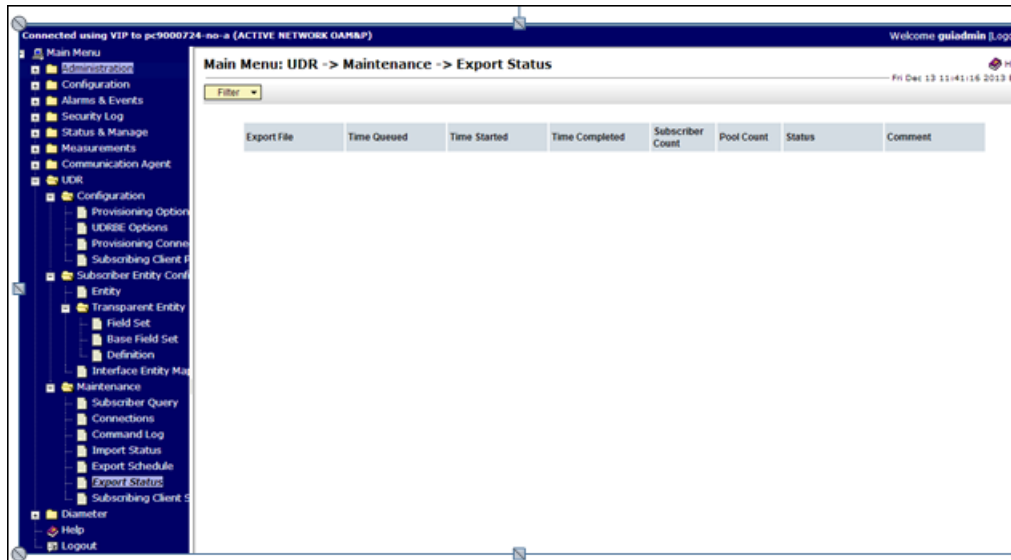


Figure 13: Export Status

# Chapter 4

## UDR Data Model

---

### Topics:

- *Data Model Overview.....44*
- *Subscriber Data.....46*
- *Pool Data.....54*

This chapter describes the UDR data model.

## Data Model Overview

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

The subscriber-related data includes:

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

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

The pool-related data includes:

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

The data architecture supports multiple Network Applications. This flexibility is achieved 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, and for every pool
  - Maps a subscription to the user identities through which it can be accessed
  - Maps an individual subscription to the pool of which they are a member
- User Identities
  - Use to map a specific user identity to a subscription
    - IMSI, MSISDN, NAI and AccountId map to an individual subscription

- PoolID maps to a pool subscription
- Pool Membership
  - Maps a pool to the list of the individual subscriber members

The Subscription Data Object (SDO):

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

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

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

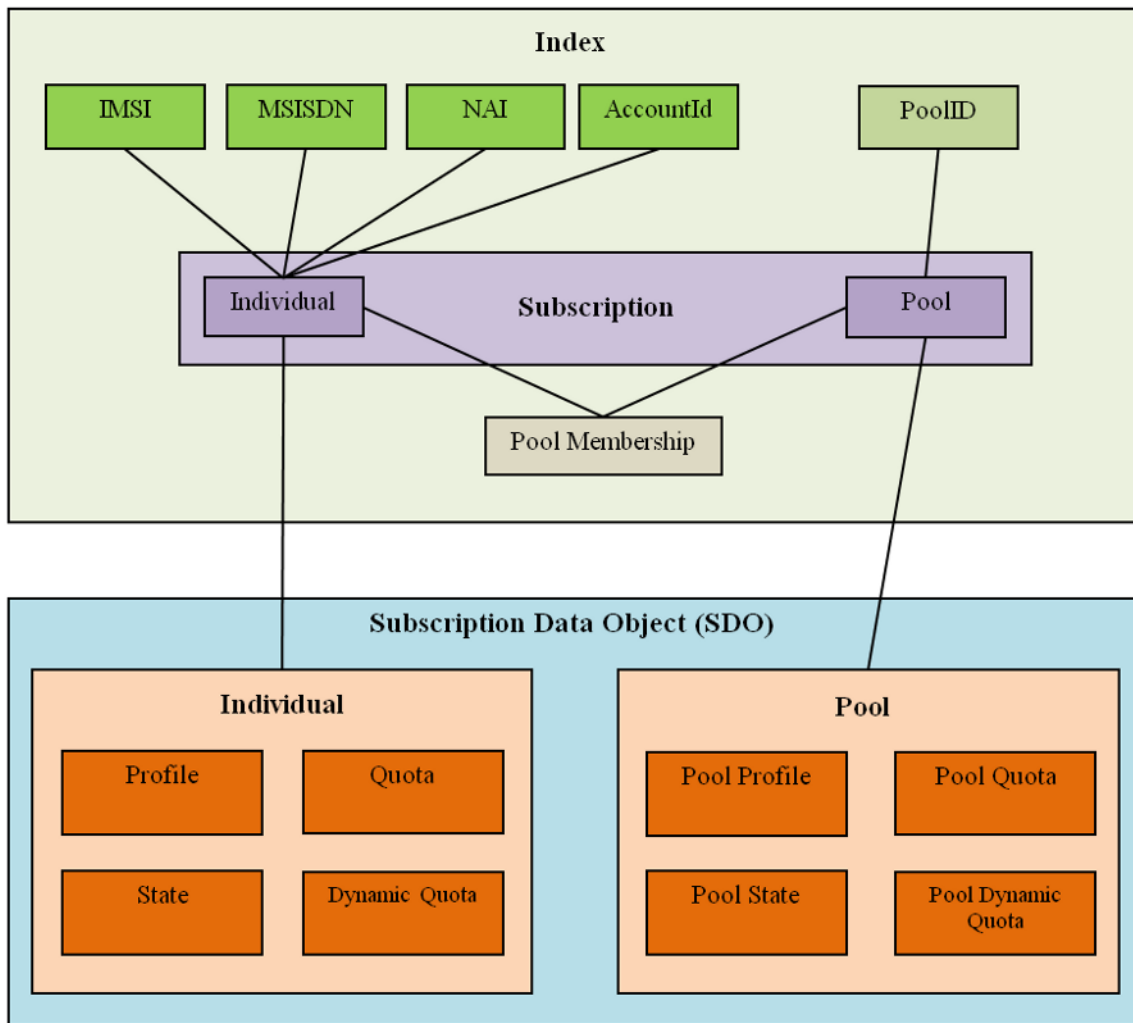


Figure 14: Data Model

## Subscriber Data

### Subscriber Profile

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

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

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

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

**Table 6: Subscriber Profile Fields**

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

Name (xml tag)	Type	Description
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.

## 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 may be present based on the version number value of the entity being accessed. In this product, only v3 of Quota is supported.

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

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

**Table 7: Quota Instance Default Values**

Name (xml tag)	Type	Default Value	Description	Quota Versions
usage	---	---	Sequence (multiplicity = 1)	
version	String	---	Version of the schema	
quota	---	---	Sequence (multiplicity = N)	
name	String	---	Quota name (identifier)	1/2/3
cid	String	---	Internal identifier used to identity a quota within a subscriber profile.	1/2/3
time	String	Empty string ""	Tracks the time-based resource consumption for a Quota.	1/2/3
totalVolume	String	"0"	Tracks the bandwidth volume-based resource consumption for a Quota.	1/2/3
inputVolume	String	"0"	Tracks the upstream bandwidth volume-based resource consumption for a Quota.	1/2/3
outputVolume	String	"0"	Tracks the downstream bandwidth volume-based resource consumption for a Quota.	1/2/3
serviceSpecific	String	Empty string ""	Tracks service-specific resource consumption for a Quota.	1/2/3

Name (xml tag)	Type	Default Value	Description	Quota Versions
nextResetTime	String	Empty string ""	Indicates the time after which the usage counters need to be reset. See below for date/time format	1/2/3
Type	String	Empty string ""	Type of the resource in use.	2/3
grantedTotalVolume	String	"0"	Represents the granted total volume of all the subscribers in the pool, in case of pool quota. In case of individual quota, it will represent the granted volume to all the PDN connections for that subscriber	2/3
grantedInputVolume	String	"0"	Granted Input Volume	2/3
grantedOutputVolume	String	"0"	Granted Output Volume	2/3
grantedTime	String	Empty string ""	Granted Total Time	2/3
grantedServiceSpecific	String	Empty string ""	Granted Service Specific Units	2/3
QuotaState	String	Empty string ""	State of the resource in use.	3
RefInstanceId	String	Empty string ""	Instance-id of the associated provisioned pass, top-up or roll-over.	3

**Note:** Date/Timestamp format is:

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

where:

- - = years before 0001
- CC = century
- YY = year
- MM = month
- DD = day
- T = Date/Time separator
- hh = hour
- mm = minutes

- ss = seconds
- Z = UTC (Coordinated Universal Time)
- +|- = time offset from UTC

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

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

## State

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

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

The State entity shall support the following sequence of attributes:

**Table 8: Supported Attribute Sequences**

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

## Dynamic Quota

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

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

The DynamicQuota entity shall support the following sequence of attributes:

**Table 9: Dynamic Quota Sequence of Attributes**

Name (xml tag)	Type	Description
definition	---	Sequence (multiplicity = 1)
version	String	Version of the schema
DynamicQuota	---	Sequence (multiplicity = N)

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

Name (xml tag)	Type	Description
expirationdatetime	String	The date/time after which the pass or top-up is considered to be exhausted See below for date/time format
purchasedatetime	String	The date/time when a pass was purchased See below for date/time format
Duration	String	The number of seconds after first use in which the pass must be used or expired. If both Duration and expirationdatetime are present, the closest expiration time is used
InterimReportingInterval	String	The number of seconds after which the GGSN/DPI/Gateway should revalidate quota grants with the PCRF

**Note:** Date/Timestamp format is:

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

where:

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

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

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

## Pool Data

### Pool Profile

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

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

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

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

**Table 10: Pool Profile Fields**

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	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.

Name (xml tag)	Type	Description
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.

## Pool Quota

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

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

The PoolQuota entity consists of the following sequence of attributes:

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

**Table 11: Pool Quota Fields**

Name (xml tag)	Type	Size	Default Value	Description
usage	---	---	---	Sequence (multiplicity = 1)
version	String	8	---	Version of the schema
quota	---	---	---	Sequence (multiplicity = N)
name	String	255		Quota name (identifier)
cid	String	255		Internal identifier used to identity a quota within a subscriber profile.
time	String	255	Empty string ""	Tracks the time-based resource consumption for a Quota.
totalVolume	String	255	"0"	Tracks the bandwidth volume-based resource consumption for a Quota.
inputVolume	String	255	"0"	Tracks the upstream bandwidth volume-based resource consumption for a Quota.
outputVolume	String	255	"0"	Tracks the downstream bandwidth volume-based resource consumption for a Quota.
serviceSpecific	String	255	Empty string ""	Tracks service-specific resource consumption for a Quota.
nextResetTime	String	255	Empty string ""	When set, it indicates the time after which the usage counters need to be reset. See below for date/time format
Type	String	255	Empty string ""	Type of the resource in use.
grantedTotalVolume	String	255	"0"	Represents the granted total volume of all the subscribers in the pool, in case of pool



Name (xml tag)	Type	Size	Default Value	Description
				quota. In case of individual quota, it will represent the granted volume to all the PDN connections for that subscriber.
grantedInputVolume	String	255	“0”	Granted Input Volume
grantedOutputVolume	String	255	“0”	Granted Output Volume
grantedTime	String	255	Empty string ""	Granted Total Time
grantedServiceSpecific	String	255	Empty string ""	Granted Service Specific Units
QuotaState	String	255	Empty string ""	State of the resource in use.
RefInstanceId	String	255	Empty string ""	Instance-id of the associated provisioned pass, top-up or roll-over.

**Note:** Date/Timestamp format is:

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

where:

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

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

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

### Pool State

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

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

The PoolState entity consists of the following sequence of attributes:

Table 12: Supported Pool State Attribute Sequences

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

## Pool Dynamic Quota

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

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

The PoolDynamicQuota entity shall support the following sequence of attributes:

Table 13: Pool Dynamic Quota Sequence of Attributes

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

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

**Note:** Date/Timestamp format is:

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

where:

- - = years before 0001
- CC = century
- YY = year
- MM = month
- DD = day
- T = Date/Time separator
- hh = hour
- mm = minutes
- ss = seconds

- Z = UTC (Coordinated Universal Time)
- +|- = time offset from UTC

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

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

# Chapter 5

## Subscriber Provisioning

---

### Topics:

- [Subscriber Profile Commands.....62](#)

This chapter describes subscriber provisioning commands.

## Subscriber Profile Commands

Table 14: 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	MSISDN, IMSI, NAI or AccountId	<deleteSubscriber>

### Create Subscriber

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>

```

**Table 15: Request Variable Definitions: Create Subscriber**

Variable	Definition	Value
IMSIX	IMSI value(s) corresponding to the subscriber. No values will be present if an IMSI is not provisioned for the subscriber	A string with 10 to 15 digits (if value is set)
MSISDNX	MSISDN value(s) corresponding to the subscriber. No values will be present if an MSISDN is not provisioned for the subscriber	A string with 8 to 15 digits (if value is set)
NAIX	NAI value(s) corresponding to the subscriber. No values will be present if an NAI is not provisioned for the subscriber	A string with 1 to 255 characters (if value is set) <b>Note:</b> 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	A string with 1 to 255 characters (if value is set)
dataName	A user defined entity type/name for the subscriber Profile	Subscriber
dataInterface	The interface type used to identify the bulk import/export interface	XMLIMPORT

Variable	Definition	Value
cdataProfile	<p>Contents of the XML data "blob" for the subscriber Profile</p> <p><b>Note:</b> Within &lt;key&gt; at least one key type is mandatory. Any combination of key types are allowed. Up to three occurrences of each repeatable key type (i.e. IMSI/MSISDN/NAI) are supported.</p> <p><b>Note:</b> Key order in the request is not important.</p>	

### Response

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the following Error Codes section.

### Error Codes

**Table 16: Error Codes: Create Subscriber**

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	XML input is invalid.
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* 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* 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]
```

## Update Subscriber

This operation replaces an existing subscriber profile, for the subscriber identified by *keyName* and *keyValue*.

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 of the *keyName/keyValue* supplied must exist.

**Request**

```

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

```

**Table 17: Request Definitions: Update Subscriber**

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> </ul>
keyValue	Corresponding key field value assigned to <i>keyName</i>	
IMSIX	IMSI value(s) corresponding to the subscriber. No values will be present if an IMSI is not provisioned for the subscriber	A string with 10 to 15 digits (if value is set)

Variable	Definition	Value
MSISDNX	MSISDN value(s) corresponding to the subscriber. No values will be present if an MSISDN is not provisioned for the subscriber	A string with 8 to 15 digits (if value is set)
NAIX	NAI value(s) corresponding to the subscriber. No values will be present if an NAI is not provisioned for the subscriber	A string with 1 to 255 characters (if value is set) <b>Note:</b> 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	A string with 1 to 255 characters (if value is set)
dataName	A user defined entity type/name for the subscriber Profile	Subscriber
dataInterface	The interface type used to identify the bulk import/export interface	XMLIMPORT
cdataProfile	Contents of the XML data "blob" for the subscriber Profile	

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

**Note:** In <subscriber>, any combination of key types are allowed. Up to three 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.

**Note:** Key order in the request is not important.

**Response**

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

**Error Codes**

**Table 18: Error Codes: Update Subscriber**

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.
OccurrenceConstraintViolation	Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field.

Error Code	Description
InvalidInputXml	XML input is invalid.
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.

## 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.

## Examples

### 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]
```

## Delete Subscriber

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

### Prerequisites

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

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

### Request

```
<deleteSubscriber>
  <key>
    <keyName>keyValue</keyName>
  </key>
</deleteSubscriber>
```

Table 19: Request Variable Definitions: Delete Subscriber

Variable Name	Definition	Values
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> </ul>
keyValue	Corresponding key field value assigned to <i>keyName</i>	

### Response

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the following Error Codes section.

### Error Codes

Table 20: Response Variable Definitions: Delete Subscriber

Variable	Definition
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

### 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 is 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* value indicates a subscriber with the given MSISDN does not exist.

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



# Chapter 6

## Pool Provisioning

---

### Topics:

- [Pool Profile Commands.....74](#)
- [Additional Pool Commands.....78](#)

This chapter describes pool provisioning commands.

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

Using bulk import, provisioning clients can create, retrieve, 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 [Update FieldSet](#).

**Note:** For command responses, the error code values described are listed in [Error Codes](#).

## Pool Profile Commands

Table 21: 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	<createPool>
Delete Pool	Delete pool profile data and all opaque data associated with the pool		<deletePool>

### Create Pool

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>
```

**Table 22: Request Variable Definitions: Create Pool**

Variable	Definition	Value
poolId	PoolID value of the pool being created Numeric value, 1-22 digits in length	1-99999999999999999999
dataName	A user defined entity type/name for the pool Profile	Pool
dataInterface	The interface type used to identify the bulk import/export interface	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 [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the following Error Codes section.

**Error Codes**

**Table 23: Error Codes0**

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	XML input is invalid.
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 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]
```

**Delete Pool**

This operation deletes all profile data (field-value pairs) and opaque data for the pool that is identified by the *poolId*.

**Prerequisites**

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

The pool must have no subscriber members, or the request will fail.

**Request**

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

**Table 24: Request Variable Definitions: Delete Pool**

Variable	Definition	Value
poolId	PoolID value of the pool being created. Numeric value, 1-22 digits in length	1-99999999999999999999

**Response**

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

**Error Codes**

**Table 25: Error Codes: Delete Pool**

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, and the affected rows are 0. The original request is not included.

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

## Additional Pool Commands

Table 26: Summary of Additional Pool Commands

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

### Add Member to Pool

This operation adds one or more subscribers to a pool.

**Prerequisites**

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

Separate subscribers with the keys of the *keyNameX/keyValueX* supplied must exist.

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

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

**Request**

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

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

**Table 27: Request Variable Definitions: Add Member to Pool**

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
subKeyNameX	A key field within the subscriber Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> </ul>
subKeyValueX	Corresponding field value assigned to <i>subKeyNameX</i>	

**Note:** Up to 10 subscribers can be added in one request.

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

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

**Response**

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

**Error Codes**

**Table 28: Error Codes: Add Member to Pool**

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
PoolLimit	Pool Member List Max Limit Reached

Error Code	Description
PoolNotFound	Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist

## Examples

### Request #1

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

```
<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 the that 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>
```



```

    </member>
  </members>
</addPoolMember>

```

**Response #3**

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

```
[error 44 errorText : line lineNumber]
```

**Request #6**

A request is made to add three 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 three subscribers are added to the pool.

## Remove Member from Pool

This operation removes one or more Subscribers from a Pool.

### Prerequisites

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

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

Table 29: Request Variable Definitions

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
subKeyNameX	A key field within the subscriber profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> </ul>
subKeyValueX	Corresponding key field value assigned to <i>subkeyName</i>	

**Note:** Up to 10 subscribers can be removed in one request.

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

### Response

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

### Error Codes

Table 30: Error Codes: Add Member to Pool

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, retrieved or removed from a pool that does not exist.

### Examples

#### Request #1

A request is made to remove a subscriber from a pool. Both the pool and the subscriber exist. The subscriber is a member of the pool. The request is not required in the response.

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

```
</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 three 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 three subscribers are removed from the pool.

# Chapter 7

## General Provisioning

---

### Topics:

- [General Editing Commands.....86](#)

This chapter describes the general editing commands.

**Note:** For command responses, the error code values described are listed in [Error Codes](#).

## General Editing Commands

Table 31: 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>

### Create Data

This operation creates an entity or row for the subscriber/pool identified by the *keyName* and *keyValue* in the request.

**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 of the *keyName/keyValue* 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.

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

#### Request

```
<create createEntityIfNotExist="createEntityIfNotExist">
  <key>
    <keyName>keyValue</keyName>
  </key>
  <entity>
    <data>
      <name>dataName</name>
      <interface>dataInterface</interface>
      <xpath>dataXPath</xpath>
    </data>
  </entity>
</create>
```

```
<content> <![CDATA[
  entityContent
]]></content>
</entity>
</create>
```

**Table 32: Request Variable Definitions: Create Data**

Variable	Definition	Value
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.
keyName	A key field within the subscriber Profile or pool Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> <li>• PoolID</li> </ul>
keyValue	Corresponding key field value assigned to <i>keyName</i>	
dataName	A user defined entity type/name for the transparent entity being updated	<ul style="list-style-type: none"> <li>• Subscriber</li> <li>• Quota</li> <li>• State</li> <li>• DynamicQuota</li> <li>• Pool</li> <li>• PoolQuota</li> <li>• PoolState</li> <li>• PoolDynamicQuota</li> </ul>
dataInterface	The interface type used to identify the bulk import/export interface	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	"/usage/quota[@name='quotaName']"
quotaName	(See dataXPath) The name that identifies the required quota row within the Quota entity	
entityContent	Content of entity/row being created	

**Response**

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

**Error Codes****Table 33: Error Codes: Create Data**

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
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

**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.

```

<create createEntityIfNotExist="true">
  <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>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.

```

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

```

**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>
]]>
    </content>
  </entity>
</create>
```

```

    <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]
```

**Update Field**

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

### Prerequisites

A subscriber/pool with the key of the `keyName/keyValue` 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.

### Request

```
<updateField clearAll="clearAll">
  <key>
    <keyName>keyValue</keyName>
  </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>
</updateField>
```

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

Table 34: Request Variable Definitions:

Variable	Definition	Value
clearAll	Indicates whether all existing value(s) in field(s) being updated should first be removed before adding the newly specified field value(s).	Value is either true or false. <b>Note:</b> For fields that are not multi-value (i.e. single value), the value of <i>clearAll</i> must be set to true else the request will attempt to add a second instance of the field, and the request will fail.
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> <li>• PoolID</li> </ul>
keyValue	Corresponding key field value assigned to <i>keyName</i>	
dataName	A user defined entity type/name for the transparent entity being updated	<ul style="list-style-type: none"> <li>• Subscriber</li> <li>• Quota</li> <li>• Pool</li> <li>• PoolQuota</li> </ul>
dataInterface	The interface type used to identify the bulk import/export interface	XMLIMPORT
dataXPath	XML XPath value which corresponds to the row element for which the reset operation needs to be performed	
fieldNameX	A user defined field within the transparent entity being updated	
fieldValueX	Corresponding field value assigned to <i>fieldNameX</i>	

### Response

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

## Error Codes

Table 35: 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
OperationNotAllowed	Operation Not Allowed

## Examples

## Request #1

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

```
<updateField clearAll="true">
  <key>
    <keyName>15141234567</keyName>
  </key>
  <entity>
    <data>
      <name>Subscriber</name>
      <interface>XMLIMPORT</interface>
      <xpath>/subscriber</xpath>
    </data>
  </entity>
</updateField>
```

```

</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 are updated.

```

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

```

**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>
    <keyName>15141234567</keyName>
  </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>
    <keyName>15141234567</keyName>
  </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>
    <keyName>15141234567</keyName>
  </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".

## Update FieldSet

This operation updates or creates an entity or row within an entity for the subscriber or pool identified by the specified key name and key value, 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 of the *keyName/keyValue* 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.

**Request**

```
<updateFieldSet [create="create"]
                [createEntityIfNotExist="createEntityIfNotExist">
  <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>
    <content>
      entityContent
    </content>
  </entity>
</updateFieldSet>
```

**Table 36: Request Variable Definitions: Update Field (Pool)**

Variable	Definition	Value
create	Indicates whether the row should be created if it does not already exists	Value is either true or false. <b>Note:</b> If the entity does not exist, and the value of <i>createEntityIfNotExist</i> is set to true, the value of <i>create</i> is ignored and the row will be created in the new entity
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.
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> </ul>

Variable	Definition	Value
		<ul style="list-style-type: none"> <li>• NAI</li> <li>• AccountId</li> <li>• PoolID</li> </ul>
keyValue	Corresponding key field value assigned to <i>keyName</i>	
dataName	A user defined entity type/name for the transparent entity being updated	<ul style="list-style-type: none"> <li>• Subscriber</li> <li>• Quota</li> <li>• State</li> <li>• DynamicQuota</li> <li>• Pool</li> <li>• PoolQuota</li> <li>• PoolState</li> <li>• PoolDynamicQuota</li> </ul>
dataInterface	The interface type used to identify the bulk import/export interface	XMLIMPORT
dataXPath	XML XPath value which corresponds to the row element for which the reset operation needs to be performed	<b>Note:</b> To update the entire entity (i.e. a complete opaque data replacement) the <i>dataXPath</i> value should be empty
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	

**Response**

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

**Error Codes**

**Table 37: Error Codes**

Error Code	Description
InterfaceEntityNameNotFound	Interface Entity Not Found

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
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

## 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[
<usage>
  <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>
</usage>
]]>
  </content>
</entity>
</updateFieldSet>

```

**Response #3**

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

```
[error 47 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[
<usage>
  <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>
</usage>
]]>
    </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[
<usage>
  <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>
</usage>
]]>
    </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>
      </version>
    </data>
  </entity>
</updateFieldSet>
```

```

        <value>3</value>
      </version>
    </data>
  </content>
<![CDATA[
<usage>
  <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>
</usage>
]]>
  </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[
<usage>
  <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>
</usage>
]]>
  </content>

```



```
</entity>
</updateFieldSet>
```

### Response #7

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

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

## Delete Field

This operation deletes the specified field(s) for the subscriber/pool identified by *keyName* and *keyValue* 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.

### Prerequisites

A subscriber/pool with the key of the *keyName/keyValue* 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.

### Request

```
<deleteField>
  <key>
    <keyName>keyValue</keyName>
  </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>

```

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

**Table 38: Request Variable Definitions**

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> <li>• PoolID</li> </ul>
keyValue	Corresponding key field value assigned to <i>keyName</i>	
dataName	A user defined entity type/name for the transparent entity being updated	<ul style="list-style-type: none"> <li>• Subscriber</li> <li>• Quota</li> <li>• State</li> <li>• DynamicQuota</li> <li>• Pool</li> <li>• PoolQuota</li> <li>• PoolState</li> <li>• PoolDynamicQuota</li> </ul>
dataInterface	The interface type used to identify the bulk import/export interface	XMLIMPORT
dataXPath	XML XPath value which corresponds to the row element for which the reset operation needs to be performed	
fieldNameX	A user defined field within the transparent entity being updated	
fieldValueX	(Optional) Corresponding field value assigned to <i>fieldNameX</i> . 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	<b>Note:</b> for multi-value fields, individual <i>fieldNameX</i> elements must be specified for each instance/value being deleted

**Response**

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

**Error Codes****Table 39: 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.

**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>
    <keyName>15141234567</keyName>
  </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.

**Delete FieldSet**

This operation deletes an entity, or row within an entity for the subscriber/pool identified by the *keyName* and *keyValue* in the request.

**Prerequisites**

A subscriber/pool with the key of the *keyName/keyValue* 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.

**Request**

```
<deleteFieldSet>
  <key>
    <keyName>keyValue</keyName>
  </key>
  <entity>
    <data>
      <name>dataName</name>
      <interface>dataInterface</interface>
      <xpath>dataXPath</xpath>
    </data>
  </entity>
</deleteFieldSet>
```

**Table 40: Request Variable Definitions**

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> <li>• PoolID</li> </ul>
keyValue	Corresponding key field value assigned to <i>keyName</i>	
dataName	A user defined entity type/name for the transparent entity being updated	<ul style="list-style-type: none"> <li>• Subscriber</li> <li>• Quota</li> <li>• State</li> <li>• DynamicQuota</li> <li>• Pool</li> <li>• PoolQuota</li> </ul>

Variable	Definition	Value
		<ul style="list-style-type: none"> <li>PoolState</li> <li>PoolDynamicQuota</li> </ul>
dataInterface	The interface type used to identify the bulk import/export interface	XMLIMPORT
dataXPath	XML XPath value which corresponds to the row element for which the reset operation needs to be performed	

### Request Variable Definitions

### Response

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

### Error Codes

**Table 41: 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.

### 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='Q1']</xpath>
    </data>
  </entity>
</deleteFieldSet>
```

#### Response #4

The request fails. The *errorValue* indicates the Quota entity does not contain a row called Q2.

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

# Chapter 8

## Special Operations

---

### Topics:

- [Special Operation Commands.....115](#)
- [Reset.....115](#)

This chapter describes the Reset command.

## Special Operation Commands

Table 42: Summary of Special Operation Commands

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

### Reset

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

**Note:** Currently, only the Quota entity has values that can be reset, but other entities can be reset if defined as such in the SEC.

#### Prerequisites

A subscriber with the key of the *keyName/keyValue* 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.

#### Request

```

<reset>
  <key>
    <keyName>keyValue</keyName>
  </key>
  <entity>
    <data>
      <name>dataName</name>
      <interface>dataInterface</interface>
      <xpath>dataXPath</xpath>
    </data>
  </entity>
</reset>

```

Table 43: Request Variable Definitions: Get poolId

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> <li>• IMSI</li> <li>• MSISDN</li> <li>• NAI</li> <li>• AccountId</li> </ul>

Variable	Definition	Value
		<ul style="list-style-type: none"> <li>PoolID</li> </ul>
keyValue	Corresponding key field value assigned to <i>keyName</i>	
dataName	A user defined entity type/name for the transparent entity being updated	Quota
dataInterface	The interface type used to identify the bulk import/export interface	XMLIMPORT
dataXPath	XML XPath value which corresponds to the row element for which the reset operation needs to be performed	Value is <code>"/usage/quota[@name='quotaName']"</code> for a Quota row
quotaName	(See dataXPath) The name that identifies the required quota row within the Quota entity	

### Response

If the request fails, a failure response will be indicated as described in section [Import Log Files](#). The different values of *errorValue* in the failure response are indicated in the Error Codes section below.

### Error Codes

**Table 44: Error Codes: Get poolId**

Error Code	Description
InterfaceEntityNameNotFound	Interface Entity Not Found
FieldSetNotFound	Field Set Not Found
FieldSetDefintionNotFound	Field Set Not Defined
EntityDefintionNoReset	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

**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.

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

**Response #2**

The request fails. 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]
```

# Appendix

# A

## Error Codes

---

### Topics:

- [Error Codes.....120](#)

This appendix lists and describes error codes that can appear when a request fails.

## Error Codes

Error codes are returned in the errorValue code of the import logfile response when a request fails (see section [Import Log Files](#)). 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 45: UDR 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



Error Code	Value	Type	Description
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
FieldSetNotFound	23	P	Field Set Not Found
FieldSetAlreadyExists	24	P	Field Set Already Exists
FieldNotFound	25	P	Field Not Found
FieldAlreadyExists	27	P	Field Already Exists
FieldNotMultiValued	28	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	29	P*	Specified XPath Not Found in XML
FieldSetDefinitionNotFound	30	P	Field Set Not Defined
FieldDefinitionNotFound	31	P*	Field Not Defined. The given field is not a valid field within the entity as defined in the SEC
FieldNotUpdatable	32	P*	Field Cannot be Updated. The field is defined in the SEC as not be updatable
EntityDefinitionNoReset	33	P*	Entity Cannot be Reset. The reset command cannot be used on the requested entity
MultipleRowsFound	34	P	Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria
InvalidOperationType	35	P	An internal error has occurred
InvalidResponseType	36	P	An internal error has occurred
XmlBuildError	37	P	An internal error has occurred
XmlParseError	38	P	An internal error has occurred
DbError	39	P	Database Operation Failed
KeyNotFound	40	P	Key Not Found. A subscriber/pool with the given key cannot be found
KeyAlreadyExists	41	P	Key Already Exists. A subscriber/pool already exists with the given key

Error Code	Value	Type	Description
SubscriberIsPoolMember	42	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	43	P	Has Pool Members. A pool cannot be deleted when it has member subscribers
MemberAlreadyExists	44	P	Already a Pool Member. The subscriber is already a member of a pool
PoolLimit	45	P	Pool Member List Max Limit Reached
NotAPoolMember	46	P	Not A Pool Member
NonEmptyXPathForOpaqueData	47	P	XPath cannot be non-empty for an Opaque-data operation
RegisterDataNotFound	48	P	Register Data Not Found
RegisterAlreadyExists	49	P	Register Already Exists
NoResultFound	50	P	An internal error has occurred
OperationNotAllowed	51	P	Operation Not Allowed
KeyValueInvalid	52	P	The key value supplied is invalid, due to invalid characters/format etc.
InterfaceNotSupported	53	P*	Requested Provisioning Interface Is Not Supported
PoolNotFound	54	P	Pool does not exist. A subscriber cannot be added or removed from a pool that does not exist
OutstandingCookieLimitReached	55	T	An internal error has occurred
MessageQueueFull	56	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. <b>Note:</b> The client shall retry the command again
DurabilityDegraded	58	T	Data could not be committed as Durability is degraded. <b>Note:</b> The client shall retry the command again
DurabilityTimeout	59	T	Data could not be made durable within the configured Durability Timeout. <b>Note:</b> 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

Error Code	Value	Type	Description
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

# Appendix B

## Bulk Import/Export Variables

---

### Topics:

- [Bulk Import/Export Variables.....125](#)

This Appendix lists and describes the bulk import and export variables.

## 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 (see [Bulk Import/Export Variables](#)) can be set via the Oracle Communications User Data Repository GUI and can be changed at runtime to effect dynamic server reconfiguration.

**Table 46: 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

## E

ESPR  
Enhanced Subscriber Profile Repository - Oracle Communications' database system that provides the storage and management of subscriber policy control data for PCRF nodes.

## I

IMSI  
International Mobile Subscriber Identity  
A unique internal network ID identifying a mobile subscriber.

## L

LDAP  
Lightweight Directory Access Protocol  
A protocol for providing and receiving directory information in a TCP/IP network.

## M

MP  
Message Processor - The role of the Message Processor is to provide the application messaging protocol interfaces and processing. However, these servers also have OAM&P components. All Message Processors replicate from their Signaling OAM's database and generate faults to a Fault Management System.

MSISDN  
Mobile Station International Subscriber Directory Number. The unique, network-specific subscriber number of a mobile communications subscriber.

**M**

MSISDN follows the E.164 numbering plan; that is, normally the MSISDN is the phone number that is used to reach the subscriber.

**N**

NAI

Network Access Identifier

The user identity submitted by the client during network authentication.

NOAM

Network Operations,  
Administration, and Maintenance**O**

OAMP

Operations, Administration,  
Maintenance and Provisioning**P**

PCRF

Policy and Charging Rules Function. The ability to dynamically control access, services, network capacity, and charges in a network.

Maintains rules regarding a subscriber's use of network resources. Responds to CCR and AAR messages. Periodically sends RAR messages. All policy sessions for a given subscriber, originating anywhere in the network, must be processed by the same PCRF.

**S**

SIP

Session Initiation Protocol

A peer-to-peer protocol used for voice and video communications.

SOAP

Simple Object Access Protocol

**S**

SS7

## Signaling System #7

A communications protocol that allows signaling points in a network to send messages to each other so that voice and data connections can be set up between these signaling points. These messages are sent over its own network and not over the revenue producing voice and data paths. The EAGLE is an STP, which is a device that routes these messages through the network.

**U**

UDR

User Data Repository - A logical entity containing user data