

Oracle® Communications
User Data Repository
REST Provisioning Interface Reference
Release 10.0
E53213 Revision 01

November 2014

Oracle® Communications REST Provisioning 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

Chapter 1: Introduction.....	12
Overview.....	13
Scope and Audience.....	13
Manual Organization.....	13
Documentation Admonishments.....	13
Related Publications.....	14
Locate Product Documentation on the Oracle Technology Network Site.....	14
Customer Training.....	15
My Oracle Support (MOS).....	15
Emergency Response.....	15
Chapter 2: System Architecture.....	17
Overview.....	18
Provisioning Interface.....	19
REST Application Server (RAS).....	20
Provisioning Clients.....	21
Security.....	22
Client Server IP Address White List.....	22
Secure Connection using SSLv3.....	22
Multiple Connections.....	24
Request Queue Management.....	25
Database Transactions.....	25
ACID Compliance.....	25
Atomicity.....	25
Consistency.....	25
Isolation.....	26
Durability.....	26
Connection Management.....	26
Connections Allowed.....	26
Disable Provisioning.....	26
Idle Timeout.....	27
Maximum Simultaneous Connections.....	27
TCP Port Number.....	27
Behavior During Low Free System Memory.....	27

Chapter 3: REST Interface Description.....	28
REST Conventions.....	29
HTTP(S) Request Headers.....	29
HTTP(S) Status Codes and Error Messages.....	30
Chapter 4: REST Interface Message Definitions.....	33
Message Conventions.....	34
HTTP Method.....	34
Base URL.....	34
REST URL.....	34
Case Sensitivity.....	35
List of Messages.....	36
Chapter 5: UDR Data Model.....	39
Overview.....	40
Subscriber Data.....	42
Subscriber Profile.....	42
Quota.....	44
State.....	46
Dynamic Quota.....	46
Pool Data.....	48
Pool Profile.....	48
Pool Quota.....	49
Pool State.....	51
Pool Dynamic Quota.....	51
Chapter 6: Subscriber Provisioning.....	54
Subscriber Profile Commands.....	55
Create Profile.....	55
Get Profile.....	59
Update Profile.....	61
Delete Profile.....	63
Subscriber Profile Field Commands.....	65
Add Field Value.....	66
Get Field.....	68
Get Field Value.....	70
Update Field.....	73

Update Multiple Fields.....	75
Delete Field.....	77
Delete Field Value.....	78
Subscriber Opaque Data Commands.....	80
Set Opaque Data.....	81
Get Opaque Data.....	84
Delete Opaque Data.....	86
Subscriber Data Row Commands.....	88
Set Row.....	89
Get Row.....	93
Delete Row.....	96
Subscriber Data Row Field Commands.....	98
Get Row Field.....	99
Get Row Field Value.....	101
Update Row Field.....	105
Delete Row Field.....	108
Special Operations.....	110
Reset Quota.....	111

Chapter 7: Pool Provisioning.....114

Pool Profile Commands.....	115
Create Pool.....	115
Get Pool.....	117
Update Pool.....	119
Delete Pool.....	121
Pool Profile Field Commands.....	122
Add Field Value.....	123
Get Field.....	125
Get Field Value.....	127
Update Field.....	129
Update Multiple Fields.....	130
Delete Field.....	132
Delete Field Value.....	134
Pool Opaque Data Commands.....	136
Set Opaque Data.....	136
Get Opaque Data.....	139
Delete Opaque Data.....	142
Additional Pool Commands.....	143
Add Member to Pool.....	144
Remove Member from Pool.....	146

Get Pool Members.....	148
Get PoolID.....	151
Appendix A: REST Interface System Variables.....	153
REST Interface System Variables.....	154
Appendix B: Legacy SPR Compatibility Mode.....	155
Get Row Response Format.....	156
Glossary.....	158

List of Figures

Figure 1: UDR High Level Architecture.....	19
Figure 2: Provisioning Interface Overview.....	21
Figure 3: Summary of Subscriber Commands.....	37
Figure 4: Summary of Pool Commands.....	38
Figure 5: Data Model.....	42

List of Tables

Table 1: Admonishments.....	14
Table 2: SSL X.509 Certificate and Key PEM-encoded Files.....	23
Table 3: SSLv3 Supported Cipher Suites.....	24
Table 4: HTTP(S) Status Codes.....	30
Table 5: Error Codes.....	31
Table 6: Subscriber Profile Attributes.....	43
Table 7: Quota Attributes.....	44
Table 8: State Attributes.....	46
Table 9: Dynamic Quota Attributes.....	46
Table 10: Pool Profile Attributes.....	48
Table 11: Pool Quota Attributes.....	49
Table 12: Pool State Attributes.....	51
Table 13: Pool Dynamic Quota Attributes.....	52
Table 14: Summary of Subscriber Profile Commands.....	55
Table 15: Request Variable Definitions: Create Profile.....	56
Table 16: Request Variable Definitions: Get Profile.....	59
Table 17: Response Variable Definitions: Get Profile.....	60
Table 18: Request Variable Definitions: Update Profile.....	62
Table 19: Request Variable Definitions: Update Profile.....	62
Table 20: Request Variable Definitions: Delete Profile.....	64
Table 21: Summary of Subscriber Profile Field Commands.....	65
Table 22: Request Variable Definitions: Add Field Value.....	66

Table 23: Request Variable Definitions: Get Field.....	68
Table 24: Response Variable Definitions: Get Field.....	69
Table 25: Request Variable Definitions: Get Field Value.....	71
Table 26: Response Variable Definitions: Get Field Value.....	72
Table 27: Request Variable Definitions: Update Field.....	73
Table 28: Request Variable Definitions: Update Multiple Fields.....	75
Table 29: Request Variable Definitions: Delete Field.....	77
Table 30: Request Variable Definitions: Delete Field Value.....	79
Table 31: Summary of Subscriber Opaque Data Commands.....	80
Table 32: Request Variable Definitions: Set Opaque Data.....	81
Table 33: Request Variable Definitions: Set Opaque Data.....	82
Table 34: Request Variable Definitions: Get Opaque Data.....	84
Table 35: Response Variable Definitions: Get Opaque Data.....	85
Table 36: Request Variable Definitions: Delete Opaque Data.....	87
Table 37: Summary of Subscriber Data Row Commands.....	88
Table 38: Request Variable Definitions: Set Row Value.....	89
Table 39: Request Variable Definitions: Set Row Value.....	90
Table 40: Request Variable Definitions: Get Row.....	93
Table 41: Response Variable Definitions: Get Row.....	94
Table 42: Request Variable Definitions: Delete Row.....	96
Table 43: Summary of Subscriber Data Row Field Commands.....	98
Table 44: Request Variable Definitions: Get Row Field.....	99
Table 45: Response Variable Definitions: Get Row Field.....	100
Table 46: Request Variable Definitions: Get Row Field Value.....	102
Table 47: Response Variable Definitions: Get Row Field Value.....	103

Table 48: Request Variable Definitions: Update Row Field.....	106
Table 49: Request Variable Definitions: Delete Row Field.....	109
Table 50: Summary of Subscriber Special Operation Commands.....	110
Table 51: Request Variable Definitions: Reset Quota Value.....	111
Table 52: Summary of Pool Profile Commands.....	115
Table 53: Request Variable Definitions: Create Pool.....	116
Table 54: Request Variable Definitions: Get Pool.....	118
Table 55: Response Variable Definitions: Get Pool.....	118
Table 56: Request Variable Definitions Update Pool.....	120
Table 57: Request Variable Definitions: Update Pool.....	120
Table 58: Request Variable Definition: Delete Pool.....	121
Table 59: Summary of Pool Profile Field Commands.....	122
Table 60: Request Variable Definitions: Add Field Value.....	124
Table 61: Request Variable Definitions: Get Field.....	125
Table 62: Response Variable Definitions: Get Field.....	126
Table 63: Request Variable Definitions: Get Field Value.....	127
Table 64: Response Variable Definitions: Get Field Value.....	128
Table 65: Request Variable Definitions: Update Field.....	129
Table 66: Request Variable Definitions: Update Multiple Fields.....	131
Table 67: Request Variable Definitions: Delete Field.....	133
Table 68: Request Variable Definitions: Delete Field Value.....	134
Table 69: Summary of Pool Opaque Data Commands.....	136
Table 70: Request Variable Definitions: Set Opaque Data.....	136
Table 71: Response Variable Definitions: Set Opaque Data.....	137
Table 72: Request Variable Definitions: Delete Opaque Data.....	142

Table 73: Summary of Additional Pool Commands.....	143
Table 74: Request Variable Definitions: Add Member to Pool.....	144
Table 75: Request Variable Definitions: Remove Member from Pool.....	147
Table 76: Request Variable Definition: Get Pool Members.....	148
Table 77: Response Variable Definitions: Get Pool Members.....	149
Table 78: Request Variable Definitions: Get PoolID.....	151
Table 79: Response Variable Definition: Get PoolID.....	151
Table 80: REST Interface System Variables.....	154

Chapter 1

Introduction

Topics:

- *Overview.....13*
- *Scope and Audience.....13*
- *Manual Organization.....13*
- *Documentation Admonishments.....13*
- *Related Publications.....14*
- *Locate Product Documentation on the Oracle Technology Network Site.....14*
- *Customer Training.....15*
- *My Oracle Support (MOS).....15*
- *Emergency Response.....15*

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 Representational State Transfer (REST) Provisioning 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 REST interfaces, an external provisioning system supplied and maintained by the network operator may add, change, delete or retrieve subscriber/pool information in the UDR database.

Scope and Audience

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

Manual Organization





This document is organized into the following chapters:

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

Documentation Admonishments

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

Table 1: Admonishments

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

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

Customer Training

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

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

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

www.oracle.com/education/contacts

My Oracle Support (MOS)

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

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

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

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

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

Emergency Response

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

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

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

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

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

Chapter 2

System Architecture

Topics:

- *Overview.....18*
- *Provisioning Interface.....19*
- *REST Application Server (RAS).....20*
- *Provisioning Clients.....21*
- *Security.....22*
- *Multiple Connections.....24*
- *Request Queue Management.....25*
- *Database Transactions.....25*
- *Connection Management.....26*
- *Behavior During Low Free System Memory.....27*

This chapter describes the Oracle Communications User Data Repository (UDR) architecture.

Overview

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 the UDR as specified in 3GPP UDC information model. UDR facilitates the share and the provisioning of user related data throughout services of 3GPP system. Several Applications Front Ends, such as: one or more PCRF/HSS/HLR/AuCFEs can be served by UDR.

The data stored in UDR can be permanent or 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.).

The 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.

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.

The 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.

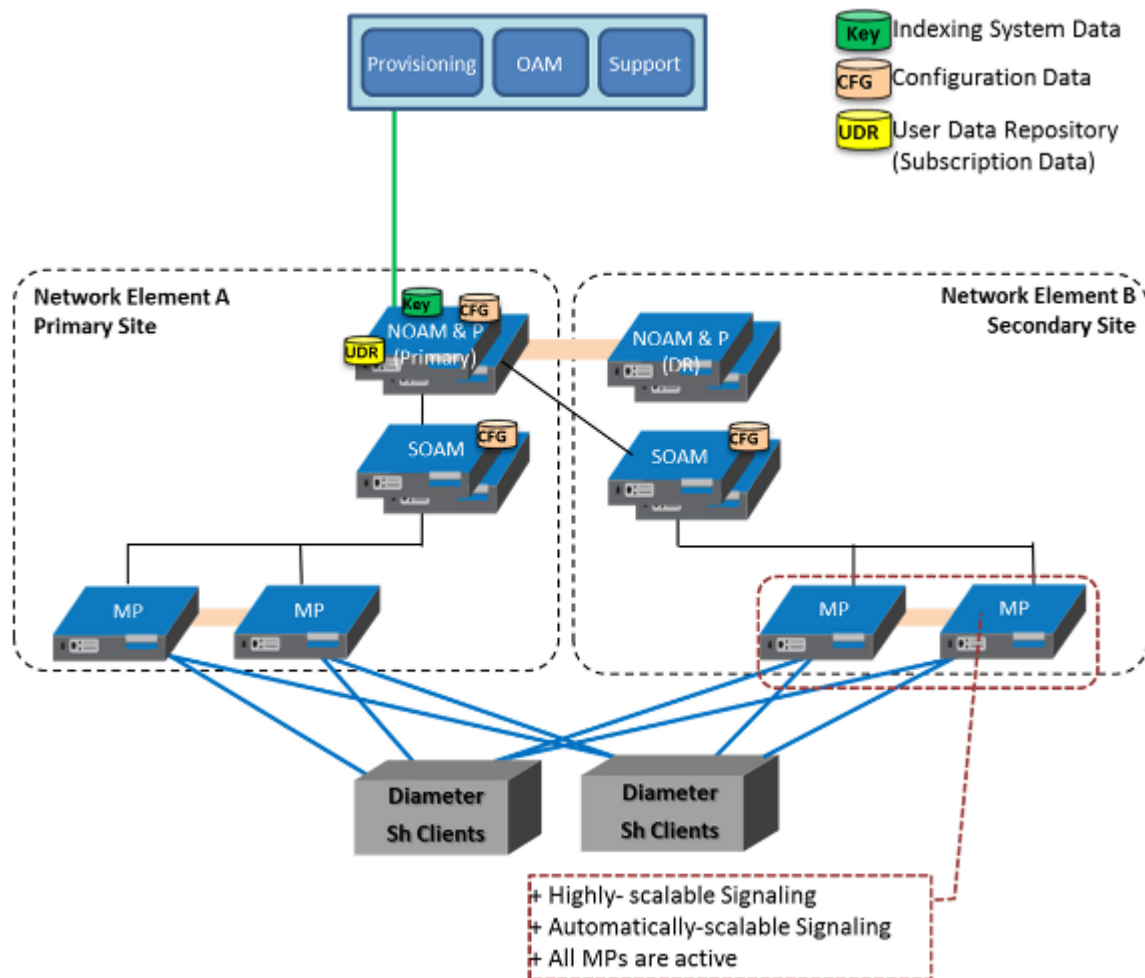


Figure 1: UDR High Level Architecture

Provisioning Interface

The REST provisioning interface provides the following data manipulation commands:

Subscriber:

- Subscriber Profile create/retrieve/modify/delete
- Subscriber Profile field add/retrieve/modify/delete
- Subscriber opaque data create/retrieve/modify/delete
 - Quota, State, and Dynamic Quota

- Reset of Subscriber Quota opaque data

Pool:

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

REST Application Server (RAS)

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

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

Figure 2: Provisioning Interface Overview illustrates the components that interact with the RAS Interface.

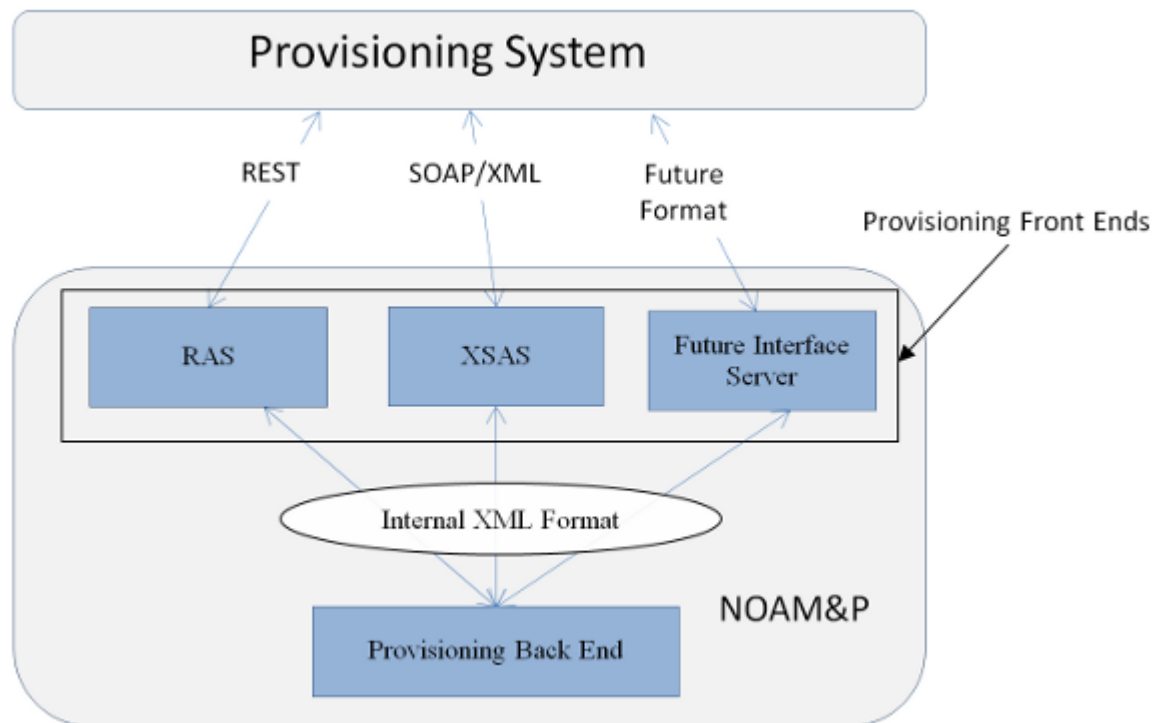


Figure 2: Provisioning Interface Overview

Provisioning Clients

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

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

Provisioning clients need to re-establish connections with the RAS using the Primary UDR's VIP upon switchover from the Primary's Active to its Standby UDR server. Provisioning clients also need to redirect connections to the Secondary's VIP upon switchover from the Primary UDR site to the Disaster Recovery UDR site.

Provisioning clients must run a timeout for the response to a request, in case a response is not sent. If no response is received, a client should drop the connection and re-establish it before trying again.

Provisioning clients are expected to re-send requests that resulted in a temporary error, or for which no response was received.

Security

The following forms of security are provided for securing connections between the REST Interface and provisioning clients in an unsecure/untrusted network:

- Client Server IP Address White List
- Secure Connections using SSLv3

Client Server IP Address White List

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

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

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

Secure Connection using SSLv3

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

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

The following capabilities of SSL address several fundamental concerns about communication over TCP/IP networks:

- **SSL server authentication**

allows a client application to confirm the identity of the server application. The client application through SSL uses standard public-key cryptography to verify that the server's certificate and public key are valid and has been signed by a trusted certificate authority (CA) that is known to the client application.

- **SSL client authentication**

allows a server application to confirm the identity of the client application. The server application through SSL uses standard public-key cryptography to verify that the client's certificate and public key are valid and has been signed by a trusted certificate authority (CA) that is known to the server application.

- **An encrypted SSL connection**

requires all information being sent between the client and server application to be encrypted. The sending application is responsible for encrypting the data and the receiving application is responsible for decrypting the data. In addition to encrypting the data, SSL provides message integrity. Message integrity provides a means to determine if the data has been tampered with since it was sent by the partner application.

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

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

SSL Certificates and Public/Private Key Pairs

SSL-enabled connections require SSL certificates. Certificates rely on asymmetric encryption (or public-key encryption) algorithms that have two encryption keys (a public key and a private key). A certificate owner can show the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted with this public key can be decrypted only using the corresponding, matching private key, which is held by the owner of the certificate.

Tekelec issues Privacy Enhanced Mail (PEM)-encoded SSL X.509v3 certificates and encryption keys to the REST Server and provisioning clients needing to establish a SSL-enabled connection with the REST Server. These files can be found on the UDR server under `/usr/TKLC/udr/ssl`. These files should be copied to the server running the provisioning client.

Table 2: SSL X.509 Certificate and Key PEM-encoded Files

Certificate and Key PEM-encoded Files	Description
tklcCaCert.pem	TEKELEC self-signed trusted root Certification Authority (CA) X.509v3 certificate.
serverCert.pem	The RAS's X.509v3 certificate and 2,048-bit RSA public key digitally signed by TEKELEC Certification Authority (CA) using SHA-1 message digest algorithm.
serverKey.nopass.pem	The RAS' corresponding, matching 2,048-bit RSA private key without passphrase digitally signed by TEKELEC Certification Authority (CA) using SHA-1 message digest algorithm.
clientCert.pem	Provisioning client's X.509v3 certificate and 2,048-bit RSA public key digitally signed by TEKELEC Certification Authority (CA) using SHA-1 message digest algorithm.
clientKey.nopass.pem	Provisioning client's corresponding, matching 2,048-bit RSA private key without passphrase digitally signed by TEKELEC Certification Authority (CA) using SHA-1 message digest algorithm.

Provisioning clients are required to send an SSL authenticating X.509v3 certificate when requested by the RAS during the secure connection handshake protocol for mutual (two-way) authentication. If the

provisioning client does not submit a certificate that is issued/signed by TEKELEC Certification Authority (CA), it will not be able to establish a secure connection with the RAS.

Supported SSLv3 Cipher Suites

A cipher suite is a set/combination of lower-level algorithms that an SSL-enabled connection uses to do authentication, key exchange, and stream encryption. The following table lists the set of cipher suites that are supported by the RAS to secure an SSL-enabled connection with provisioning clients. The cipher suites are listed and selected for use in the order of key strength, from highest to lowest. This ensures that during the handshake protocol of a SSL-enabled connection, cipher suite negotiation selects the most secure suite possible from the list of cipher suites the client wishes to support, and if necessary, back off to the next most secure, and so on down the list. Note: Cipher suites containing anonymous DH ciphers, low bit-size ciphers (currently those using 64 or 56 bit encryption algorithms but excluding export cipher suites), export-crippled ciphers (including 40 and 56 bits algorithms), or the MD5 hash algorithm are not supported due to their algorithms having known security vulnerabilities.

Table 3: SSLv3 Supported Cipher Suites

Cipher Suite	Key Exchange	Signing/Authentication	Encryption (Bits)	MAC (Hash) Algorithms
AES256-SHA	RSA	RSA	AES (256)	SHA-1
DES-CBC3-SHA	RSA	RSA	3DES(168)	SHA-1
AES128-SHA	RSA	RSA	AES(128)	SHA-1
KRB5-RC4-SHA	KRB5	KRB5	RC4(128)	SHA-1
RC4-SHA	RSA	RSA	RC4(128)	SHA-1
KRB5-DES-CBC3-SHA	KRB5	KRB5	3DES(168)	SHA-1

Multiple Connections

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

In case the client does not want to maintain a connection for more than that request, it should send a Connection header including the connection-token close. If either the client or the server sends the close token in the Connection header, that request becomes the last one for the connection.

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

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

- For example, if the certified maximum provisioning TPS rate is 200 TPS, and the **Maximum REST Connections** (see [REST Interface System Variables](#)) is set to 100, then up to 100 connections may be required in order to achieve 200 TPS. It is not possible to achieve the maximum provisioning TPS rate on a single connection.

Request Queue Management

If multiple clients simultaneously issues requests, each request is queued and processed in the order in which it was received on a per connection basis. The client must wait for a response from one request before issuing another.

Incoming requests, whether multiple requests from a single client or requests from multiple clients, are not prioritized. Multiple requests from a single client are handled on a first-in, first-out basis. Requests are processed in the order in which they are received.

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

Database Transactions

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

ACID Compliance

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

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

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

- 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)

Connection Management

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

Connections Allowed

The configuration variable **Allow REST Provisioning Connections** (see [REST Interface System Variables](#)) controls whether REST interface connections are allowed to the configured port. If this variable is set to **NOT_ALLOWED**, then all existing connections are immediately dropped. Alarm 13000 is raised; any attempts to connect are rejected.

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

Disable Provisioning

When the UDR GUI option to disable provisioning is selected, existing connections remain up, and new connections are allowed. But, any provisioning request that is sent will be rejected with a **SERVICE_UNAVAILABLE** error indicating the service is unavailable.

For an example of a provisioning request/response when provisioning is disabled, see the last example in [Create Profile](#).

Idle Timeout

HTTP connection between Provisioning client and RAS process is handled persistent fashion The configuration variable **REST Interface Idle Timeout** (see [REST Interface System Variables](#)) indicates the time to wait before closing the connection due to inactivity (i.e. no requests are received).

Maximum Simultaneous Connections

The configuration variable **Maximum REST Connections** (see [REST Interface System Variables](#)) defines the maximum number of simultaneous REST Interface client connections.

TCP Port Number

The configuration variable *REST Interface Port* (see [REST Interface System Variables](#)) defines the REST Interface TCP Listening Port

Behavior During Low Free System Memory

If the amount of free system memory available to the database falls below a critical limit, then requests that create or update data may fail with the error MSR4068. Before this happens, memory threshold alarms will be raised indicating the impending behavior if the critical level is reached.

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

HTTP Status Code: 507

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4068">errorText</error>
```

Chapter 3

REST Interface Description

Topics:

- [REST Conventions.....29](#)

UDR provides an Application Programming Interface (API) for programmatic management of subscriber data. This interface supports querying, creation, modification, and deletion of subscriber and pool data.

The API is an XML over HTTP(S) interface that is designed based upon RESTful concepts. This section defines the operations that can be performed using the REST interface.

REST Conventions

The REST interface uses the following RESTful concepts:

- HTTP(s) headers
- HTTP(s) status codes
- Error message representation in the response content for all 4xx and 5xx codes.

HTTP(S) Request Headers

The following HTTP(S) requirements must be followed.

HTTP Version

For non-secure HTTP requests, the client must set the header *Request Version* property to:

```
Request Version : HTTP/1.1
```

For secure HTTPS requests, the client must set the header *Request Version* property to :

```
Request Version : HTTPS SSL v3/TLS v1
```

Accept Header

Set the *Accept* header property to the correct MIME version using the following format:

```
Accept: application/camiant-msr-v1+xml      <- version number is 1 or 2.0
or
Accept: application/camiant-msr-v2.0+xml
or
Accept : */*
or
Accept :application/*
```

The *Accept* header must match the version supported by the client. This is true even for requests that do not expect entity response data so that any error content is accepted.

Operations in UDR support both versions 1 and 2.0.

The UDR response to an incorrect MIME version is a Bad Request, for example, with error code *Invalid Accept: application/camiant-msr-v1+xml*.

Transfer-Encoding Header

If a client wishes to use chunked transfer encoding, then the Transfer-Encoding header must be set to:

```
Transfer-Encoding: chunked
```

Requests with Body Content

Requests, which contain body contents, must set the *Content-Type* header property to :

```
Content-Type: application/camiant-msr-v2.0+xml
```

An XML blob for an entity supplied in body contents must begin with an XML version and encoding element as below:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

HTTP(S) Status Codes and Error Messages

The REST interface uses standard HTTP(S) status codes in the response messages. Any operation in the REST interface that results in an HTTP error response in the 4xx or 5xx range will include response content that includes an error Message entity.

[Table 4: HTTP\(S\) Status Codes](#) provides a list of most common Status Codes that an operation may return under normal operating conditions. A more detailed description of the response status codes are provided in each of the provisioning command descriptions.

Table 4: HTTP(S) Status Codes

Status Code	Description
200 - OK	Indicates the successful completion of request processing.
201 - Created	Used for newly created entities.
204 - No Content	The request completed successfully and no response content body is sent back to the client.
400 - Bad Request	This indicates that there is a problem with how the request is formatted or that the data in the request caused a validation error.
404 - Not Found	Indicates that the client tried to operate on a resource that did not exist.
409 - Conflict	Indicated that the client tried to operate on a resource where the operation was not appropriate for that resource.
4xx - Other	Status codes in the 4xx range that are also client request issues. For example, the client may be calling an operation that is not implemented/available or that is asking for a mime type that is not supported.
500 - Internal Server Error	This error and other errors in the 5xx range indicate server problems.
503 - Service Unavailable	Indicates that the client tried to send a provisioning request when provisioning was disabled.
507 - Insufficient Storage	Indicates that free system memory is low, and the database cannot store any new data.

Besides the HTTP status codes, following additional error codes are provided for the 4xx and 5xx range of Status Codes. Note that the "Description" column is for reference only, it is not included in

the HTTP response. Additional text may be included in the HTTP response in some cases, for some responses.

Table 5: Error Codes

Error Code	Description	Additional Text Included ?
MSR4000	Invalid content request data supplied.	No
MSR4001	Subscriber/pool not found.	No
MSR4002	Subscriber/pool/data field is not defined.	Yes
MSR4003	A key is detected to be already in the system for another subscriber/pool.	No
MSR4004	Unique key not found for subscriber/pool.	No
MSR4005	Field does not support multiple values and value for field already exists.	Yes
MSR4049	Data type is not defined	No
MSR4050	Unknown key, the key provided in the request is invalid.	No
MSR4051	The value provided for the field is invalid.	Yes
MSR4053	Subscriber/pool and field exist but the value provided is incorrect, applies to delete field value.	No
MSR4055	Subscriber is a member of a pool	No
MSR4056	Field is not updatable	No
MSR4057	Request only contains one field to update	No
MSR4058	Data type not found	No
MSR4059	Data row does not exist	No
MSR4060	Number of pool members exceeded	No
MSR4061	Specified pool does not exist	No
MSR4062	Subscriber is not a member of the pool	No
MSR4063	Entity cannot be reset	No
MSR4064	Occurrence constraint violation	No
MSR4065	Field is not set	No
MSR4066	Field value already exists	No
MSR4067	Multiple matching rows found	No
MSR4068	Free system memory is low	No
MSR4098	Provisioning is disabled	No

Error Code	Description	Additional Text Included ?
MSR4099	Unexpected server error has occurred	Yes

This example defines both an error code and additional error text to explain the error.

```
<?xml version="1.0" encoding="UTF-8"?>  
<error code="MSR4051">Field value not valid: Field: 'nextResetTime' Value:  
'100' [MSISDN:9971701913]</error>
```

Note: Within the examples shown in the following sections, the error text associated with the MSRxxxx is not shown as this varies depending on the entity/key/field values used.

Chapter 4

REST Interface Message Definitions

Topics:

- [Message Conventions.....34](#)

This chapter describes the syntax and parameters of XML requests and responses.

Message Conventions

HTTP Method

The POST, PUT, GET, and DELETE HTTP methods are used on the REST interface.

Base URI

The base URI ({baseURI}) that is the prefix for the documented URIs uses the following syntax:

```
http(s)://{DNS Name or IP address}<IP Port>/rs
```

The curly brackets denote replacement variables and are not part of the actual operation syntax. Any replacement variable data that contains any special characters must be encoded. The value in the curly brackets can be determined by how the UDR is installed in the network.

For example, if the UDR is installed with the DNS name *udr.oracle.com* on a system with IP address *1.2.3.4*, with a port number of *8787*, the base URI could be either:

```
http://udr.oracle.com:8787/rs
```

or

```
https://1.2.3.4:8787/rs
```

REST URL

The REST interface uses the following XML conventions in the REST command URL.

Subscriber or Pool in URL

Keyword *sub* indicates subscriber operations and *pool* indicates pool operations

For example, for a subscriber:

```
DELETE {baseURI}/msr/sub/IMSI/302370123456789/field/inputVolume
```

And for a pool:

```
DELETE {baseURI}/msr/pool/100000/field/custom12
```

Opaque Data Operations in URL

For opaque data *operations* the keyword *data* is used. The data type indicated in the URL can be any valid opaque or transparent data type.

Note: opaque data operations can be performed on entities defined as opaque or transparent. An opaque data operation works on the entire XML blob creating/getting/deleting it in its entirety.

For example when deleting the Quota data for a subscriber:

```
DELETE {baseURI}/msr/sub/IMSI/302370123456789/data/quota
```

Field in URL

For field operations on the subscriber Profile, the keyword *field* is used. A Field in the URL can be any field, including key fields.

For example, to delete the outputVolume field for a subscriber:

```
DELETE {baseURI}/msr/sub/IMSI/302370123456789/field/outputVolume
```

Transparent Data Row Operations in URL

For transparent data row based operations the keyword *data* is also used. The data type indicated in the URL can be any valid transparent data type which is row based. The data row name is also supplied.

For example when deleting a row in Quota data for a subscriber:

```
DELETE {baseURI}/msr/sub/IMSI/302370123456789/data/quota/10GBMonth
```

Transparent Data Row Field Operations in URL

For transparent data row field based operations the keyword *data* is also used. The data type indicated in the URL can be any valid transparent data type which is row based. The data row name and field name are also supplied.

For example when deleting a row field in Quota data for a subscriber:

```
DELETE {baseURI}/msr/sub/IMSI/302370123456789/data/quota/10GBMonth/totalVolume
```

Case Sensitivity

The URL *constructs* that REST requests are made up of (i.e *msr* , *sub* , *pool* , *field* , *data* , *multipleFields*) 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:

```
POST {baseURI}/msr/sub/MSISDN/33123654862/field/Entitlement/DayPass
```

But the following is NOT:

```
POST {baseURI}/msr/Sub/MSISDN/33123654862/field/Entitlement/DayPass
```

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

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

Examples:

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

List of Messages

[Figure 3: Summary of Subscriber Commands](#) provides a list of operations/messages for subscriber data. Each row of the table represents a command. Parameters required for each command are shown in a colored column. Any blank/uncolored column represents unused parameter for the corresponding command.

Operation Data	Command (Method)	URL	Main object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	Field Name	FieldValue	Additional input											
Subscriber Profile	Create Profile (POST)	[Base URL]/msr	sub	{keyName}, MSISDN, NAI, IMSI, accountId	{keyValue}						Request Content											
	Get Profile (GET)																					
	Update Profile (PUT)										Request Content											
	Delete Profile (DELETE)																					
Subscriber Field	Add Field Value (POST)					field/multipleFields		{field Name}	{fieldValue}													
	Get Field (GET)																					
	Get Field Value (GET)							{field Name}	{fieldValue}													
	Update Field (PUT)																					
	Delete Field (DELETE)																					
	Delete Field Value (DELETE)								{field Name}	{fieldValue}												
Subscriber Opaque Data	Set Opaque Data (PUT)					data		{opaqueDataType}							Request Content							
	Get Opaque Data (GET)																					
	Delete Opaque Data (DELETE)																					
Subscriber Data Row	Set Row (PUT)									data		{opaqueDataType}							Request Content			
	Get Row (GET)																					
	Delete Row (DELETE)																					
Subscriber Data Row Field	Get Row Field (GET)	data		{opaqueDataType}																		
	Get Row Field Value (GET)																					
	Update Row Field (PUT)																			{field Name}	{FieldValue}	Request Content
	Delete Row Field (DELETE)																					

Figure 3: Summary of Subscriber Commands

Figure 4: Summary of Pool Commands provides a list of operations/messages for pool data. Each row of the table represents a command. Parameters required for each command are shown in a colored column. Any blank/uncolored column represents an unused parameter for the corresponding command.

Operation Data	Command (Method)	URL	Main object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	FieldName	FieldValue	Additional input				
Pool Profile	Create Pool (POST)	[Base URL]/msr	pool		{keyValue}						Request Content				
	Get Pool (GET)														
	Update Pool (PUT)										Request Content				
	Delete Pool (DELETE)														
Pool Profile Field	Add Field Value(POST)					field/ multipleFields						{fieldName}	{fieldName}	{fieldName}	
	Get Field (GET)														
	Get Field Value (GET)														
	Update Field (PUT)												{fieldName}	{fieldName}	
	Delete Field (DELETE)														
	Delete Field Value (DELETE)												{fieldName}	{fieldName}	
Pool Opaque Data	Set Opaque Data (PUT)					data					{opaqueData type}			Request Content	
	Get Opaque Data (GET)														
	Delete Opaque Data (DELETE)														

Figure 4: Summary of Pool Commands

Chapter 5

UDR Data Model

Topics:

- [Overview.....40](#)
- [Subscriber Data.....42](#)
- [Pool Data.....48](#)

This chapter describes the subscriber data and their defined entities and attributes.

Overview

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

The subscriber-related data includes:

- **Profile/Subscriber Data:** pre-provisioned information that describes the capabilities of each subscriber. This data is typically written by the customer's OSS system (via a provisioning interface) and referenced by the PCRF (via the Sh interface).
- **Quota:** information that represents the subscriber's use of managed resources (quota, pass, top-up, rollover). 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.

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. The only exception to this is when a mandatory field has a default value. If the field is deleted, it will be set to the default value.

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

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

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

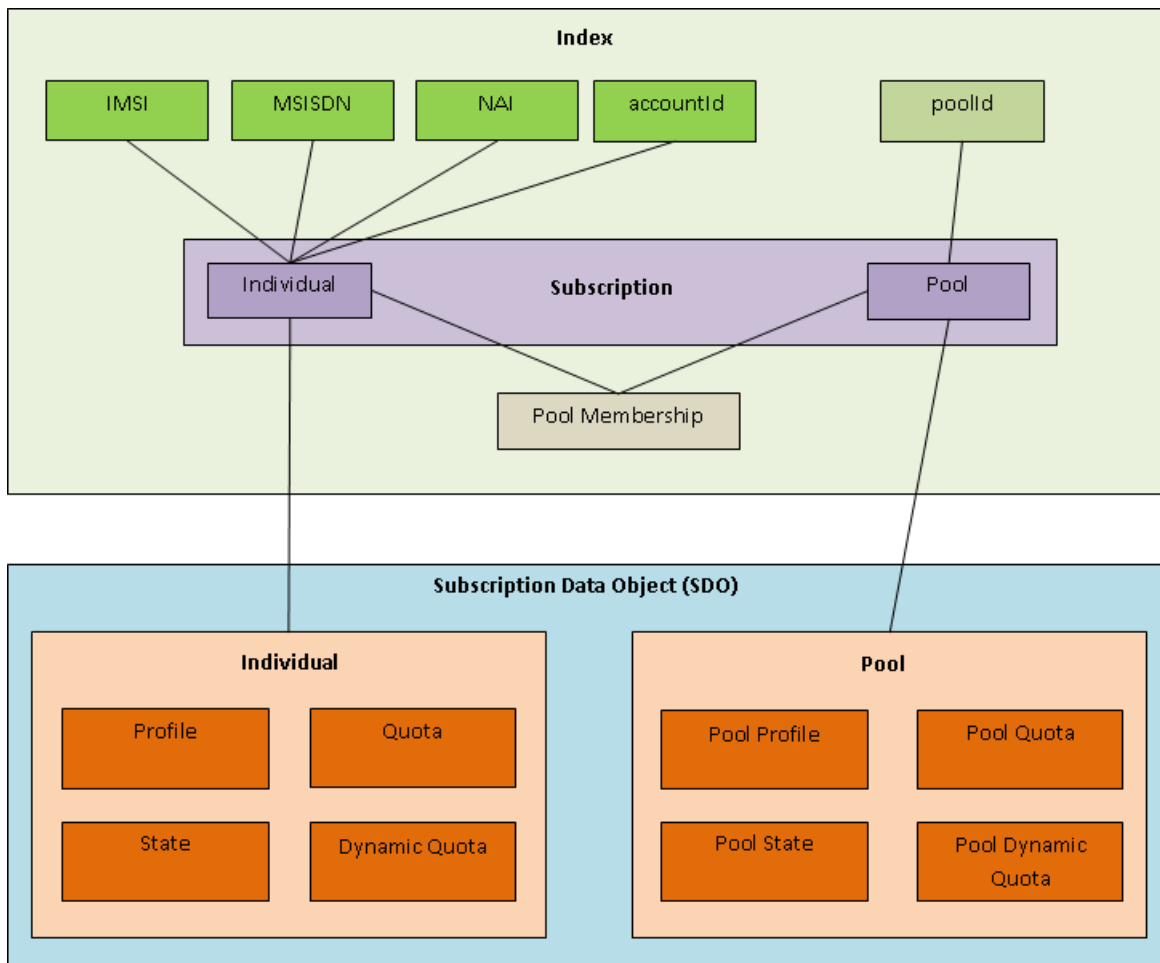


Figure 5: Data Model

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.

Table 6: Subscriber Profile Attributes

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.
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.

Name (xml tag)	Type	Description
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 UDR, only v3 of Quota is supported.

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

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

Table 7: Quota Attributes

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 identify a quota within a subscriber profile.	1/2/3
time	String	Empty string ""	This element tracks the time-based resource consumption for a Quota.	1/2/3
totalVolume	String	"0"	This element tracks the bandwidth volume-based resource consumption for a Quota.	1/2/3
inputVolume	String	"0"	This element tracks the upstream bandwidth volume-based resource consumption for a Quota.	1/2/3
outputVolume	String	"0"	This element tracks the downstream bandwidth volume-based resource consumption for a Quota.	1/2/3
serviceSpecific	String	Empty string ""	This element tracks service-specific resource consumption for a Quota.	1/2/3

Name (xml tag)	Type	Default Value	Description	Quota Versions
nextResetTime	String	Empty string ""	When set, it 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"	Granted Total Volume, will represent the granted total volume of all the subscribers in the pool, in case of pool quota. In case of individual quota, it 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: State Attributes

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 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.

Name (xml tag)	Type		Description
InstanceId	String		A unique identifier to identify this instance of a dynamic quota object.
Priority	String		An integer represented as a string. This number allows service providers to specify when one pass or top-up should be used before another pass or top-up.
InitialTime	String		An integer represented as a string. The number of seconds initially granted for the pass/top-up.
InitialTotalVolume	String		An integer represented as a string. The number of bytes of total volume initially granted for the pass/top-up.
InitialInputVolume	String		An integer represented as a string. The number of bytes of input volume initially granted for the pass/top-up.
InitialOutputVolume	String		An integer represented as a string. The number of bytes of output volume initially granted for the pass/top-up.
InitialServiceSpecific	String		An integer represented as a string. The number of service specific units initially granted for the pass/top-up.
activationdatetime	String		The date/time after which the pass or top-up may be active. See below for date/time format
expirationdatetime	String		The date/time after which the pass or top-up is considered to be exhausted See below for date/time format
purchasedatetime	String		The date/time when a pass was purchased See below for date/time format
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 only included in the record if they are specified when the record is created/updated.

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

Table 10: Pool Profile Attributes

Name (xml tag)	Type	Description
pool	---	Sequence (multiplicity = 1)
PoolID	String	Pool identifier (1-22 numeric digits, minimum value 100000)
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 subscriber's profile..
Tier	String	Pool's Tier.
Custom1	String	Field used to store customer-specific data.
Custom2	String	Field used to store customer-specific data.
Custom3	String	Field used to store customer-specific data.
Custom4	String	Field used to store customer-specific data.

Name (xml tag)	Type	Description
Custom5	String	Field used to store customer-specific data.
Custom6	String	Field used to store customer-specific data.
Custom7	String	Field used to store customer-specific data.
Custom8	String	Field used to store customer-specific data.
Custom9	String	Field used to store customer-specific data.
Custom10	String	Field used to store customer-specific data.
Custom11	String	Field used to store customer-specific data.
Custom12	String	Field used to store customer-specific data.
Custom13	String	Field used to store customer-specific data.
Custom14	String	Field used to store customer-specific data.
Custom15	String	Field used to store customer-specific data.
Custom16	String	Field used to store customer-specific data.
Custom17	String	Field used to store customer-specific data.
Custom18	String	Field used to store customer-specific data.
Custom19	String	Field used to store customer-specific data.
Custom20	String	Field 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 may be present based on the version number value of the entity being accessed. In UDR, there is only one version number of 1.

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 Attributes

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)

Name (xml tag)	Type	Size	Default Value	Description
cid	String	255		Internal identifier used to identify a quota within a subscriber profile.
time	String	255	Empty string ""	This element tracks the time-based resource consumption for a Quota.
totalVolume	String	255	"0"	This element tracks the bandwidth volume-based resource consumption for a Quota.
inputVolume	String	255	"0"	This element tracks the upstream bandwidth volume-based resource consumption for a Quota.
outputVolume	String	255	"0"	This element tracks the downstream bandwidth volume-based resource consumption for a Quota.
serviceSpecific	String	255	Empty string ""	This element 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"	Granted Total Volume, will represent the granted total volume of all the subscribers in the pool, in case of pool quota. In case of individual quota, it 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: Pool State Attributes

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 may 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 consists of the following sequence of attributes:

Table 13: Pool Dynamic Quota Attributes

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

Note: Date/Timestamp format is: CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm]

where:

- - = years before 0001

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

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

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

Chapter 6

Subscriber Provisioning

Topics:

- *Subscriber Profile Commands.....55*
- *Subscriber Profile Field Commands.....65*
- *Subscriber Opaque Data Commands.....80*
- *Subscriber Data Row Commands.....88*
- *Subscriber Data Row Field Commands.....98*
- *Special Operations.....110*

Subscriber data identifies the attributes associated with the user, including the level of services. This chapter describes the subscriber provisioning commands and their attributes (parameters).

Subscriber Profile Commands

Table 14: Summary of Subscriber Profile Commands

Command	Description	Key(s)	Command Syntax
Create Profile	Create a new subscriber/subscriber Profile	MSISDN, NAI, IMSI, AccountID	POST {baseURI}/msr/sub
Get Profile	Get subscriber Profile data		GET {baseURI}/msr/sub/keyName/keyValue
Update Profile	Replace an existing subscriber Profile		PUT {baseURI}/msr/sub/keyName/keyValue
Delete Profile	Delete all subscriber Profile data and all opaque data associated with the subscriber		DELETE {baseURI}/msr/sub/keyName/keyValue

Create Profile

Description

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

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

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

Prerequisites

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

Request URL

POST {baseURI}/msr/sub

Request Content

A <subscriber> element that contains a <field> element for every field-value pair defined for the new subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="keyName1">keyValue1</field>
  [
    <field name="keyName2">keyValue2</field>
```

```

:
<field name="keyNameN">keyValueN</field>
]
[
<field name="fieldName1">fieldValue1</field>
<field name="fieldName2">fieldValue2</field>
:
<field name="fieldNameN">fieldValueN</field>
]
</subscriber>

```

Table 15: Request Variable Definitions: Create Profile

Variable	Definition	Value
keyNameX	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValueX	Corresponding key field value assigned to <i>keyNameX</i>	
fieldNameX	A user defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to <i>fieldNameX</i>	

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

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

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Successfully created
400	MSR4000	Invalid content request data supplied
400	MSR4003	A key is detected to be already in the system for another subscriber
400	MSR4004	The field list does not contain at least one unique key
400	MSR4051	Invalid value for a field
400	MSR4064	Occurrence constraint violation
404	MSR4002	Subscriber field is not defined

Examples

Request 1

A subscriber is created, with *AccountId*, *MSISDN* and *IMSI* keys. The *BillingDay*, *Tier*, *Entitlement*, and *Custom15* fields are set.

Request URL: **POST** {baseURI}/msr/sub

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="AccountId">10404723525</field>
  <field name="MSISDN">33123654862</field>
  <field name="IMSI">184569547984229</field>
  <field name="BillingDay">1</field>
  <field name="Tier"></field>
  <field name="Entitlement">DayPass,DayPassPlus</field>
  <field name="Custom15">allocate</field>
</subscriber>
```

Response 1

The request is successful, and the subscriber was created.

HTTP Status Code: 201

Response Content: None.

Request 2

A subscriber is created, with *MSISDN* and *IMSI* keys. The *BillingDay* and *location* fields are set. *location* is not a valid field name for a subscriber.

Request URL: **POST** {baseURI}/msr/sub

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="MSISDN">5141234567</field>
  <field name="IMSI">184126781623863</field>
  <field name="BillingDay">2</field>
  <field name="location">Montreal</field>
</subscriber>
```

Response 2

The request fails. The error code indicates the field name is not valid.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4002">errorText</error>
```

Request 3

A subscriber is created, with *MSISDN* and *IMSI* keys. The *BillingDay* and *Entitlement* fields are set. A subscriber already exists with the given *IMSI*.

Request URL: **POST** {baseURI}/msr/sub

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="MSISDN">5141112223334</field>
  <field name="IMSI">184126781612121</field>
  <field name="BillingDay">2</field>
  <field name="Entitlement">DayPass</field>
  <field name="Entitlement">DayPassPlus</field>
</subscriber>
```

Response 3

The request fails. The error code indicates the key already exists.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4003">errorText</error>
```

Request 4

A subscriber is created. The *BillingDay* and *Entitlement* fields are set . No key values are supplied.

Request URL: **POST** {baseURI}/msr/sub

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="BillingDay">2</field>
  <field name="Entitlement">DayPass</field>
</subscriber>
```

Response 4

The request fails because no key values were supplied.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4004">errorText</error>
```

Request 5

A subscriber is created, with *MSISDN* and *IMSI* keys. The *BillingDay* and *Custom15* fields are set. *Provisioning has been disabled.*

Request URL: **POST** {baseURI}/msr/sub

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="MSISDN">33123654862</field>
  <field name="IMSI">184569547984229</field>
```

```
<field name="BillingDay">1</field>
<field name="Custom15">allocate</field>
</subscriber>
```

Response 5

The request fails because provisioning has been disabled.

HTTP Status Code: 503

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4098">errorText</error>
```

Get Profile**Description**

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

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

Prerequisites

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

Request URL

GET {baseURI}/msr/sub/*keyName*/*keyValue*

Table 16: Request Variable Definitions: Get Profile

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

Request Content

None

Response Content

A <subscriber> element that contains a <field> element for every field-value pair defined for the subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="keyName1">keyValue1</field>
  [
    <field name="keyName2">keyValue2</field>
    :
    <field name="keyNameN">keyValueN</field>
  ]
  [
    <field name="fieldName1">fieldValue1</field>
    <field name="fieldName2">fieldValue2</field>
    :
    <field name="fieldNameN">fieldValueN</field>
  ]
</subscriber>
```

Table 17: Response Variable Definitions: Get Profile

Variable	Definition	Values
keyNameX	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValueX	Corresponding key field value assigned to <i>keyNameX</i>	
fieldNameX	A user-defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to <i>fieldNameX</i>	

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

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Successfully located the subscriber
400	MSR4051	Invalid value for a field
404	MSR4001	Could not find the subscriber by key

Examples**Request 1**

The subscriber with the given AccountId is retrieved. The subscriber exists.

Request URL: **GET** {baseURI}/msr/sub/accountId/10404723525

Request Content: None

Response 1

The request is successful, and the subscriber was retrieved.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="accountId">10404723525</field>
  <field name="MSISDN">33123654862</field>
  <field name="IMSI">184569547984229</field>
  <field name="BillingDay">1</field>
  <field name="Tier"></field>
  <field name="Entitlement">DayPass</field>
</subscriber>
```

Request 2

The subscriber with the given IMSI is retrieved. The subscriber does NOT exist.

Request URL: **GET** {baseURI}/msr/sub/IMSI/184126781623863

Request Content: None

Response 2

The request fails. The error code indicates the subscriber does not exist.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4001">errorText</error>
```

Update Profile**Description**

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.

Note: The key value specified by *keyName* and *keyValue* must be present in the request content.

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

Prerequisites

A subscriber with a key of the *keyName/keyValue* must be present in the request content.

Request URL

PUT {baseURI}/msr/sub/*keyName/keyValue*

Table 18: Request Variable Definitions: Update Profile

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

Request Content

A <subscriber> element that contains a <field> element for every field-value pair defined for the new subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="keyName1">keyValue1</field>
  [
    <field name="keyName2">keyValue2</field>
    :
    <field name="keyNameN">keyValueN</field>
  ]
  [
    <field name="fieldName1">fieldValue1</field>
    <field name="fieldName2">fieldValue2</field>
    :
    <field name="fieldNameN">fieldValueN</field>
  ]
</subscriber>
```

Table 19: Request Variable Definitions: Update Profile

Variable	Definition	Values
keyNameX	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValueX	Corresponding key field value assigned to <i>keyNameX</i>	
fieldNameX	A user defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to <i>fieldNameX</i>	

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

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

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	The subscriber data was replaced successfully
400	MSR4000	Invalid content request data supplied
400	MSR4003	A key is detected to be already in the system for another subscriber
400	MSR4004	The field list does not contain at least one unique key
400	MSR4051	Invalid value for a field
400	MSR4064	Occurrence constraint violation
404	MSR4001	Could not find the subscriber by key
404	MSR4002	Subscriber field is not defined

Example

Request 1

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

Request URL: **PUT {baseURI}/msr/sub/MSISDN/33123654862**

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="accountId">10404723525</field>
  <field name="IMSI">184569547984229</field>
  <field name="MSISDN">33123654862</field>
  <field name="BillingDay">12</field>
  <field name="Tier"></field>
  <field name="Entitlement">DayPass,DayPassPlus</field>
</subscriber>
```

Response 1

The request is successful, and the subscriber was updated.

HTTP Status Code: 204

Response Content: None

Delete Profile

Description

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 URL

DELETE {baseURI}/msr/sub/*keyName/keyValue*

Table 20: Request Variable Definitions: Delete Profile

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

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	The subscriber was successfully deleted
404	MSR4001	Could not find the subscriber by key
409	MSR4055	Cannot delete, subscriber belongs to a pool

Examples**Request 1**

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

Request URL: **DELETE** {baseURI}/msr/sub/MSISDN/33123654862

Request Content: None

Response 1

The request is successful.

HTTP Status Code: 204

Response Content: None.

Request 2

The subscriber with the given NAI is deleted. The subscriber exists. The subscriber is a member of a pool.

Request URL: **DELETE** {baseURI}/msr/sub/NAI/mum@foo.com

Request Content: None

Response 2

The request fails because the subscriber is a member of a pool.

HTTP Status Code: 409

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4055">errorText</error>
```

Subscriber Profile Field Commands

Table 21: Summary of Subscriber Profile Field Commands

Command	Description	Key(s)	Command Syntax
Add Field Value	Adds value to the specified field. This operation does not affect any pre-existing values for the field	MSISDN, IMSI, NAI or AccountId	POST {baseURI}/msr/sub/ <i>keyName/keyValue/ field/fieldName/fieldValue</i>
Get Field	Retrieve the value(s) for the specified field		GET {baseURI}/msr/sub/ <i>keyName/keyValue/field/fieldname</i>
Get Field Value	Retrieve the single value for the specified field (if set as specified)		GET {baseURI}/msr/sub/ <i>keyName/keyValue/ field/fieldName/fieldValue</i>
Update Field Value	Updates field to the specified value		PUT {baseURI}/msr/sub/ <i>keyName/keyValue/ field/fieldName/fieldValue</i>
Update Multiple Fields	Update multiple fields to the specified values		PUT {baseURI}/msr/sub/ <i>keyName/keyValue/ multipleFields/fieldName1/fieldValue1/ fieldName2/fieldValue2/...</i>
Delete Field	Delete all the values for the specified field		DELETE {baseURI}/msr/sub/ <i>keyName/keyValue/field/fieldname</i>
Delete Field Value	Delete a value for the specified field		DELETE {baseURI}/msr/sub/ <i>keyName/keyValue/ field/fieldName/fieldValue</i>

Add Field Value

Description

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

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

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

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

Note: If the field to which the value is being added does not exist, it will be created.

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

Prerequisites

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

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

The value *fieldValue* being added must NOT already be present in the field.

Request URL

POST {baseURI}/msr/sub/*keyName/keyValue/field/fieldName/fieldValue*

Table 22: Request Variable Definitions: Add Field Value

Variable	Definition	Values
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
fieldName	A user defined field within the subscriber Profile	
fieldValue	Corresponding field value assigned to <i>fieldName</i> Note: For multi-value fields, the value will contain a semicolon separated list of values on a single line. For example, "a;b;c"	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Successfully added field values
400	MSR4005	Field does not support multiple values
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4066	Field value already exists
404	MSR4001	Subscriber is not found
404	MSR4002	Subscriber field is not defined

Examples**Request 1**

A request is made to add the value *DayPass* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* value is not already present in the *Entitlement* field.

Request URL: **POST**

```
{baseURI}/msr/sub/MSISDN/33123654862/field/Entitlement/DayPass
```

Request Content: None

Response 1

The request is successful, and the value was added to the *Entitlement* field.

HTTP Status Code: 200

Response Content: None

Request 2

A request is made to add the values *DayPass* and *HighSpeedData* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* and *HighSpeedData* values are not already present in the *Entitlement* field.

Request URL: **POST**

```
{baseURI}/msr/sub/NAI/dad@op.com/field/Entitlement/DayPass;HighSpeedData
```

Request Content: None

Response 2

The request is successful, and the values were added to the *Entitlement* field.

HTTP Status Code: 200

Response Content: None

Request 3

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

Request URL: **POST** `{baseURI}/msr/sub/NAI/dad@op.com/field/Tier/Gold`

Request Content: None

Response 3

The request fails because the *Tier* field is not a multi-value field.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4005">errorText</error>
```

Get Field

Description

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

Prerequisites

A subscriber with the key of the *keyName/keyValue* supplied must exist. The requested field *fieldName* must be a valid field in the subscriber Profile.

RequestURL

GET `{baseURI}/msr/sub/keyName/keyValue/field/fieldName`

Table 23: Request Variable Definitions: Get Field

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

Request Content

None

Response Content

A **<subscriber>** element that contains a **<field>** element for every field-value pair for the requested field defined for the subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="fieldName">fieldValue1</field>
  [
    <field name="fieldName">fieldValue2</field>
    :
    <field name="fieldName">fieldValueN</field>
  ]
</subscriber>
```

Table 24: Response Variable Definitions: Get Field

Variable	Definition	Values
fieldName	A user defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to <i>fieldName</i>	

Note: for multi-value fields, more than one **<field>** element may be returned. One element per value.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested field exists for subscriber
404	MSR4001	Subscriber is not found
404	MSR4002	Subscriber field is not defined
404	MSR4065	Field is not set

Examples**Request 1**

A request is made to get the *AccountId* field for a subscriber.

Request URL: **GET** {baseURI}/msr/sub/MSISDN/33123654862/field/AccountId

Request Content: None

Response 1

The request is successful, and the requested value is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
```

```
<field name="accountId">10404723525</field>
</subscriber>
```

Request 2

A request is made to get the *Entitlement* field for a subscriber. The Entitlement field is a multi-value field.

Request URL: **GET** {baseURI}/msr/sub/MSISDN/33123654862/field/Entitlement

Request Content: None

Response 2

The request is successful, and the requested value is returned. Two values are set for the multi-value field.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="Entitlement">DayPass</field>
  <field name="Entitlement">HighSpeedData</field>
</subscriber>
```

Request 3

A request is made to get the *custom11* field for a subscriber. The field is valid, but is not set for the subscriber.

Request URL: **GET** {baseURI}/msr/sub/MSISDN/33123654862/field/custom11

Request Content: None

Response 3

The request is successful, and an empty value is returned.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4065">errorText</error>
```

Get Field Value

Description

This operation retrieves the values for the specified field for the subscriber identified by the *keyName* and *keyValue* in the request.

For a request where the presence of multiple values for a multi-value field is requested, a match is only considered to have been made if the requested values form a subset of the values stored in the profile. That is, if all of the values requested exist in the profile, return success, regardless of how many other values may exist in the profile. If any or all of the values are not present as part of the profile, an error is returned.

Note: Note that depending upon the field entered, there may be multiple field-value pairs returned by this operation.

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

Prerequisites

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

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

The requested field must contain the value(s) supplied in the *fieldValue*.

Request URL

GET {baseURI}/msr/sub/*keyName/keyValue/field/fieldName/fieldValue*

Table 25: Request Variable Definitions: Get Field Value

Variable	Definition	Values
keyName	A key field within the subscriber Profile.	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
fieldName	A user defined field within the subscriber Profile	
fieldValue	<p>Corresponding key field value assigned to <i>fieldName</i></p> <p>Note: For multi-value fields, the value will contain a semicolon-separated list of values on a single line, e.g., "a;b;c".</p> <p>Note: The semicolon between the field values may need to be encoded as %3B for certain clients.</p>	

Request Content

None

Response Content

A <subscriber> element that contains a <field> element for every field-value pair requested that matches the value supplied for the existing subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="fieldName1">fieldValue1</field>
  [
    <field name="fieldName2">fieldValue2</field>
    :
    <field name="fieldNameN">fieldValueN</field>
  ]
</subscriber>
```

```
]
</subscriber>
```

Table 26: Response Variable Definitions: Get Field Value

Variable	Definition	Values
fieldNameX	The requested user-defined field within the subscriber Profile	
fieldValueX	Corresponding field value assigned to <i>fieldName</i> Note: for multi-value fields, more than one <field> element maybe returned. One element per value.	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested field exists for subscriber with given value
400	MSR4053	Subscriber and field exist, but value(s) do not match
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Examples

Request 1

A request is made to get the *AccountId* field with the value *10404723525* . The field exists and has the specified value.

Request URL:

```
GET {baseURI}/msr/sub/MSISDN/33123654862/field/AccountId/10404723525
```

Request Content: None

Response 1

The request is successful, and the requested value is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="accountId">10404723525</field>
</subscriber>
```

Request 2

A request is made to get the *Entitlement* field with the values *DayPass* and *HighSpeedData*. The Entitlement field is a multi-value field. The field exists and has the specified values.

Request URL:

GET

{baseURI}/msr/sub/MSISDN/33123654862/field/Entitlement/DayPass;HighSpeedData

Request Content: None

Response 2

The request is successful, and the requested values are returned. Two values are set for the multi-value field.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="Entitlement">DayPass</field>
  <field name="Entitlement">HighSpeedData</field>
</subscriber>
```

Update Field

Description

This operation updates a field to the specified value for the subscriber identified by the specified *keyName* and *keyValue*.

This operation replaces ("sets") the value of the field, which means that any existing values for the field are deleted first. For multi-value fields, all previous values are erased and the new set specified here is inserted. Adding values to a current set is accomplished using Add Field Value.

Note: This command cannot be used to update key values (i.e., IMSI/MSISDN/NAI/AccountId).

Prerequisites

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

The field *fieldName* must all be a valid field in the subscriber Profile.

Request URL

PUT {baseURI}/msr/sub/ *keyName/keyValue/field/fieldName/fieldValue*

Table 27: Request Variable Definitions: Update Field

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

Variable	Definition	Values
fieldValue	<p>Corresponding field value assigned to <i>fieldName</i></p> <p>Note: For multi-value fields, the value can contain a comma separated list of values on a single line. E.g. "a,b,c"</p> <p>Note: The semicolon between the field values may need to be encoded as %3B for certain clients.</p>	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Field(s) were successfully updated
400	MSR4051	The value provided for the field is invalid
400	MSR4056	Field is not updatable
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Examples**Request 1**

A request is made to update the value of the *Tier* field to *Silver* .

Request URL: **PUT** `{baseURI}/msr/sub/MSISDN/33123654862/field/Tier/Silver`

Request Content: None

Response 1

The request is successful, and the *Tier* field was updated.

HTTP Status Code: 201

Response Content: None

Request 2

A request is made to update the *Entitlement* field with the values *DayPass* and *HighSpeedData* . The *Entitlement* field is a multi-value field.

Request URL: **PUT**

`{baseURI}/msr/sub/MSISDN/33123654862/field/Entitlement/DayPass;HighSpeedData`

Request Content:

None

Response 2

The request is successful, and the *Entitlement* field was updated.

HTTP Status Code: 201

Response Content: None

Update Multiple Fields

Description

This operation updates 2 or 3 fields to the specified values for the subscriber identified by the specified *keyName* and *keyValue*.

This operation replaces ("sets") the value of the field, which means that any existing values for the field are deleted first. For multi-value fields, all previous values are erased and the new set specified here is inserted. Adding values to a current set is accomplished using Add Field Value.

This command allows the update of multiple fields in a single command for subscriber data.

ALL fields that can be modified in the "single field" request can also be modified in the "multiple fields" request. Two or three fields can be updated at once. Updating only a single field will result in an error.

All fields are updated at once in the DB. All fields and all values must be valid for the update to be successful. In other words, as soon as one error is detected, processing the request is stopped (and return an error). For example, if the third field fails validation, then none of the fields are updated.

Note: This command cannot be used to update key values (i.e., IMSI/MSISDN/NAI/AccountId).

Prerequisites

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

The field(s) *fieldNameX* must all be valid fields in the subscriber Profile.

Request URL

```
PUT {baseURI}/msr/sub/keyName/keyValue/
multipleFields/fieldName1/fieldValue1/
fieldName2/fieldValue2/[fieldName3/fieldValue3]
```

Table 28: Request Variable Definitions: Update Multiple Fields

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

Variable	Definition	Values
fieldNameX	A user defined field within the subscriber Profile. Note: a field name cannot be for a key value - i.e. <i>IMSI</i> , <i>MSISDN</i> , <i>NAI</i> , or <i>AccountId</i> .	
fieldValueX	Corresponding field value assigned to <i>fieldNameX</i> Note: For multi-value fields, the value will contain a semicolon-separated list of values on a single line, e.g., "a;b;c". Note: The semicolon between the field values may need to be encoded as %3B for certain clients.	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Field(s) were successfully updated
400	MSR4051	The value provided for the field is invalid
400	MSR4056	Field is not updatable
400	MSR4057	Request only contains one field to update
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Example**Request**

A request is made to update the *Entitlement* field to *YearPass*, the *Tier* field to *Silver*, and the *BillingDay* field to *11*.

Request URL: `PUT {baseURI}/msr/sub/MSISDN/33123654862/multipleFields/Entitlement/YearPass/Tier/Silver/BillingDay/11`

Request Content: None

Response

The request is successful, and the *Entitlement*, *Tier*, and *BillingDay* fields were all updated.

HTTP Status Code: 201

Response Content: None

Delete Field

Description

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

If the field is multi-value field, then all values are deleted. Deletion of a field results in the removal of the entire field from the subscriber Profile, that is, the field is not present, not just empty (without a value).

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

Note: This command cannot be used to update key values (i.e., IMSI/MSISDN/NAI/AccountId).

Note: If the field being deleted is mandatory and is defined as having a default value, then the field will not be removed, but will have the default value assigned.

Prerequisites

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

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

Request URL

DELETE {baseURI}/msr/sub/*KeyName/KeyValue/field/fieldName*

Table 29: Request Variable Definitions: Delete Field

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
fieldName	A user defined field within the subscriber Profile. Note: a field name cannot be for a key value , which are <i>IMSI, MSISDN, NAI,or AccountId</i>	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Field was successfully deleted
400	MSR4056	Field is not updatable
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Example

Request 1

A request is made to delete the *Tier* field. The field is a valid subscriber Profile field.

Request URL: **DELETE** {baseURI}/msr/sub/MSISDN/33123654862/Tier

Request Content: None

Response 1

The request is successful, and the field was deleted.

HTTP Status Code: 204

Response Content: None

Delete Field Value

Description

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

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

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

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

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

Prerequisites

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

The field *fieldName* must be a valid field in the subscriber Profile, and set to the value supplied to be removed successfully.

Request URL

DELETE {baseURI}/msr/sub/keyName/keyValue/field/fieldName/fieldValue

Table 30: Request Variable Definitions: Delete Field Value

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
fieldName	A user-defined field within the subscriber Profile. Note: a field name cannot be for a key value , which are <i>IMSI, MSISDN, NAI, or AccountId</i>	
fieldValue	Corresponding field value assigned to <i>fieldName</i> Note: For multi-value fields, the value will contain a semicolon-separated list of values on a single line, e.g., "a;b;c". Note: The semicolon between the field values may need to be encoded as %3B for certain clients.	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested field(s) were successfully deleted
400	MSR4005	Field does not support multiple values
400	MSR4056	Field is not updatable
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Examples

Request 1

A request is made to delete the values *DayPass* and *HighSpeedData* from the *Entitlement* field. The *Entitlement* field is a multi-value field. The field exists and contains the specified values.

Request URL: **DELETE** {baseURI}/msr/sub/MSISDN/33123654862/field/Entitlement/DayPass;HighSpeedData

Request Content: None

Response 1

The request is successful, and the values were deleted from the field.

HTTP Status Code: 204

Response Content: None

Request 2

A request is made to delete the *Tier* field which has the value *Gold*. The *Tier* field is not a multi-value field.

Request URL: **DELETE** {baseURI}/msr/sub/MSISDN/33123654862/field/Tier/Gold

Request Content: None

Response 2

The request fails because the *Tier* field is not a multi-value field.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4005">errorText</error>
```

Subscriber Opaque Data Commands

Note: The following commands perform opaque data *operations*. They can be used on entities defined as either opaque or transparent. The opaque data *operation* operates on the entity at the entire XML blob level. The entire contents of the entity is set/returned/deleted.

Table 31: Summary of Subscriber Opaque Data Commands

Command	Description	Key	Command Syntax
Set Opaque Data	Create/update opaque data of the specified type	MSISDN, IMSI, NAI or AccountId	PUT {baseURI}/msr/sub/KeyName/KeyValue/data/opaqueDataType
Get Opaque Data	Retrieve opaque data of the specified type		GET {baseURI}/msr/sub/KeyName/KeyValue/data/opaqueDataType
Delete Opaque Data	Delete opaque data of the specified type		DELETE {baseURI}/msr/sub/KeyName/KeyValue/data/opaqueDataType

Set Opaque Data

Description

This operation updates (or creates if it does not exist) the opaque data of the specified type for the subscriber identified by the *keyName* and *keyValue* in the request.

The opaque data is provided in the request content.

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

Prerequisites

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

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

Request URL

PUT {baseURI}/msr/sub/*keyName*/*keyValue*/data/*opaqueDataType*

Table 32: Request Variable Definitions: Set Opaque Data

Variable	Definition	Values
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> IMSI MSISDN NAI AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
opaqueDataType	A user defined type/name for the opaque data	<ul style="list-style-type: none"> quota state dynamicquota

Request Content

A **<subscriber>** element that contains a **<data>** element, which contains the specified opaque data for the identified subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="opaqueDataType">
    <![CDATA[
      cdataFieldValue
    ]]>
  </data>
</subscriber>
```

Table 33: Request Variable Definitions: Set Opaque Data

Variable	Definition	Values
opaqueDataType	A user defined type/name for the opaque data	<ul style="list-style-type: none"> • quota • state • dynamicquota
cdataFieldValue	Contents of XML data "blob"	

Note: The *opaqueDataType* in the Request Content is currently ignored and not validated. The *opaqueDataType* in the URL is used solely to identify the opaque data type.

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Data was successfully created/updated
400	MSR4000	Request content is not valid
400	MSR4051	Invalid value for a field
400	MSR4064	Occurrence constraint violation
404	MSR4002	Field is not defined for this data type
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined

Examples

Request 1

A request is made to create the *quota* opaque data. The subscriber does not have an existing Quota entity.

Request URL: **PUT** {baseURI}/msr/sub/MSISDN/33123654862/data/quota

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
```

```

    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
</data>
</subscriber>

```

Response 1

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

HTTP Status Code: 201

Response Content: None

Request 2

A request is made to update the *state* opaque data. The subscriber already has an existing State entity.

Request URL: PUT {baseURI}/msr/sub/MSISDN/33123654862/data/state

Request Content:

```

<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="state">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]>
  </data>
</subscriber>

```

Response 2

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

HTTP Status Code: 201

Response Content: None

Get Opaque Data

Description

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

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

Prerequisites

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

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

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

Request URL

GET {baseURI}/msr/sub/*KeyName*/*keyValue*/data/*opaqueDataType*

Table 34: Request Variable Definitions: Get Opaque Data

Variable	Definition	Values
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
opaqueDataType	A user defined type/name for the opaque data	<ul style="list-style-type: none"> • quota • state • dynamicquota

Request Content

None

Response Content

A **<subscriber>** element that contains a **<data>** element, which contains the requested opaque data for the identified subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="opaqueDataType">
    <![CDATA[
      cdataFieldValue
    ]]>
  </data>
</subscriber>
```

Table 35: Response Variable Definitions: Get Opaque Data

Variable	Definition	Values
opaqueDataType	A user defined type/name for the opaque data	<ul style="list-style-type: none"> • quota • state • dynamicquota
cdataFieldValue	Contents of XML data "blob"	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested data exists for subscriber
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found

Examples

Request 1

A request is made to get the *quota* opaque data for a subscriber.

Request URL: **GET** {baseURI}/msr/sub/MSISDN/33123654862/data/quota

Request Content: None

Response 1

The request is successful, and the Quota opaque data is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
    <![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
```

```
</data>
</subscriber>
```

Request 2

Request URL: **GET** {baseURI}/msr/sub/MSISDN/33123654862/data/state

Request Content: None

Response 2

The request is successful, and the State opaque data is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="state">
    <![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]>
  </data>
</subscriber>
```

Delete Opaque Data**Description**

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

Only one opaque data type can be deleted per request.

Note: If the opaque data of the *opaqueDataType* does not exist for the pool, this is not considered an error and a successful result will be returned.

Prerequisites

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

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

Request URL

DELETE {baseURI}/msr/sub/keyName/keyValue/data/opaqueDataType

Table 36: Request Variable Definitions: Delete Opaque Data

Variable	Definition	Values
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
opaqueDataType	A user defined type/name for the opaque data	<ul style="list-style-type: none"> • quota • state • dynamicquota

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Data was successfully deleted
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined

Examples**Request 1**

A request is made to delete the *quota* opaque data.

Request URL: **DELETE** {baseURI}/msr/sub/MSISDN/33123654862/data/quota

Request Content: None

Response 1

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

HTTP Status Code: 204

Response Content: None

Request 2

A request is made to delete the *state* opaque data.

Request URL: **DELETE** {baseURI}/msr/sub/MSISDN/33123654862/data/state

Request Content: None

Response 2

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

HTTP Status Code: 204

Response Content: None

Request 3

A request is made to delete the *state* opaque data. The subscriber does not have any *state* opaque data.

Request URL: **DELETE** {baseURI}/msr/sub/MSISDN/33123654862/data/state

Request Content: None

Response 3

The request is successful, although no State opaque data was deleted.

HTTP Status Code: 204

Response Content: None

Subscriber Data Row Commands

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

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

Table 37: Summary of Subscriber Data Row Commands

Command	Description	Key	Command Syntax
Set Row	Create/update data row in data of the specified type.	(MSISDN, IMSI, NAI or AccountId) and Row Identifier	PUT {baseURI}/msr/sub/ <i>KeyName/KeyValue/data/transparentDataType/rowIdValue</i>
Get Row	Retrieve data row from data of the specified type.		GET {baseURI}/msr/sub/ <i>KeyName/KeyValue/data/transparentDataType/rowIdValue</i>
Delete Row	Delete data row within data of the specified type		DELETE {baseURI}/msr/sub/ <i>KeyName/KeyValue/data/transparentDataType/rowIdValue</i>

Set Row

Description

This operation creates a new or updates an existing data row for the subscriber identified by the *keyName* and *keyValue*.

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

If more than one existing row matches the requested *rowIdValue*, then the update request will fail.

If the specified row does not exist, it is created. If the row does exist, it is updated/replaced.

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

Note: The attribute in the XML blob that contains the row name (for example, for Quota, the element <quota **name**="AggregateLimit"> contains the attribute called "name") is case-sensitive, and must be specified as defined.

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

Prerequisites

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

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

Request URL

PUT {baseURI}/msr/sub/*keyName*/*keyValue*/data/*transparentDataType*/*rowIdValue*

Table 38: Request Variable Definitions: Set Row Value

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
transparentDataType	A user defined type/name for the transparent data	quota
rowIdValue	The row name value that identifies the row within the data blob	

Request Content

A **<subscriber>** element that contains a **<data>** element, which contains the specified data *row* for the identified subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
rowValue
```

Table 39: Request Variable Definitions: Set Row Value

Variable	Definition	Value
rowValue	Contents of the XML data "blob", with the row data Note: the rowValue is of the same format as an entire Quota entity, just containing a single row, the row being added	

Note: The data contained within the *rowValue* will contain the same *rowIdValue* as specified in the URL. The *rowIdValue* in the **URL** is currently ignored, and is not validated. The *rowIdValue* **in the request content** is solely used to identify the row.

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Data row was successfully created/updated
400	MSR4000	Request content is not valid
400	MSR4051	Invalid value for a field
400	MSR4064	Occurrence constraint violation
400	MSR4067	Multiple matching rows found
404	MSR4001	Subscriber is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined

Examples

Request 1

A request is made to create a data row in the *quota* transparent data for a subscriber. The data row identifier field value is *AggregateLimit*. The subscriber does not have an existing Quota row called *AggregateLimit*.

Request URL: **PUT {baseURI}/msr/sub/MSISDN/33123654862/data/quota/AggregateLimit**

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
```

Response 1

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

HTTP Status Code: 201

Response Content: None

Request 2

A request is made to update a data row in the *quota* transparent data for a subscriber. The data row identifier field value is *Q1*. The subscriber has an existing Quota row called *Q1*.

Request URL: PUT {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Q1">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
```

Response 2

The request is successful, and the data row *Q1* was updated.

HTTP Status Code: 201

Response Content: None

Request 3

A request is made to update a data row in the *quota* transparent data for a subscriber. The data row identifier field value is *Weekday*. Two instances of the *Weekday* data row exist.

Request URL: PUT {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Weekday">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
```

Response 3

The request fails, as more than one row called *Weekday* exists.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4067">errorText</error>
```

Request 4

A request is made to update a data row in the *quota* transparent data for a subscriber. The data row identifier field value is *Weekday*. The subscriber does not have Quota transparent data.

Request URL: **PUT {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday**

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Weekday">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
```

Response 4

The request is successful, and the data row as well as the Quota entity are created.

HTTP Status Code: 201

Response Content: None

Get Row

Description

This operation retrieves a transparent data row for the subscriber identified by the *keyName* and *keyValue*. The data row identifier is specified in *rowIdValue*.

All data rows that match the requested *rowIdValue* are returned.

The transparent data row identifier field value is specified in *rowIdValue*.

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

Prerequisites

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

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

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

Request URL

GET {baseURI}/msr/sub/*keyName*/*keyValue*/data/*transparentDataType*/*rowIdValue*

Table 40: Request Variable Definitions: Get Row

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
transparentDataType	A user defined type/name for the transparent data	quota
rowIdValue	The row name value that identifies the row within the transparent data blob	

Request Content

None

Response Content

A **<subscriber>** element that contains a **<data>** element, which contains the specified transparent data *row* (if it exists) for the identified subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="transparentDataType">
<![CDATA[
  cdataRowValue
```

```

]]>
  </data>
</subscriber>

```

Table 41: Response Variable Definitions: Get Row

Variable	Definition	Value
transparentDataType	A user defined type/name for the transparent data	quota
cdataRowValue	Contents of the XML data "blob", with the row data	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested data row exists for subscriber
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples**Request 1**

A request is made to get the *Q1* data row from the *quota* transparent data for a subscriber. The subscriber has the Quota entity, and the *Q1* data row exists.

Request URL: **GET** {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1

Request Content: None

Response 1

The request is successful, and the Quota transparent data row requested is returned.

HTTP Status Code: 200

Response Content:

```

<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Q1">
    <cid>9223372036854775807</cid>
    <time>1</time>
    <totalVolume>0</totalVolume>

```

```

    <inputVolume>0</inputVolume>
    <outputVolume>0</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-12T16:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
</data>
</subscriber>

```

Request 2

A request is made to get the *Weekend* data row from the *quota* transparent data for a subscriber. The subscriber has the Quota entity, but the *Weekend* data row does NOT exist.

Request URL: **GET** {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekend

Request Content: None

Response 2

The request fails because the data row does not exist.

HTTP Status Code: 404

Response Content:

```

<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4059">errorText</error>

```

Request 3

A request is made to get the *Weekday* data row from the *quota* transparent data for a subscriber. The subscriber has the Quota entity. Two instances of the *Weekday* data row exist.

Request URL: **GET** {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday

Request Content: None

Response 3

The request is successful, and the Quota transparent data rows requested are returned.

HTTP Status Code: 200

Response Content:

```

<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
    <![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Weekend">
    <cid>9223372036854775807</cid>
    <time>1</time>
    <totalVolume>0</totalVolume>
    <inputVolume>0</inputVolume>
    <outputVolume>0</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-12T16:00:00-05:00</nextResetTime>
  </quota>

```

```

<quota name="Weekend">
  <cid>7682364872564782343</cid>
  <time>32</time>
  <totalVolume>250</totalVolume>
  <inputVolume>4570</inputVolume>
  <outputVolume>11230</outputVolume>
  <serviceSpecific>29</serviceSpecific>
  <nextResetTime>2010-06-01T16:00:00-05:00</nextResetTime>
</quota>
</usage>
]]>
</data>
</subscriber>

```

Delete Row

Description

This operation deletes a transparent data row for the subscriber identified by the *keyName* and *keyValue*. The transparent data row identifier field value is specified in *rowIdValue*.

If more than one row matches the requested *rowIdValue*, then all matching rows are deleted.

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

Prerequisites

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

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

Request URL

DELETE {baseURI}/msr/sub/*keyName*/*keyValue*/data/*transparentDataType*/*rowIdValue*

Table 42: Request Variable Definitions: Delete Row

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> IMSI MSISDN NAI AccountId
keyValue	Corresponding key field value assigned to <i>keyNameX</i>	
transparentDataType	A user defined type/name for the transparent data	Quota
rowIdValue	The row name value that identifies the row within the data blob	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Data row was successfully deleted
400	MSR4064	Occurrence constraint violation
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found

Examples**Request 1**

A request is made to delete the *Q1* data row in the Quota transparent data. The *Q1* data row exists in the Quota data.

Request URL: **DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1**

Request Content: None

Response 1

The request is successful, and the data row in the Quota transparent data was deleted.

HTTP Status Code: 204

Response Content: None

Request 2

A request is made to delete the *Weekend* data row in the Quota transparent data. The *Weekend* data row does NOT exist in the Quota transparent data.

Request URL: **DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekend**

Request Content: None

Response 2

The request is successful, even though the *Weekend* Quota row does not exist.

HTTP Status Code: 204

Response Content: None

Request 3

A request is made to delete the *Bonus* data row in the Quota transparent data. The Quota opaque data is a valid entity, but the requested subscriber does not contain any Quota opaque data.

Request URL: **DELETE** {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Bonus

Request Content: None

Response 3

The request fails, because the specified subscriber does not contain Quota data.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4058">errorText</error>
```

Subscriber Data Row Field Commands

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

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

Table 43: Summary of Subscriber Data Row Field Commands

Command	Description	Key(s)	Command Syntax
Get Row Field	Retrieve value(s) for the specified field	(MSISDN, IMSI, NAI or AccountId) and Row Identifier and Field name	GET {baseURI}/msr/sub/ <i>keyName</i> / <i>keyValue</i> /data/ <i>transparentDataType</i> / <i>rowIdValue</i> / <i>fieldName</i>
Get Row Field Value	Retrieve a single value for the specified field		GET {baseURI}/msr/sub/ <i>keyName</i> / <i>keyValue</i> /data/ <i>transparentDataType</i> / <i>rowIdValue</i> / <i>fieldName</i> / <i>fieldValue</i>
Update Field	Update field to the specified value		PUT {baseURI}/msr/sub/ <i>keyName</i> / <i>keyValue</i> /data/ <i>transparentDataType</i> / <i>rowIdValue</i> / <i>fieldName</i> / <i>fieldValue</i>
Delete Field	Delete all values for the specified field		DELETE {baseURI}/msr/sub/ <i>keyName</i> / <i>keyValue</i> /data/ <i>transparentDataType</i> / <i>rowIdValue</i> / <i>fieldName</i>

Get Row Field

Description

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

All data rows that match the requested *rowIdValue* are returned.

If more than one row matches the requested *rowIdValue*, then all matching rows will be returned.

The transparent data row identifier field value is specified in *rowIdValue*. The field name is specified in *fieldName*.

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

Prerequisites

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

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

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

The field name specified must be a valid field for the Entity as defined in the SEC.

Request URL

```
GET {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/
rowIdValue/fieldName
```

Table 44: Request Variable Definitions: Get Row Field

Variable	Definition	Value
KeyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
transparentDataType	A user defined type/name for the transparent data	quota
rowIdValue	The row name value that identifies the row within the transparent data blob	
fieldName	A user defined field within the transparent data row	

Request Content

None

Response Content

A **<subscriber>** element that contains a **<data>** element, which contains the specified transparent data *row* (if it exists) for the identified subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="transparentDataType">
    <![CDATA[
      cdataRowFieldValue
    ]]>
  </data>
</subscriber>
```

Table 45: Response Variable Definitions: Get Row Field

Variable	Definition	Value
transparentDataType	A user defined type/name for the transparent data	quota
cdataRowValuee	Contents of the XML data "blob", with the row data	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested data row field exists for subscriber
404	MSR4001	Subscriber is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist
404	MSR4065	Field is not set

Examples

Request 1

A request is made to get the *inputVolume* field in the *Q1* data row of the Quota transparent data for a subscriber.

Request URL:

```
GET {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/inputVolume
```

Request Content: None

Response 1

The request is successful, and the requested field value is returned

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Q1">
    <inputVolume>980</inputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Request 2

A request is made to get the *outputVolume* field in the *Weekday* data row of the Quota transparent data for a subscriber. Two instances of the *Weekday* data row exist.

Request URL:

```
GET {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday/outputVolume
```

Request Content: None

Response 2

The request is successful, and the field from two matching *Weekday* rows are returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Q1">
    <inputVolume>980</outputVolume>
  </quota>
  <quota name="Q1">
    <inputVolume>2140</outputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Get Row Field Value

Description

This operation retrieves a field with a given value, within a transparent data row for the subscriber identified by the *keyName* and *keyValue*.

If more than one row matches the requested *rowIdValue*, then all matching rows will be returned.

The transparent data row identifier field value is specified in *rowIdValue*. The field name is specified in *fieldName*. The field value is specified in *fieldValue*.

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

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

Prerequisites

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

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

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

The field name specified must be a valid field for the Entity as defined in the SEC.

The field value in *fieldValue* must match the specified value in the request.

Request URL

```
GET {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/
rowIdValue/fieldName
```

Table 46: Request Variable Definitions: Get Row Field Value

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
transparentDataType	A user-defined type/name for the transparent data	quota
rowIdValue	The row name value that identifies the row within the transparent data blob	
fieldName	A user defined field within the transparent data row	
fieldValue	Corresponding field value assigned to <i>fieldName</i>	

Request Content

None

Response Content

A **<subscriber>** element that contains a **<data>** element, which contains the specified transparent data *row* (if it exists) for the identified subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="transparentDataType">
    <![CDATA[
      cdataRowFieldValue
    ]]>
  </data>
</subscriber>
```

Table 47: Response Variable Definitions: Get Row Field Value

Variable	Definition	Value
transparentData Type	A user defined type/name for the transparent data	quota
cdataRowFieldValue	Contents of the XML data "blob", with the field from the row data	

Note: The response content is only present if the requested field is present in the transparent data row, and the field is set to the supplied value.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested data row field/value exists for subscriber
400	MSR4053	Data row field value does not match
404	MSR4001	Subscriber is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples

Request 1

A request is made to get the *inputVolume* field in the *Q1* data row of the Quota transparent data for a subscriber. The *inputVolume* field exists, and is set to the value 980.

Request URL:

```
GET {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/inputVolume/980
```

Request Content: None

Response 1

The request is successful, and the requested field value is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Q1">
    <inputVolume>980</inputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Request 2

A request is made to get the *outputVolume* field with the value of 2000 in the *Q4* data row of the Quota transparent data for a subscriber. The *outputVolume* field exists, but is set to the value 1500.

Request URL:

```
GET {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/outputVolume/2000
```

Request Content: None

Response 2

The request fails because the requested field does not have the supplied value.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4053">errorText</error>
```

Request 3

A request is made to get the *inputVolume* field with the value of 2330 in the *Weekday* data row of the Quota transparent data for a subscriber. Two instances of the *Weekday* data row exist. The *inputVolume* field exists in both rows, and is set to the value 3220 in both rows.

Request URL:

```
GET {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday/inputVolume/3220
```

Request Content: None

Response 3

The request is successful, and the field from two matching *Weekday* rows are returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Weekday">
    <inputVolume>3220</inputVolume>
  </quota>
  <quota name="Weekday">
    <inputVolume>3220</inputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Request 4

A request is made to get the *inputVolume* field with the value of 980 in the *Weekday* data row of the Quota transparent data for a subscriber. Two instances of the *Weekday* data row exist. The *inputVolume* field exists in both rows, and in one row is set to the value 980, and in the other row it is set to the value 3220

Request URL:

```
GET {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday/inputVolume/980
```

Request Content: None

Response 4

The request is successful, and the field from the single matching *Weekday* row is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="Weekday">
    <inputVolume>980</inputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Update Row Field**Description**

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

The transparent data row identifier field is value is specified in *rowIdValue*. The field name is specified in *fieldName*.

If the specified field is valid, but does not currently exist, it will be created.

If more than one existing row matches the requested *rowIdValue*, then the update request will fail.

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

Prerequisites

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

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

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

The field name specified must be a valid field for the Entity as defined in the SEC.

Request URL

```
PUT {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/fieldName/fieldValue
```

Table 48: Request Variable Definitions: Update Row Field

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
transparentDataType	A user defined type/name for the transparent data	quota
rowIdValue	The row name value that identifies the row within the transparent data blob	
fieldName	A user defined field within the transparent data row	
fieldValue	Corresponding field value assigned to <i>fieldName</i>	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Requested transparent data row field was successfully created
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4067	Multiple matching rows found
404	MSR4001	Subscriber is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples

Request 1

A request is made to update the value of the *inputVolume* field in the *Q1* data row of the Quota transparent data for a subscriber.

Request URL:

```
PUT {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/inputVolume/0
```

Request Content: None

Response 1

The request is successful, and the field in the data row in the Quota transparent data was updated.

HTTP Status Code: 201

Response Content: None

Request 2

A request is made to update the *cid* field in the *Q1* data row in the Quota transparent data. The *cid* field is not allowed to be updated.

Request URL:

```
PUT {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/cid/45678
```

Request Content: None

Response 2

The request fails, because the *cid* field cannot be updated.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4056">errorText</error>
```

Request 3

A request is made to update the *inputVolume* field in the *Weekday* data row of the Quota transparent data for a subscriber. Two instances of the *Weekday* data row exist.

Request URL:

```
PUT {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday/inputVolume/0
```

Request Content: None

Response 3

The request fails because more than one row called *Weekday* exists.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4067">errorText</error>
```

Delete Row Field

Description

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

The transparent data row identifier field value is specified in *rowIdValue*. The field name is specified in *fieldName*.

If more than one row matches the requested *rowIdValue*, then the delete request will fail.

Note: If the field with opaque data of the *opaqueDataType* does not exist, this is not considered an error and a successful result will be returned.

Note: if the field being deleted is mandatory, and is defined as having a default value, then the field will not be removed, but will have the default value assigned.

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

Prerequisites

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

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

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

The field name specified must be a valid field for the Entity as defined in the SEC. The field must be updatable.

Request URL**DELETE**

`{baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/fieldName`

Table 49: Request Variable Definitions: Delete Row Field

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
transparentDataType	A user defined type/name for the transparent data	quota
rowIdValue	The row name value that identifies the row within the transparent data blob	
fieldName	A user-defined field within the subscriber Profile	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested transparent data row field was successfully deleted
400	MSR4056	Field is not updatable
400	MSR4067	Multiple matching rows found
404	MSR4001	Subscriber is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples**Request 1**

A request is made to delete the *inputVolume* field in the *Q1* data row in the Quota transparent data for subscriber.

Request URL:

```
DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/inputVolume
```

Request Content: None

Response 1

The request is successful, and the data row in the Quota transparent data was deleted.

HTTP Status Code: 204

Response Content: None

Request 2

A request is made to delete the *inputVolume* field in the *Weekday* data row in the Quota transparent data for a subscriber. Two instances of the *Weekday* data row exist.

Request URL:

```
DELETE {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday/inputVolume
```

Request Content: None

Response 2

The request fails because more than one row called *Weekday* exists.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4067">errorText </error>
```

Special Operations

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

The required row is identified in the request by the *rowIdValue*. A specific instance of a quota (i.e. a specified row) within the Quota transparent data entity can have its fields reset to pre-defined values using a provisioning command.

Table 50: Summary of Subscriber Special Operation Commands

Command	Description	Key	Command Syntax
Reset Quota	Reset the fields within the specified Quota	(MSISDN, IMSI, NAI or AccountId) and Row Identifier	POST {BaseURI}/msr/sub/keyName/KeyValue/data/transparentDataType/rowIdValue

Reset Quota

Description

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

If more than one row matches the requested *rowIdValue*, then the delete request will fail.

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

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

Prerequisites

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

The Quota transparent data must exist for the subscriber.

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

Request URL

POST {BaseURI}/msr/sub/*keyName/keyValue*/data/*transparentDataType/rowIdValue*

Table 51: Request Variable Definitions: Reset Quota Value

Variable	Definition	Value
keyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValue	Corresponding key field value assigned to <i>keyName</i>	
transparentDataType	A user defined type/name for the transparent data	quota
rowIdValue	The row name value that identifies the row within the transparent data blob	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested transparent data row was successfully reset
400	MSR4067	Multiple matching rows found
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist
409	MSR4063	Entity cannot be reset

Examples

Request 1

A request is made to reset the *Q1* Quota row for a subscriber. The subscriber has Quota transparent data, and the Quota transparent data contains a Quota row called *Q1*.

Request URL: **POST** {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1

Request Content: None

Response 1

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

HTTP Status Code: 204

Response Content: None

Request 2

A request is made to reset the *Q1* Quota row for a subscriber. The subscriber does not have Quota transparent data.

Request URL: **POST** {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1

Request Content: None

Response 2

The request fails because the subscriber does not have Quota transparent data.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4049">errorText</error>
```

Request 3

A request is made to reset the *Q6* Quota row for a subscriber. The subscriber has Quota transparent data, but the Quota transparent data does NOT contain a Quota row called *Q6*.

Request URL: **POST** {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1

Request Content: None

Response 3

The request fails because the Q6 row does not exist.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4059">errorText</error>
```

Request 4

A request is made to reset the *Weekday* Quota row for a subscriber. The subscriber has Quota transparent data, and the Quota transparent data contains two instances of the *Weekday* data row exist.

Request URL: **POST** {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday

Request Content: None

Response 4

The request fails because more than one row called *Weekday* exists.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4067">errorText</error>
```

Pool Provisioning

Topics:

- *Pool Profile Commands.....115*
- *Pool Profile Field Commands.....122*
- *Pool Opaque Data Commands.....136*
- *Additional Pool Commands.....143*

A pool is a group of subscribers that share common data. Subscribers in a pool share all the entities of that pool.

Provisioning clients can create, retrieve, modify, and delete pool data. Pool data is accessed via the PoolID value associated with the pool.

Pool Profile Commands

Table 52: Summary of Pool Profile Commands

Command	Description	Key	Command Syntax
Create Pool	Create a new pool/pool Profile.	PoolID	POST <code>{baseURI}/msr/pool</code>
Get Pool	Get pool Profile data.		GET <code>{baseURI}/msr/pool/keyName/keyValue</code>
Update Pool	Replace an existing pool Profile.		PUT <code>{baseURI}/msr/pool/keyName/keyValue</code>
Delete Pool	Delete all pool Profile data and all opaque data associated with the pool.		DELETE <code>{baseURI}/msr/pool/keyName/keyValue</code>

Create Pool

Description

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

Unlike other pool commands, the *key value* (PoolID) is not specified in the URL. Request content includes *poolId* and field-value pairs, all as specified in the Subscriber Entity Configuration.

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

Prerequisites

A pool with the supplied *PoolID* must not exist.

Request URL

POST `{baseURI}/msr/pool`

Request Content

A `<pool>` element that contains a `<field>` element for every field-value pair defined for the new subscriber.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="PoolID">poolId</field>
  [
    <field name="fieldName1">fieldValue1</field>
    <field name="fieldName2">fieldValue2</field>
  ]
</pool>
```

```

:
<field name="fieldNameN">fieldValueN</field>
]
</pool>

```

Table 53: Request Variable Definitions: Create Pool

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-9999999999999999999999
fieldNameX	A user defined field within the pool Profile	
fieldValueX	Corresponding field value assigned to <i>fieldNameX</i> Note: The PoolID/field order in the request is not important.	

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Successfully created
400	MSR4000	The field list does not contain at least one unique key
400	MSR4003	A key is detected to be already in the system for another pool
400	MSR4004	The field list does not contain at least one unique key
400	MSR4051	Invalid value for a field
400	MSR4064	Occurrence constraint violation
404	MSR4002	Pool field is not defined

Examples**Request 1**

A pool is created, with a *PoolID* key. The *BillingDay*, *Tier*, *Entitlement*, and *Custom15* fields are set.

Request URL: **POST** {baseURI}/msr/pool

Request Content:

```

<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="PoolID">100000</field>
  <field name="BillingDay">5</field>
  <field name="Tier">12</field>
  <field name="Entitlement">Weekpass</field>

```

```
<field name="Entitlement">Daypass</field>
<field name="Custom15">allocate</field>
</pool>
```

Response 1

The request is successful, and the pool was created.

HTTP Status Code: 201

Response Content: None.

Request 2

A pool is created, with a *PoolID* key. The *BillingDay* and *Entitlement* fields are set . A pool already exists with the given *PoolID*.

Request URL: **POST** {baseURI}/msr/pool

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="PoolID">100001</field>
  <field name="BillingDay">5</field>
  <field name="Entitlement">Weekpass,Daypass</field>
</pool>
```

Response 2

The request fails. The error code indicates that the *PoolID* already exists.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4003">errorText</error>
```

Get Pool

Description

This operation retrieves all field-value pairs created for a pool that is identified by the *poolId*.

The response content includes only valid field-value pairs which have been previously provisioned or created by default.

Prerequisites

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

Request URL

GET {baseURI}/msr/pool/*poolId*

Table 54: Request Variable Definitions: Get Pool

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

Request Content

None

Response Content

A <pool> element that contains a <field> element for every field-value pair defined for the pool.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="PoolID">poolId</field>
  [
    <field name="fieldName1">fieldValue1</field>
    <field name="fieldName2">fieldValue2</field>
    :
    <field name="fieldNameN">fieldValueN</field>
  ]
</pool>
```

Table 55: Response Variable Definitions: Get Pool

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1- 99999999999999999999
fieldNameX	A user defined field within the pool profile	
fieldValueX	Corresponding field value assigned to <i>fieldNameX</i> Note: Key/field order in the request is not important.	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Successfully located the pool
404	MSR4001	Could not find the subscriber by key PoolID

Examples**Request 1**

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

Request URL: **GET** {baseURI}/msr/pool/100000

Request Content: None

Response 1

The request is successful, and the pool was retrieved.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="PoolID">100000</field>
  <field name="BillingDay">5</field>
  <field name="Tier">12</field>
  <field name="Entitlement">Weekpass</field>
  <field name="Entitlement">Daypass</field>
  <field name="Custom15">allo</field>
</pool>
```

Request 2

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

Request URL: **GET** {baseURI}/msr/pool/222200

Request Content: None

Response 2

The request fails because the pool does not exist.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4001">errorText</error>
```

Update Pool

Description

This operation replaces an existing pool profile with the pool identified by the *poolId*.

With the exception of the *PoolID*, all existing data for the pool is completely removed and replaced by the request content. Therefore, it is not necessary to include the *PoolID* from the URI in the request content (although it is not an error if it is included).

Note:

If the *PoolID* is included in the content, and it is different from the value specified in the URL, the request will fail.

Prerequisites

A pool with the *poolId* supplied must exist.

Request URL

PUT {baseURI}/msr/pool/*poolId*

Table 56: Request Variable Definitions Update Pool

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-9999999999999999999999

Request Content

A <pool> element that contains a <field> element for every field-value pair defined for the pool.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
[
  <field name="PoolID">poolId</field>
  <field name="fieldName1">fieldValue1</field>
  <field name="fieldName2">fieldValue2</field>
  :
  <field name="fieldNameN">fieldValueN</field>
]
</pool>
```

Table 57: Request Variable Definitions: Update Pool

Variable	Definition	Value
poolId	poolId value of the pool. Numeric value, 1-22 digits in length	1-9999999999999999999999
fieldNameX	A user defined field within the pool Profile	
fieldValueX	Corresponding field value assigned to <i>fieldNameX</i> Note: PoolID/field order in the request is not important.	

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	The pool data was replaced successfully
400	MSR4000	Invalid content/payload
400	MSR4000	The PoolID supplied in URL and request content do not match
400	MSR4051	Invalid value for a field
400	MSR4064	Occurrence constraint violation
404	MSR4001	Could not find the pool by PoolID
404	MSR4002	Pool field is not defined

Example**Request**

A pool is updated. The *BillingDay*, *Tier*, *Entitlement*, and *Custom15* fields are set. The pool exists.

Request URL: **PUT** `{BaseURI}/msr/pool/100000`

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <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>
```

Response

The request is successful, and the pool was updated.

HTTP Status Code: 204

Response Content: None

Delete Pool**Description**

This operation deletes all pool profile data 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 not have any member subscribers, or the request will fail.

Request URL

DELETE `{baseURI}/msr/pool/poolId`

Table 58: Request Variable Definition: Delete Pool

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999999999999999

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	The pool was successfully deleted
404	MSR4001	Could not find the pool by PoolID
409	MSR4055	The pool could not be deleted as it has member subscribers

Examples**Request 1**

The pool with the given *PoolID* is deleted. The pool exists and has no member subscribers.

Request URL: **DELETE** {baseURI}/msr /pool/100000

Request Content: None

Response 1

The request is successful.

HTTP Status Code: 204

Response Content: None.

Request 2

The pool with the given *PoolID* is deleted. The pool exists and has no member subscribers.

Request URL: **DELETE** {baseURI}/msr/pool/200000

Request Content: None

Response 2

The request fails because the pool has member subscribers.

HTTP Status Code: 409

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4055">errorText</error>
```

Pool Profile Field Commands

Table 59: Summary of Pool Profile Field Commands

Command	Description	Key	Command Syntax
Add Field Value	Add a value to the specified field. This operation does not affect	poolId	POST {baseURI}/msr/pool/ <i>PoolID</i> / <i>field/fieldName/fieldValue</i>

Command	Description	Key	Command Syntax
	any pre-existing values for the field		
Get Field	Retrieve the value(s) for the specified field		GET {baseURI}/msr/pool/poolId/field/fieldName
Get Field Value	Retrieve the single value for the specified field (if set as specified)		GET {baseURI}/msr/pool/poolId/field/fieldName/fieldValue
Update Field Value	Updates field to the specified value		PUT {baseURI}/msr/pool/poolId/field/fieldName/fieldValue
Update Multiple Fields	Update multiple fields to the specified values		PUT {baseURI}/msr/pool/poolId/multipleFields/fieldName1/fieldValue1/fieldName2/fieldValue2/...
Delete Field	Delete all values for the specified field		DELETE {baseURI}/msr/pool/poolId/field/fieldname
Delete Field Value	Delete a value for the specified field		DELETE {baseURI}/msr/pool/poolId/field/fieldName/fieldValue

Add Field Value

Description

This operation adds a value to the specified multi-value field for the pool identified by the *poolId*.

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

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

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

Note: If the field to which the value is being added does not exist, it will be created.

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

Prerequisites

A pool with the *poolId* supplied must exist.

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

The value *fieldValue* being added must NOT already be present in the field.

Request URL

POST {baseURI}/msr/pool/poolId/field/fieldName/fieldValue

Table 60: Request Variable Definitions: Add Field Value

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
fieldName	A user defined field within the pool Profile	
fieldValue	Corresponding field value assigned to <i>fieldName</i> Note: For multi-value fields, the value will contain a semicolon-separated list of values on a single line. E.g. "a;b;c"	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Successfully added field values
400	MSR4005	Field does not support multiple values
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4066	Field value already exists
404	MSR4001	Pool is not found
404	MSR4002	Pool field is not defined

Examples**Request 1**

A request is made to add the value *DayPass* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* value is not already present in the *Entitlement* field.

Request URL: **POST** `{baseURI}/msr/pool/poolId/100000/field/Entitlement/DayPass`

Request Content: None

Response 1

The request is successful, and the value was added to the *Entitlement* field.

HTTP Status Code: 200

Response Content: None

Request 2

A request is made to add the values *DayPass* and *HighSpeedData* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* and *HighSpeedData* values are not already present in the *Entitlement* field.

Request URL: **POST {baseURI}/msr/pool/200000/field/Entitlement/DayPass; HighSpeedData**

Request Content: None

Response 2

The request is successful, and the values were added to the *Entitlement* field.

HTTP Status Code: 200

Response Content: None

Get Field

Description

This operation retrieves the value(s) for the specified field(s) for the pool identified by the specified *poolId*.

Note that depending upon the field entered, there may be multiple field-value pairs returned by this operation.

Prerequisites

A pool with the *PoolID* supplied must exist. The requested field *fieldName* must be a valid field in the pool profile.

RequestURL

GET {baseURI}/msr/pool/*poolId*/field/*fieldName*

Table 61: Request Variable Definitions: Get Field

Variable	Definition	Value
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999999999999999
fieldName	A user defined field within the pool Profile	

Request Content

None

Response Content

A <pool> element that contains a <field> element for every value defined for the specified field within the pool.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="fieldName">fieldValue1</field>
  [
    <field name="fieldName">fieldValue2</field>
    :
    <field name="fieldName">fieldValueN</field>
  ]
</pool>
```

Table 62: Response Variable Definitions: Get Field

Variable	Definition	Value
fieldName	A user defined field within the pool Profile	
fieldValueX	Corresponding field value assigned to <i>fieldName</i>	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested field exists for pool
404	MSR4001	Pool is not found
404	MSR4002	Pool field is not defined
404	MSR4065	Field is not set

Example**Request**

A request is made to get the *Entitlement* field for a pool.

Request URL: **GET** {BaseURI}/msr/pool/100000/field/Entitlement

Request Content: None

Response

The request is successful, and the requested value is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="Entitlement">Weekpass</field>
  <field name="Entitlement">Daypass</field>
</pool>
```

Get Field Value

Description

This operation retrieves the values for the specified field for the pool identified by the *poolId* in the request.

For a request where the presence of multiple values for a multi-value field is requested, a match is only considered to have been made if the requested values form a subset of the values stored in the pool profile. That is, if all of the values requested exist in the pool profile, the request returns success regardless of how many other values may exist in the profile. If any or all of the values are not present as part of the pool profile, an error is returned.

Note that depending upon the field entered, there may be multiple field-value pairs returned by this operation.

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

Prerequisites

A pool with the *PoolID* supplied must exist.

The requested field *fieldName* must be a valid field in the pool profile.

The field value in the *fieldValue* field must match the specified value in the request.

Request URL

GET {baseURI}/msr/pool/*poolId*/field/*fieldName*/*fieldValue*

Table 63: Request Variable Definitions: Get Field Value

Variable	Definition	Values
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
fieldName	A user-defined field within the pool Profile	
fieldValue	Corresponding field value assigned to <i>fieldName</i> Note: For multi-value fields, the value will contain a semicolon-separated list of values on a single line, e.g., "a;b;c". Note: The semicolon between the field values may need to be encoded as %3B for certain clients.	

Request Content

None

Response Content

A `<pool>` element that contains a `<field>` element for every field-value pair requested that matches the value supplied for the pool.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="fieldName">fieldValue1</field>
  [
    <field name="fieldName">fieldValue2</field>
    :
    <field name="fieldName">fieldValueN</field>
  ]
</pool>
```

Table 64: Response Variable Definitions: Get Field Value

Variable	Definition	Values
fieldName	A user defined field within the subscriber Profile	
fieldValueX	Corresponding key field value assigned to <i>fieldName</i>	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested field exists for pool
400	MSR4053	Pool and field exist, but value(s) do not match
404	MSR4001	Pool is not found
404	MSR4002	Pool field is not defined

Examples**Request 1**

A request is made to get the *Tier* field with the value *Gold*. The field exists and has the specified value.

Request URL: **GET** {baseURI}/msr/pool/200000/field/Tier/Gold

Request Content: None

Response 1

The request is successful, and the requested value is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="Tier">Gold</field>
</pool>
```

Request 2

Variable	Definition	Values
	Note: The semicolon between the field values may need to be encoded as %3B for certain clients.	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Field was successfully updated
400	MSR4051	The value provided for the field is invalid
400	MSR4056	Field is not updatable
404	MSR4001	Pool does not exist
404	MSR4002	Pool field is not defined

Example**Request**

A request is made to update the *Entitlement* field with the values *DayPass* and *HighSpeedData*. The *Entitlement* field is a multi-value field.

Request URL:

```
PUT {baseURI}/msr/pool/100000/field/Entitlement/DayPass;HighSpeedData
```

Response Content: None.

Response

The request is successful, and the *Entitlement* field was updated.

HTTP Status Code: 201

Response Content: None

Update Multiple Fields**Description**

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

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Field(s) were successfully updated
400	MSR4051	The value provided for the field is invalid
400	MSR4056	Field is not updatable
400	MSR4057	Request contains only one field to update
404	MSR4001	Pool does not exist
404	MSR4002	Pool field is not defined

Example**Request**

A request is made to update the *Entitlement* field to *Weekend* and *YearPass*, the *Tier* field to *Silver*, and the *BillingDay* field to *11*.

Request URL:

```
PUT {baseURI}/msr/pool/300001/multipleFields/Entitlement/Weekend;YearPass/Tier/Silver/BillingDay/11
```

Request Content: None

Response

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

HTTP Status Code: 201

Response Content: None

Delete Field**Description**

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

If the field is a multi-value field, then all values are deleted. Deletion of a field results in the removal of the entire field from the pool profile, that is, the field is not present, not just empty (without a value).

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

Note: This command cannot be used to delete the PoolID.

Note: If the field being deleted is mandatory and is defined as having a default value, then the field will not be removed but will have the default value assigned.

Prerequisites

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

The requested field *fieldName* must be a valid field in the pool profile.

Request URL

DELETE {baseURI}/msr/pool/poolId/field/fieldName

Table 67: Request Variable Definitions: Delete Field

Variable	Definition	Values
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
fieldName	A user defined field within the subscriber Profile	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Field was successfully deleted
400	MSR4056	Field is not updatable
404	MSR4001	Pool does not exist
404	MSR4002	Pool field is not defined

Example

Request

A request is made to delete the *Entitlement* field. The field is a valid pool profile field.

Request URL: **DELETE** {BaseURI}/msr/pool/100000/field/Entitlement

Request Content: None

Response

The request is successful, and the field was deleted.

HTTP Status Code: 204

Response Content: None

Delete Field Value

Description

This operation deletes a single value from the specified field for the pool profile identified by the *poolId* in the request.

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

Each individual value is removed from the pool profile. If a supplied value does not exist, then it is ignored. For example, if a profile contains values "a;b;c" and a request to delete "a;b" is made, this succeeds and the profile is left with "c" as the value. If the profile contains "a;b;c" and a request is made to delete "c;d" the request succeeds and the profile is left with "a;b" as the value.

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

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

Prerequisites

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

The field *fieldName* must be a valid field in the pool Profile, and set to the value supplied to be removed successfully.

Request URL

DELETE {baseURI}/msr/pool/*poolId*/field/*fieldName*/*fieldValue*

Table 68: Request Variable Definitions: Delete Field Value

Variable	Definition	Values
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
fieldName	A user-defined field within the pool profile	
fieldValue	Corresponding key field value assigned to <i>fieldName</i> Note: For multi-value fields, the value will contain a semicolon separated list of values on a single line, e.g., "a;b;c". Note: The semicolon between the field values may need to be encoded as %3B for certain clients.	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested field(s) were successfully deleted
400	MSR4005	Field does not support multiple values
400	MSR4056	Field is not updatable
404	MSR4001	Pool does not exist
404	MSR4002	Pool field is not defined

Examples

Request 1

A request is made to delete the values *DayPass* and *WeekendPass* from the *Entitlement* field. The *Entitlement* field is a multi-value field. The *Entitlement* field exists but only contains the *DayPass* value, not the *WeekendPass* value.

Request URL:

```
DELETE {baseURI}/msr/pool/200003/field/Entitlement/DayPass;WeekendPass
```

Request Content: None

Response 1

The request is successful, because the *Entitlement* field does not contain the *WeekendPass* value.

HTTP Status Code: 204

Response Content: None

Request 2

A request is made to delete the values *DayPass* and *WeekendPass* from the *Entitlement* field. The *Entitlement* field is a multi-value field. The field exists and contains the specified values.

Request URL:

```
DELETE {baseURI}/msr/pool/300003/field/Entitlement/DayPass;HighSpeedData
```

Request Content: None

Response 2

The request is successful, and the values were deleted from the field.

HTTP Status Code: 204

Response Content: None

Pool Opaque Data Commands

Note: The following commands perform opaque data *operations*. They can be used on entities defined as either opaque or transparent. The opaque data *operation* operates on the entity at the entire XML blob level. The entire contents of the entity is set/returned/deleted.

Table 69: Summary of Pool Opaque Data Commands

Command	Description	Key	Command Syntax
Set Opaque Data	Create/update opaque data of the specified type	PoolID	PUT {baseURI}/msr/pool/poolId/data/ <i>opaqueDataType</i>
Get Opaque Data	Retrieve opaque data of the specified type		GET {baseURI}/msr/pool/PoolID/data/ <i>opaqueDataType</i>
Delete Opaque Data	Delete opaque data of the specified type		DELETE {baseURI}/msr/pool/poolId/data/ <i>opaqueDataType</i>

Set Opaque Data

Description

This operation updates (or creates if it does not exist) opaque data of the specified type for the pool identified by the *poolId* in the request.

The opaque data is provided in the request content.

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

Prerequisites

A pool with the *poolId* supplied must exist.

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

Request URL

PUT {baseURI}/msr/pool/poolId/data/*opaqueDataType*

Table 70: Request Variable Definitions: Set Opaque Data

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

opaqueDataType	A user defined type/name for the opaque data	<ul style="list-style-type: none"> poolquota poolstate pooldynamicquota
----------------	--	--

Request Content

A `<pool>` element that contains a `<data>` element, which contains the specified opaque data for the identified pool.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="opaqueDataType">
    <![CDATA[
      cdataFieldValue
    ]]>
  </data>
</pool>
```

Table 71: Response Variable Definitions: Set Opaque Data

Variable	Definition	Values
opaqueDataType	A user defined type/name for the opaque data	Value is either <i>poolquota</i> , <i>poolstate</i> , or <i>pooldynamicquota</i>
cdataFieldValue	Contents of the XML data "blob"	

Note: The *opaqueDataType* in the Request Content is currently ignored and not validated. The *opaqueDataType* in the URL is used solely to identify the opaque data type.

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Data was successfully created/updated
400	MSR4000	Request content is not valid
400	MSR4051	Invalid value for a field
400	MSR4064	Occurrence constraint violation
404	MSR4002	Field is not defined for this data type
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined

Examples**Request 1**

A request is made to create the *poolquota* opaque data. The pool does not have an existing PoolQuota entity.

Request URL: **PUT** {baseURI}/msr/pool/100000/data/poolquota

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="poolquota">
    <![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>1</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</pool>
```

Response 1

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

HTTP Status Code: 201

Response Content: None

Request 2

A request is made to update the *poolstate* opaque data. The pool already has an existing PoolState entity.

Request URL: **PUT** {baseURI}/msr/pool/100002/data/poolstate

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="poolstate">
    <![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
</state>
</data>
</pool>
```

```
<property>
  <name>approved</name>
  <value>yes</value>
</property>
</state>
]]>
</data>
</pool>
```

Response 2

HTTP Status Code: 201

Response Content: None

Get Opaque Data

Description

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

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

Prerequisites

A pool with the *poolId* supplied must exist.

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

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

Request URL

GET {baseURI}/msr/pool/*poolId*/data/*opaqueDataType*

Variable	Definition	Values
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
opaqueDataType	A user defined type/name for the opaque data	<ul style="list-style-type: none"> • poolquota • poolstate • pooldynamicquota

Request Content

None

Response Content

A `<pool>` element that contains a `<data>` element, which contains the requested opaque data for the identified pool.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="opaqueDataType">
    <![CDATA[
      cdataFieldValue
    ]]>
  </data>
</pool>
```

Variable	Definition	Values
opaqueDataType	A user defined type/name for the opaque data	<ul style="list-style-type: none"> poolquota poolstate pooldynamicquota
cdataFieldValue	Contents of the XML data "blob"	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested opaque data exists for the pool
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found

Examples**Request 1**

A request is made to get the *poolquota* opaque data for a pool.

Request URL: `GET {baseURI}/msr/pool/100001/data/poolquota`

Request Content: None

Response 1

The request is successful, and the PoolQuota opaque data is returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="poolquota">
    <![CDATA[
      <?xml version="1.0" encoding="UTF-8"?>
      <usage>
```

```

<version>1</version>
<quota name="AggregateLimit">
  <cid>9223372036854775807</cid>
  <time>3422</time>
  <totalVolume>1000</totalVolume>
  <inputVolume>980</inputVolume>
  <outputVolume>20</outputVolume>
  <serviceSpecific>12</serviceSpecific>
  <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
</quota>
</usage>
]]>
</data>
</pool>

```

Request 2

A request is made to get the *poolstate* opaque data for a pool.

Request URL: **GET {baseURI}/msr/pool/100004/data/poolstate**

Request Content: None

Response 2

The request is successful, and the PoolState opaque data is returned.

HTTP Status Code: 200

Response Content:

```

<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="poolstate">
    <![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]>
  </data>
</pool>

```

Delete Opaque Data

Description

This operation deletes the opaque data of the specified *opaqueDataType* for the pool identified by the *poolId* in the request.

Only one opaque data type can be deleted per request.

Note: If the opaque data of the *opaqueDataType* does not exist for the pool, this is not considered an error and a successful result will be returned.

Prerequisites

A pool with the *poolId* supplied must exist.

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

Request URL

DELETE {baseURI}/msr/pool/*poolId*/data/*opaqueDataType*

Table 72: Request Variable Definitions: Delete Opaque Data

Variable	Definition	Values
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999
opaqueDataType	A user-defined type/name for the opaque data	<ul style="list-style-type: none"> • poolquota • poolstate • pooldynamicquota

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Data was successfully deleted
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined

Examples**Request 1**

A request is made to delete the *pooldynamicquota* opaque data.

Request URL: **DELETE** {baseURI}/msr/pool/500005/data/pooldynamicquota

Request Content: None

Response 1

The request is successful, and the PoolDynamicQuota opaque data was deleted.

HTTP Status Code: 204

Response Content: None

Request 2

A request is made to delete the *poolstate* opaque data. The PoolState opaque data is a valid entity, but the requested pool does not contain any PoolState opaque data.

Request URL: **DELETE** {baseURI}/msr/pool/600006//data/poolstate

Request Content: None

Response 2

The request fails because the specified pool does not contain PoolState data.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4058">errorText</error>
```

Additional Pool Commands

Table 73: Summary of Additional Pool Commands

Command	Description	Key	Command Syntax
Add Member to Pool	Add subscriber to a Pool	PoolID and (MSISDN, IMSI, NAI, or AccountId)	POST {BaseURI}/msr/pool/poolId/member/subKeyName/subKeyValue
Remove Member from Pool	Remove subscriber from a Pool		DELETE {BaseURI}/msr/pool/PoolID/member/subKeyName/subKeyValue
Get Pool Members	Retrieve pool member subscribers by PoolID	PoolID	GET {BaseURI}/msr/pool/poolId/member

Command	Description	Key	Command Syntax
Get PoolID	Retrieve PoolID for specified member subscriber	MSISDN, IMSI, NAI, or AccountId	<code>GET {BaseURI}/msr/sub/subKeyName/subKeyValue/pool</code>

Add Member to Pool

Description

This operation adds a subscriber to a pool.

Prerequisites

A pool with the *poolId* supplied must exist.

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

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

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

Request URL

`POST {BaseURI}/msr/pool/poolId/member/subKeyName/subKeyValue`

Table 74: Request Variable Definitions: Add Member to Pool

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

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Subscriber successfully added to pool
400	MSR4060	Number of pool members exceeded
404	MSR4001	Subscriber is not found
404	MSR4061	Specified pool does not exist
409	MSR4055	Subscriber is already a member of a pool

Examples**Request 1**

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

Request URL: **POST** {BaseURI}/msr/pool/100000/member/MSISDN/380561234567

Request Content: None

Response 1

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

HTTP Status Code: 204

Response Content: None

Request 2

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

Request URL: **POST** {BaseURI}/msr/pool/100009/member/IMSI/184569547984229

Request Content: None

Response 2

The request fails because the pool does not exist.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4061">errorText</error>
```

Request 3

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

Request URL: **POST** {BaseURI}/msr/pool/900000/member/NAI/mum@foo.com

Request Content: None

Response 3

The request fails because the subscriber does not exist.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4001">errorText</error>
```

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.

Request URL: **POST** {BaseURI}/msr/pool/100000/member/accountId/10404723525

Request Content: None

Response 4

The request fails because the subscriber is already a member of a pool.

HTTP Status Code: 409

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4055">errorText</error>
```

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 already has the maximum number of members allowed.

Request URL: **POST** {BaseURI}/msr/pool/100000/member/MSISDN/15141234567

Request Content: None

Response 5

The request fails because the pool has the maximum number of members allowed.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4060">errorText</error>
```

Remove Member from Pool

Description

This operation removes a subscriber from a pool.

Prerequisites

A pool with the *poolId* supplied must exist.

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

The subscriber must be a member of the specified pool.

Request URL

DELETE {baseURI}/msr/pool/poolId/member/subKeyName/subKeyValue

Table 75: Request Variable Definitions: Remove Member from Pool

Variable	Definition	Values
poolId	PoolID value of the pool. Numeric value, 1-22 digits in length	1-99999999999999999999999999999999
subKeyName	A key field within the subscriber Profile	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
subKeyValue	Corresponding key field value assigned to keyName	

Request Content

None

Response Content

None

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Subscriber successfully removed from pool
404	MSR4001	Subscriber is not found
404	MSR4061	Specified pool does not exist
404	MSR4062	Subscriber is not a member of the given pool

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.

Request URL: **DELETE** {baseURI}/msr/pool/100000/member/MSISDN/380561234567

Request Content: None

Response 1

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

HTTP Status Code: 204

Response Content: None

Request 2

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 the pool.

Request URL: **DELETE** {BaseURI}/msr/pool/100000/member/MSISDN/380561234567

Request Content: None

Response 2

The request fails because the subscriber is not a member of the pool.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4062">errorText</error>
```

Get Pool Members

Description

This operation gets the list of Subscriber members of a Pool by *poolId*.

Prerequisites

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

Request URL

GET {BaseURI}/msr/pool/*poolId*/member

Table 76: Request Variable Definition: Get Pool Members

Variable	Definition	Values
poolID	PoolID value of pool. Numeric value, 1-22 digits in length	1-99999999999999999999

Request Content

None

Response Content

A <members> element that contains a <member> element for every subscriber that is a member of the pool. The <member> element is optional. There can be zero, one, or many <member> elements. It is only present if the pool has member subscribers. One instance is present for every subscriber that is a member of the pool. A <member> element contains details about a single subscriber, containing all known user identities for that subscriber, one user identity per <id> element. There can be one or many <id> elements per <member> element.

```
<members>
[
  <member>
```

```

    <id><name>keyName1</name><value>keyValue1</value></id>
  [
    <id><name>keyName2</name><value>keyValue2</value></id>
    :
    <id><name>keyNameN</name><value>keyValueN</value></id>
  ]
</member>
]
[
  <member>
    <id><name>keyName1</name><value>keyValue1</value></id>
  [
    <id><name>keyName2</name><value>keyValue2</value></id>
    :
    <id><name>keyNameN</name><value>keyValueN</value></id>
  ]
  </member>
  :
  <member>
    <id><name>keyName1</name><value>keyValue1</value></id>
  [
    <id><name>keyName2</name><value>keyValue2</value></id>
    :
    <id><name>keyNameN</name><value>keyValueN</value></id>
  ]
  </member>
]
</members>

```

Table 77: Response Variable Definitions: Get Pool Members

Variable	Definition	Values
keyNameX	A key field for the member subscriber	<ul style="list-style-type: none"> • IMSI • MSISDN • NAI • AccountId
keyValueX	Corresponding key field value assigned to <i>keyNameX</i>	

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Pool exists, and membership returned OK
404	MSR4061	Specified pool does not exist

Examples

Request 1

A request is made to get the list of subscribers for a pool.

Request URL: **GET {BaseURI}/msr/pool/100000/member**

Request Content: None

Response 1

The request is successful and the three member subscribers are returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>NAI</name><value>user@domain</value></id>
  </member>
  <member>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
    <id><name>NAI</name><value>user3@domain3</value></id>
  </member>
</members>
```

Request 2

A request is made to get the list of subscribers for a pool. The pool exists, but has no member subscribers.

Request URL: **GET {BaseURI}/msr/pool/200000/member**

Request Content: None

Response 2

The request is successful, and no member subscribers are returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
</members>
```

Request 3

A request is made to get the list of subscribers for a pool. The pool does not exist.

Request URL: **GET {BaseURI}/msr/pool/300000/member**

Request Content: None

Response 3

The request fails because the pool was not found.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4061">errorText</error>
```

Get PoolID

Description

This operation gets the PoolID related to a subscriber, based on the given user identity of the subscriber.

Prerequisites

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

The subscriber must be a member of a pool.

Request URL

GET {baseURI}/msr/sub/*keyName*/*keyValue*/pool

Table 78: Request Variable Definitions: Get PoolID

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

Request Content

None

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="PoolID">poolId</field>
</pool>
```

Table 79: Response Variable Definition: Get PoolID

Variable	Definition	Values
poolId	PoolID value of pool. Numeric value, 1-22 digits in length	1-99999999999999999999999999999999

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Subscriber pool membership successfully returned
404	MSR4001	Subscriber is not found

HTTP Status Code	Error Code	Description
404	MSR4062	Subscriber is not a member of a pool

Example

Request 1

A request is made to get the PoolID for a subscriber. The subscriber is a member of a pool.

Request URL: **GET** {BaseURI}/msr/sub/MSISDN/380561234567/pool

Request Content: None

Response 1

The request is successful, and the PoolID value was returned.

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
<field name="poolId">100000</field>
</pool>
```

Request 2

A request is made to get the PoolID for a subscriber. The subscriber is NOT a member of a pool.

Request URL: **GET** {BaseURI}/msr/sub/NAI/joe@foo.com/pool

Request Content: None

Response 2

The request fails, because the subscriber is not a member of a pool.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4062">errorText</error>
```


Appendix

A

REST Interface System Variables

Topics:

- [REST Interface System Variables.....154](#)

REST Interface System Variables

The REST Interface has a set of system variables that affect its operation as it runs. REST Interface System variables (shown below in [Table 80: REST Interface System Variables](#)) can be set via the UDR GUI [T8] and can be changed at runtime to effect dynamic server reconfiguration.

Table 80: REST Interface System Variables

Parameter	Description
REST Interface Port	XML-REST Interface TCP Listening Port. NOTE: Changes to the TCP listening port do not take effect until the 'ras' process is restarted. Also, you must specify a different port than the SOAP/XML interface. DEFAULT = 8787; RANGE = 0-65535
REST Interface Idle Timeout	The maximum time (in seconds) that an open XML-REST connection will remain active without a request being sent, before the connection is dropped. DEFAULT = 1200; RANGE = 1-86400
Maximum XML-REST Connections	Maximum number of simultaneous XML-REST Interface client connections. DEFAULT = 100; RANGE = 1-100
Allow REST Connections	Whether or not to allow incoming connections on the XML-REST Interface. DEFAULT = CHECKED
XML-REST Secure Mode	Whether the REST Interface operates in secure mode (using SSL) or unsecure mode (plain text). NOTE: Changes to the Secure Mode do not take effect until the 'ras' process is restarted. DEFAULT = Unsecure
Transaction Durability Timeout*	The amount of time (in seconds) allowed between a transaction being committed and it becoming durable. If Transaction Durability Timeout lapse, DURABILITY_TIMEOUT response is sent to the originating client. The associated request should be resent to ensure that the request was committed. DEFAULT = 5; RANGE = 2-3600
Compatibility Mode*	Indicates whether backwards compatibility is enabled. Note: Change to Compatibility Mode may cause the existing provisioning connections to be dropped. DEFAULT = R10+

Note: Parameters labeled with an "*" are existing system variables defined and used by other components of UDR.

Appendix B

Legacy SPR Compatibility Mode

Topics:

- [Get Row Response Format.....156](#)

UDR can be configured to run in a compatibility mode with the legacy SPR.

When the **Compatibility Mode** system option, which is configurable by the UDR GUI [T8], is set to "R10+" then UDR will behave as described in the main body of this document. When **Compatibility Mode** is set to "R9.x", then the differences contained in this appendix will apply.

Enabling this configuration option results in the format of some request/responses being different to the default UDR behavior. This appendix lists the different request/responses to which enabling the option applies.

Get Row Response Format

UDR returns the data row in the format in which it is defined/stored (either "Field Based" or "Element Based"). The legacy SPR returns a (Quota) data row in "Element Based" format, even though the Quota entity is "Element Based".

When configured in legacy SPR mode, UDR returns the (Quota) data row in "Field Based" format, within the CDATA. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
<version>1</version>
<field name="Cid"/>
<field name="Time"/>
<field name="totalVolume">0</field>
<field name="inputVolume">0</field>
<field name="outputVolume">0</field>
<field name="serviceSpecific"/>
<field name="nextResetTime"/>
<field name="Type">quota</field>
<field name="GrantedTotalVolume">0</field>
<field name="GrantedInputVolume">0</field>
<field name="GrantedOutputVolume">0</field>
<field name="GrantedTime"/>
<field name="GrantedServiceSpecific"/>
<field name="QuotaState">Valid/Inactive</field>
<field name="RefInstanceId"/>
<field name="Name">test</field>
</usage>
]]>
</data>
</subscriber>
```

If more than one matching row is found, then multiple **<quota>** rows are returned. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
<version>3</version>
:
</usage>
]]>
</data>
</subscriber>

<subscriber>
<data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
<version>3</version>
:
```

```
</usage>  
  ]>  
</data>  
</subscriber>
```

#

3GPP
3rd Generation Partnership Project.
The standards body for wireless communications.

A

ACID
Atomicity, Consistency, Isolation and Durability

AuC
Authentication Center

B

blob
binary large object
A collection of binary data stored as a single entity in the Subscription Profile Repository.

C

CA
Certificate Authority: An entity that issues digital certificates

CPS
Customer Provisioning System

E

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

G

GUI
Graphical User Interface

G

The term given to that set of items and facilities which provide the user with a graphic means for manipulating screen data rather than being limited to character based commands.

H

HLR

Home Location Register

A component within the Switching Subsystem of a GSM network. The HLR database is the central database within the GSM architecture. This is where information about the mobile communications subscribers who are assigned to a specific location area is stored. The subscriber data is used to establish connections and control services. Depending on the network size, the number of subscribers and the network organization, a number of HLRs can exist within a GSM network.

HSS

Home Subscriber Server

A central database for subscriber information.

I

IP

Internet Protocol - IP specifies the format of packets, also called datagrams, and the addressing scheme. The network layer for the TCP/IP protocol suite widely used on Ethernet networks, defined in STD 5, RFC 791. IP is a connectionless, best-effort packet switching protocol. It provides packet routing, fragmentation and re-assembly through the data link layer.

I

IPFE

IP Front End

A traffic distributor that routes TCP traffic sent to a target set address by application clients across a set of application servers. The IPFE minimizes the number of externally routable IP addresses required for application clients to contact application servers.

L

LDAP

Lightweight Directory Access Protocol

A protocol for providing and receiving directory information in a TCP/IP network.

M

Message Processor

See MP

MP

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

MSISDN

Mobile Station International Subscriber Directory Number. The unique, network-specific subscriber number of a mobile communications subscriber. MSISDN follows the E.164 numbering plan; that is, normally the MSISDN is the phone number that is used to reach the subscriber.

N

N

NAI	Network Access Identifier The user identity submitted by the client during network authentication.
-----	---

O

OAMP	Operations, Administration, Maintenance and Provisioning
OSS	Operations Support System Computer systems used by telecommunications service providers, supporting processes such as maintaining network inventory, provisioning services, configuring network components, and managing faults.

P

pass	A quota profile that provides a one-time override of a subscriber's default plan.
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.

PEM	Privacy Enhanced Mail
-----	-----------------------

Q

Q

quota
Specifies restrictions on the amount of data volume, active session time, or service-specific events that a subscriber can consume.

R

RAS
REST Application Server

REST
Representational State Transfer - used by the provisioning system to send HTTP requests (GET, POST, PUT) to manipulate and query data in the provisioning database.

rollover
A quota convention that allows a subscriber to carry forward unused units from one billing cycle to another.

RSA
The Rivest-Shamir-Adleman Algorithm for public-key encryption developed by Ron Rivest, Adi Shamir, and Leonard Adleman.

S

SDO
Subscriber Data Object
Subscription Data Object. An SDO consists of subscription state information and a collection of registers for storing entities. An individual SDO applies to one subscriber. A pool SDO applies to a group of subscribers.

SIP
Session Initiation Protocol
A peer-to-peer protocol used for voice and video communications.

S

SOAM	System Operations, Administration, and Maintenance
SOAP	Simple Object Access Protocol
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.
SSL	Secure Socket Layer (SSL) is an industry standard protocol for clients needing to establish secure (TCP-based) SSL-enabled network connections

T

TCP	Transmission Control Protocol A connection-oriented protocol used by applications on networked hosts to connect to one another and to exchange streams of data in a reliable and in-order manner.
top-up	A quota convention that allows a subscriber to obtain additional units for an existing plan.

U

UDR	User Data Repository - A logical entity containing user data
-----	--

U

UTC

Coordinated Universal Time

V

VIP

Virtual IP Address

Virtual IP is a layer-3 concept employed to provide HA at a host level. A VIP enables two or more IP hosts to operate in an active/standby HA manner. From the perspective of the IP network, these IP hosts appear as a single host.