

User Data Repository REST Provisioning Interface Specification

Release 12.2

E71788-03

June 2018



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

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

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

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

See more information on My Oracle Support in Appendix C.

Copyright ©2014, 2018, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

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

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

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

Table of Contents

1. INTRODUCTION	10
1.1 Purpose and Scope.....	10
1.2 References.....	10
1.3 Glossary.....	10
2. SYSTEM ARCHITECTURE	12
2.1 Overview.....	12
2.2 Provisioning Interface.....	13
2.3 REST Application Server (RAS).....	14
2.4 Provisioning Clients.....	14
2.5 Security.....	14
2.5.1 Client Server IP Address White List	14
2.5.2 Secure Connection using TLS.....	15
2.5.2.1 TLS Certificates and Public/Private Key Pairs	15
2.5.2.2 Supported TLS Cipher Suites.....	16
2.6 Multiple Connections.....	17
2.7 Request Queue Management	17
2.8 Database Transactions	18
2.8.1 ACID-Compliance.....	18
2.8.1.1 Atomicity.....	18
2.8.1.2 Consistency.....	18
2.8.1.3 Isolation	18
2.8.1.4 Durability.....	18
2.9 Connection Management	18
2.9.1 Connections Allowed	19
2.9.2 Disable Provisioning	19
2.9.3 Idle Timeout.....	19
2.9.4 Maximum Simultaneous Connections	19
2.9.5 TCP Port Number	19
2.10 Behavior during Low Free System Memory.....	19
2.11 Congestion Control	19
2.12 Pools Spanning UDRs	20
2.13 Enterprise Pools	20
2.14 Rest Conventions.....	20
2.14.1 HTTP(S) Request Headers	21
2.14.1.1 HTTP version.....	21

2.14.1.2	Accept Header	21
2.14.1.3	Transfer-Encoding Header	21
2.14.1.4	Requests with body content	21
2.14.2	HTTP(S) Status Codes and Error Messages	21
3.	REST INTERFACE MESSAGE DEFINITIONS	25
3.1	Message Conventions	25
3.1.1	HTTP Method	25
3.1.2	Base URI	25
3.1.3	REST URL	25
3.1.3.1	Subscriber or Pool in URL	25
3.1.3.2	Opaque Data Operations in URL	25
3.1.3.3	Field in URL	25
3.1.3.4	Transparent Data Row Operations in URL	26
3.1.3.5	Transparent Data Row Operations using an Instance Identifier in URL	26
3.1.3.6	Transparent Data Row Field Operations in URL	26
3.1.3.7	Transparent Data Row Field Operations using an Instance Identifier in URL	26
3.1.3.8	Transparent Data Field Operations in URL	26
3.1.4	URL Character Encoding	26
3.2	Case Sensitivity	27
3.3	XML Comments in a Request	27
3.4	Request Content in a Request	28
3.5	List of Messages	28
4.	UDR DATA MODEL	32
4.1	Subscriber Data	34
4.1.1	Subscriber Profile	34
4.1.2	Quota	36
4.1.3	State	38
4.1.4	Dynamic Quota	39
4.2	Pool Data	40
4.2.1	Pool Profile	40
4.2.2	Pool Quota	41
4.2.3	Pool State	42
4.2.4	Pool Dynamic Quota	42
4.3	Date/Timestamp Format	42
5.	SUBSCRIBER PROVISIONING	44
5.1	Subscriber Profile Commands	44

5.1.1 Create Subscriber.....	44
5.1.2 Get Profile.....	49
5.1.3 Update Profile.....	51
5.1.4 Delete Profile.....	53
5.2 Subscriber Profile Field Commands.....	55
5.2.1 Add Field Value.....	55
5.2.2 Get Field.....	59
5.2.3 Get Field Value.....	61
5.2.4 Update Field.....	63
5.2.5 Update Multiple Fields.....	67
5.2.6 Delete Field.....	69
5.2.7 Delete Field Value.....	72
5.3 Subscriber Opaque Data Commands.....	74
5.3.1 Set Opaque Data.....	75
5.3.2 Get Opaque Data.....	78
5.3.3 Delete Opaque Data.....	80
5.4 Subscriber Data Row Commands.....	82
5.4.1 Set Row.....	83
5.4.2 Get Row.....	88
5.4.3 Delete Row.....	92
5.5 Subscriber Data Row Field Commands.....	96
5.5.1 Get Row Field.....	97
5.5.2 Get Row Field Value.....	100
5.5.3 Update Row Field.....	105
5.5.4 Delete Row Field.....	109
5.6 Subscriber Data Field Commands.....	112
5.6.1 Set Data Field.....	113
5.6.2 Get Data Field.....	116
5.6.3 Delete Data Field.....	119
5.7 Subscriber Special Operation Commands.....	122
5.7.1 Reset Quota.....	122
6. POOL PROVISIONING.....	127
6.1.1 Create Pool.....	127
6.1.2 Get Pool.....	130
6.1.3 Update Pool.....	132
6.1.4 Delete Pool.....	134
6.2 Pool Profile Field Commands.....	136

6.2.1 Add Field Value	136
6.2.2 Get Field	138
6.2.3 Get Field Value	140
6.2.4 Update Field	142
6.2.5 Update Multiple Fields	144
6.2.6 Delete Field.....	145
6.2.7 Delete Field Value	147
6.3 Pool Opaque Data Commands.....	150
6.3.1 Set Opaque Data.....	150
6.3.2 Get Opaque Data	153
6.3.3 Delete Opaque Data.....	156
6.4 Pool Data Row Commands	158
6.4.1 Set Row	158
6.4.2 Get Row.....	162
6.4.3 Delete Row	167
6.5 Pool Data Row Field Commands.....	169
6.5.1 Get Row Field.....	170
6.5.2 Get Row Field Value.....	174
6.5.3 Update Row Field	178
6.5.4 Delete Row Field	181
6.6 Pool Data Field Commands.....	184
6.6.1 Set Data Field.....	185
6.6.2 Get Data Field.....	188
6.6.3 Delete Data Field.....	191
6.7 Additional Pool Commands	193
6.7.1 Add Member to Pool.....	194
6.7.2 Remove Member from Pool.....	197
6.7.3 Get Pool Members.....	199
6.7.4 Get Pool ID	202
6.7.5 Get All Pool Members.....	203
6.8 Pool Special Operation Commands.....	211
6.8.1 Reset Pool Quota	211

List of Figures

Figure 1: User Data Repository High Level Architecture	13
Figure 2: Data Model	34

List of Tables

Table 1: Glossary	10
Table 2: TLS X.509 Certificate and Key PEM-encoded Files.....	16
Table 3: TLS Supported Cipher Suites.....	17
Table 4: HTTP and HTTPS Status Codes.....	22
Table 5: Error Codes	22
Table 6: Summary of Subscriber Commands.....	29
Table 7: Summary of Pool Commands	30
Table 8: Subscriber Profile Entity Definition	35
Table 9: Quota Entity Definition.....	37
Table 10: State Entity Definition.....	38
Table 11: Dynamic Quota Entity Definition	39
Table 12: Pool Profile Entity Definition	40
Table 13: Summary of Subscriber Profile Commands	44
Table 14: Create Subscriber Response Status/Error Codes.....	45
Table 15: Get Profile Response Status/Error Codes	50
Table 16: Update Profile Response Status/Error Codes.....	52
Table 17: Delete Profile Response Status/Error Codes	54
Table 18: Summary of Subscriber Profile Field Commands	55
Table 19: Add Field Value Response Status/Error Codes	56
Table 20: Response Status/Error Codes	59
Table 21: Get Field Value Response Status/Error Codes.....	62
Table 22: Update Field Response Status/Error Codes.....	64
Table 23: Update Multiple Fields Response Status/Error Codes.....	68
Table 24: Delete Field Response Status/Error Codes	70
Table 25: Delete Field Value Response Status/Error Codes.....	73
Table 26: Summary of Subscriber Opaque Data Commands	74
Table 27: Set Opaque Data Response Status/Error Codes.....	76
Table 28: Get Opaque Data Response Status/Error Codes	78
Table 29: Delete Opaque Data Response Status/Error Codes	81
Table 30: Summary of Subscriber Data Row Commands.....	83
Table 31: Set Row Response Status/Error Codes	84
Table 32: Get Row Response Status/Error Codes	89
Table 33: Delete Row Response Status/Error Codes.....	93
Table 34: Summary of Subscriber Data Row Field Commands	96
Table 35: Get Row Field Response Status/Error Codes.....	98
Table 36: Get Row Field Value Response Status/Error Codes.....	102
Table 37 Update Row Field Response Status/Error Codes.....	106
Table 38: Delete Row Field Response Status/Error Codes.....	110
Table 39: Summary of Subscriber Data Field Commands	112
Table 40: Set Data Field Response Status/Error Codes	114
Table 41: Get Data Field Response Status/Error Codes	117
Table 42: Delete Data Field Response Status/Error Codes.....	120
Table 43: Summary of Subscriber Special Operation Commands.....	122
Table 44: Reset Quota Response Status/Error Codes	123

Table 45: Summary of Pool Profile Commands	127
Table 46: Create Pool Response Status/Error Codes	128
Table 47: Get Pool Response Status/Error Codes	131
Table 48: Update Pool Response Status/Error Codes	133
Table 49: Delete Pool Response Status/Error Codes.....	135
Table 50: Summary of Pool Profile Field Commands	136
Table 51: Add Field Value Response Status/Error Codes	137
Table 52: Get Field Response Status/Error Codes.....	139
Table 53: Update Field Response Status/Error Codes.....	143
Table 54: Update Multiple Fields Response Status/Error Codes.....	145
Table 55: Delete Field Response Status/Error Codes	146
Table 56: Delete Field Value Response Status/Error Codes.....	148
Table 57: Summary of Pool Opaque Data Commands	150
Table 58: Set Opaque Data Response Status/Error Codes.....	151
Table 59: Get Opaque Data Response Status/Error Codes	154
Table 60: Delete Opaque Data Response Status/Error Codes	157
Table 61: Summary of Pool Data Row Commands.....	158
Table 62: Set Row Response Status/Error Codes	160
Table 63: Get Row Response Status/Error Codes	163
Table 64: Delete Row Response Status/Error Codes.....	168
Table 65: Summary of Pool Data Row Field Commands	169
Table 66: Get Row Field Response Status/Error Codes.....	171
Table 67: Get Row Field Value Response Status/Error Codes.....	175
Table 68: Update Row Field Response Status/Error Codes.....	179
Table 69: Delete Row Field Response Status/Error Codes	183
Table 70: Summary of Pool Data Field Commands	185
Table 71: Set Data Field Response Status/Error Codes	186
Table 72: Get Data Field Response Status/Error Codes	189
Table 73: Delete Data Field Response Status/Error Codes.....	192
Table 74: Summary of Additional Pool Commands.....	193
Table 75: Add Member to Pool Response Status/Error Codes	195
Table 76: Remove Member from Pool Response Status/Error Codes	198
Table 77: Get Pool Members Response Status/Error Codes.....	200
Table 78: Get Pool ID Response Status/Error Codes	202
Table 79: Get All Pool Members Response Status/Error Codes.....	205
Table 80: Summary of Pool Special Operation Commands.....	211
Table 81: Reset Pool Quota Response Status/Error Codes	212
Table 82: REST Interface System variables.....	216

1. INTRODUCTION

1.1 Purpose and Scope

This document presents the REST Provisioning interface to be used by 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, or delete subscriber/pool information in the UDR database.

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

1.2 References

These external document references list the source material used to create this document.

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

1.3 Glossary

This section lists terms and acronyms specific to this document.

Table 1: Glossary

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

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

2. SYSTEM ARCHITECTURE

2.1 Overview

Oracle Communications User Data Repository (UDR) performs the function of an SPR, which is a database system that acts as a single logical repository that stores subscriber data. The subscriber data that traditionally was stored in the HSS, HLR, AuC, or Application Servers is stored in UDR as specified in 3GPP UDC information model [3]. UDR facilitates the sharing and provisioning of user related data throughout services of 3GPP system. Several applications front ends, such as one or more PCRF, HSS, HLR, or AuCFEs can be served by UDR.

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

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

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

Figure 1 illustrates a high level the UDR Architecture.

As illustrated 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 UDR database resides on the NOAM&P servers. This release focuses on the development of the Sh messaging interface for use with the UDR application.

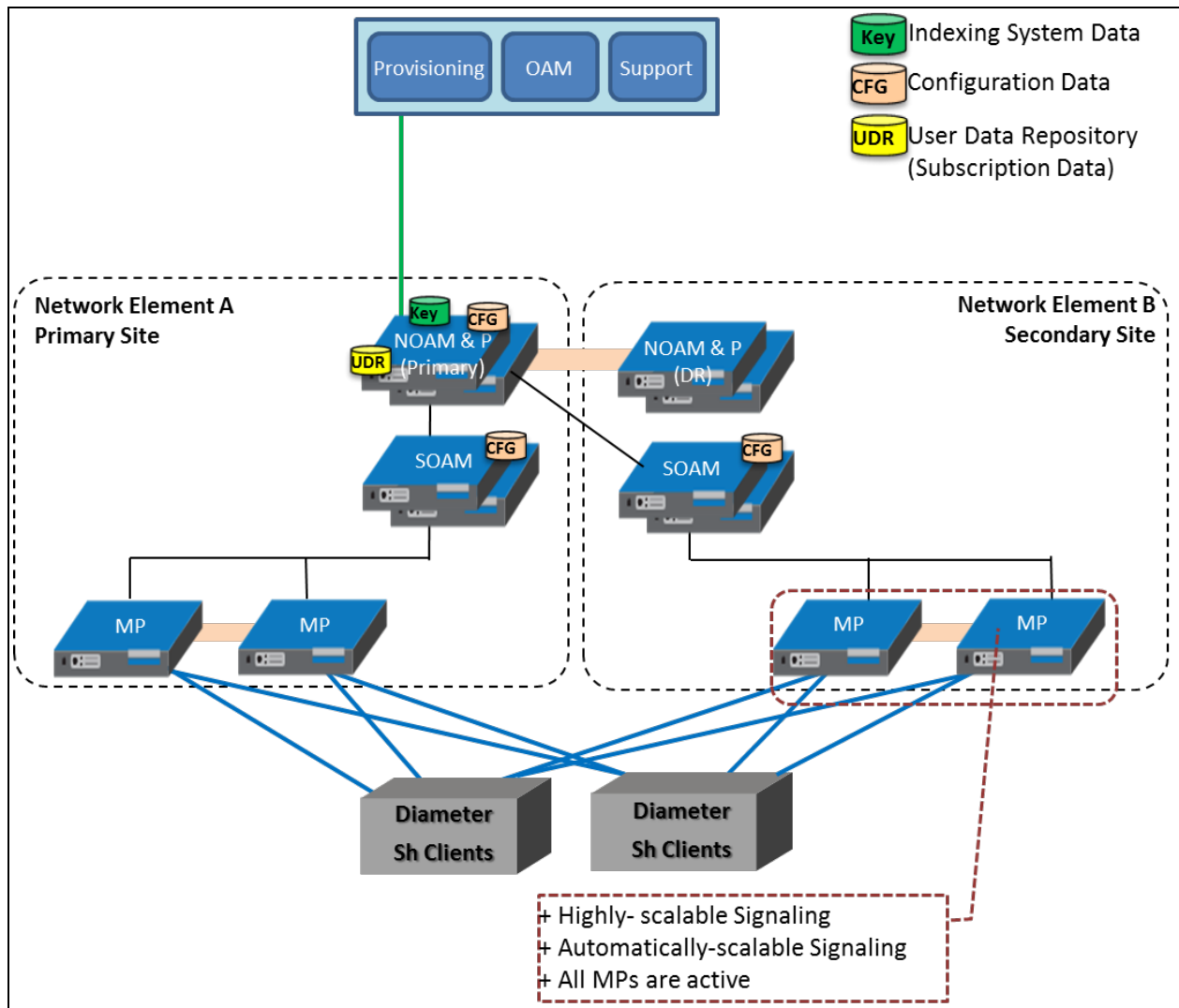
As the product evolves forward, the Subscriber Profiles in UDR can be expanded to support data associated with additional applications. Along with that, the MPs can be expanded to support additional Diameter interfaces associated with these applications. The IPFE can be integrated with the product to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

The Network level OAMP server (NOAM&P) in Figure 1 provides the provisioning, configuration and maintenance functions for all the network elements under it.

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

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

Figure 1: UDR High Level Architecture



2.2 Provisioning Interface

The REST provisioning interface provides data manipulation commands for subscriber and pool:

Subscriber:

- Subscriber Profile create, retrieve, modify, and delete
- Subscriber Profile field add, retrieve, modify, and delete
- Subscriber opaque data create, retrieve, modify, and delete

Quota, State, and Dynamic Quota

- Subscriber transparent data create, retrieve, modify, and delete

Quota, State, and Dynamic Quota

- Reset of a row in Subscriber Quota transparent data

Pool:

- Pool Profile create, retrieve, modify, and delete
- Pool Profile field add, retrieve, modify, and delete

- Pool opaque data create, retrieve, modify, and delete
Pool Quota, Pool State and Pool Dynamic Quota
- Pool transparent data create, retrieve, modify, and delete
Pool Quota, Pool State and Pool Dynamic Quota
- Reset of a row in Pool Quota transparent data
- Pool subscriber membership operations
 - Add or remove from pool
 - Get pool subscriber membership
 - Get pool for subscriber

2.3 REST Application Server (RAS)

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

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

2.4 Provisioning Clients

The RAS provides connections to 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, 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 VIP for the primary NOAM&P.

Provisioning clients must re-establish connections with the RAS using the VIP for the primary UDR when switching over from the active primary to the standby UDR server. Provisioning clients also redirect connections to the VIP for the Secondary after switchover from the primary UDR site to the Disaster Recover UDR site.

If a response is not sent, provisioning clients must run a timeout for the response to a request. If a response is not received, a client drops the connection and re-establishes it before trying again.

Provisioning clients are expected to re-send requests that resulted in a temporary error, or for which a response was not received.

2.5 Security

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

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

2.5.1 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 and/or IP address ranges from which clients are authorized to establish a TCP/IP connection from.

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

2.5.2 Secure Connection using TLS

The RAS supports secure (encrypted) connections between provisioning clients and the RAS using Transport Layer Security version 1.0 (TLSv1.0) protocol implemented using OpenSSL based on SSLeay library developed by Eric A. Young and Tim J. Hudson.

TLS is an industry standard protocol for clients needing to establish secure (TCP-based) TLS-enabled network connections. TLS 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 capabilities of TLS address several fundamental concerns about communication over TCP/IP networks:

- TLS server authentication

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

Allows a server application to confirm the identity of the client application. The server application through TLS uses standard public-key cryptography to verify that the certificate and public key for the client are valid and has been signed by a trusted certificate authority (CA) that is known to the server application.
- An encrypted TLS 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, TLS 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 on the mode the RAS is configured to operate in (secure/unsecure), provisioning clients can connect using unsecure or secure connections to the well-known TCP/TLS listening port for the RAS (configured using the REST Secure Mode configuration variable using the UDR GUI).

NOTE: A TLS-enabled connection is slower than an unsecure TCP/IP connection. This is a direct result of providing adequate security. On a TLS-enabled connection, more data is transferred than normal. Data is transmitted in packets, which contain information required by the TLS 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.

2.5.2.1 TLS Certificates and Public/Private Key Pairs

TLS-enabled connections require TLS 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 has the public key for the owner. 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.

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

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

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

Provisioning clients are required to send a TLS 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 or signed by Oracle Certification Authority (CA), it is not be able to establish a secure connection with the RAS.

2.5.2.2 Supported TLS Cipher Suites

A cipher suite is a set (combination) of lower-level algorithms that a TLS-enabled connection uses to do authentication, key exchange, and stream encryption. Table 3 lists the set of TLS cipher suites from the relevant specification and their OpenSSL equivalents that are supported by the RAS to secure a TLS-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 TLS-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 (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 security vulnerabilities.

Table 3: TLS Supported Cipher Suites

Cipher Suite (RFC)	OpenSSL Equivalent	Key Exchange	Signing/ Authentication	Encryption (Bits)	MAC (Hash) Algorithms
TLS_RSA_WITH_AES_256_CBC_SHA	AES256-SHA	RSA	RSA	AES (256)	SHA-1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	DES-CBC3-SHA	RSA	RSA	3DES(168)	SHA-1
TLS_RSA_WITH_AES_128_CBC_SHA	AES128-SHA	RSA	RSA	AES(128)	SHA-1
TLS_KRB5_WITH_RC4_128_SHA	KRB5-RC4-SHA	KRB5	KRB5	RC4(128)	SHA-1
TLS_RSA_WITH_RC4_128_SHA	RC4-SHA	RSA	RSA	RC4(128)	SHA-1
TLS_KRB5_WITH_3DES_EDE_CBC_SHA	KRB5-DES-CBC3-SHA	KRB5	KRB5	3DES(168)	SHA-1

2.6 Multiple Connections

The RAS supports multiple connections and each connection is considered persistent unless declared otherwise. The HTTP persistent connections do not use separate keep-alive messages, they just allow multiple requests to use a same TCP/IP connection. However, connections are closed after being idle for a time limit configured in idle timeout (See section 2.9.3).

If the client does not want to maintain a connection for more than that request, it sends 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 TCP/IP connection to RAS before sending the first REST command. After the execution of the request, the RAS sends a response message back and keeps the connection alive as long as a 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 Appendix A) 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.

2.7 Request Queue Management

If multiple clients simultaneously issues requests, each request is queued and processed in the order in which it was received on a per connection basis. The client 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 run in parallel but requests on each connection are still processed serially.

2.8 Database Transactions

Each create, update, delete, or request coming from the REST interface triggers a unique database transaction. A database transaction started by a request is committed before sending a response.

2.8.1 ACID-Compliance

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

2.8.1.1 Atomicity

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

2.8.1.2 Consistency

Data across all requests performed inside a transaction is consistent.

2.8.1.3 Isolation

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

2.8.1.4 Durability

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

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

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

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

2.9 Connection Management

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

2.9.1 Connections Allowed

The configuration variable `Allow REST Provisioning Connections` (see Appendix A) 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. Any attempts to connect are rejected.

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

2.9.2 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 is rejected with a `SERVICE_UNAVAILABLE` error indicating the service is unavailable.

For an example of a provisioning request/response when provisioning is disabled, see the last example in section 6.1.1.

2.9.3 Idle Timeout

HTTP connection between Provisioning client and RAS is handled persistent fashion. The configuration variable `REST Interface Idle Timeout` (see Appendix A) indicates the time to wait before closing the connection due to inactivity (requests are not received).

2.9.4 Maximum Simultaneous Connections

The configuration variable `Maximum REST Connections` (see Appendix A) defines the maximum number of simultaneous REST interface client connections.

2.9.5 TCP Port Number

The configuration variable `REST Interface Port` (see Appendix A) defines the REST interface TCP listening port.

2.10 Behavior during Low Free System Memory

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

HTTP Status Code

The error returned by the REST interface when the critical level has been reached is 507.

Response Content

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

2.11 Congestion Control

If UDR starts to encounter congestion (based on high CPU usage), then based on the congestion level, UDR rejects requests (based on the HTTP method, see section 4.1.1).

- If the minor CPU usage threshold is crossed (CL1), then UDR rejects GET requests
- If the major CPU usage threshold is crossed (CL2), then UDR rejects GET and PUT requests
- If the critical CPU usage threshold is crossed (CL3), then UDR rejects all requests

HTTP Status Code

The error returned by the REST interface when a request is rejected due to congestion is 503.

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4097">errorText</error>
```

2.12 Pools Spanning UDRs

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

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

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

2.13 Enterprise Pools

Enterprise Pools have the capability to support more than 25 members in a pool. Basic Pools maintain a threshold of 25 members as the maximum number of subscribers that are allowed. Enterprise Pools are pools containing more than 25 members and there is not a maximum number of pool members enforced.

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

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

REST Interface Description

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 or HTTPS interface that is designed based on RESTful concepts. This section defines the operations that can be performed using the REST interface.

2.14 Rest Conventions

The REST interface uses these RESTful concepts:

- HTTP or HTTPS headers
- HTTP or HTTPS status codes
- Error message representation in the response content for all 4xx and 5xx codes.

2.14.1 HTTP(S) Request Headers

These HTTP and HTTPS requirements must be followed.

2.14.1.1 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 TLS v1
```

2.14.1.2 Accept Header

Set the Accept header property to the correct MIME version using the one of these formats:

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

Note: The Accept header is optional, and if omitted the value is treated as if the value **/** was supplied.

2.14.1.3 Transfer-Encoding Header

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

```
Transfer-Encoding: chunked
```

2.14.1.4 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"?>
```

2.14.2 HTTP and HTTPS Status Codes and Error Messages

The REST interface uses standard HTTP and HTTPS 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 includes response content that has an error Message entity.

Table 4 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 and HTTPS Status Codes

Status Code	Description
200—OK	Indicates the successful completion of request processing.
201—Created	Used for created entities.
204—No Content	The request completed successfully and response content body is not sent back to the client.
400—Bad Request	This indicates 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 more data.

Besides the HTTP status codes, additional error codes (Table 5) are provided for the 4xx and 5xx range of Status Codes.

NOTE: The Description column is for reference only, it is not included in the HTTP response. Additional text may be included in the HTTP response.

Table 5: Error Codes

Error Code	Description
MSR4000	Invalid content request data supplied
MSR4001	Subscriber/pool not found
MSR4002	Subscriber/pool/data field is not defined
MSR4003	A key is detected in the system for another subscriber/pool
MSR4004	Unique key not found for subscriber/pool

Error Code	Description
MSR4005	Field does not support multiple values and value for field exists
MSR4049	Data type is not defined
MSR4050	Unknown key, the key provided in the request is invalid
MSR4051	The value provided for the field is invalid
MSR4053	Subscriber/pool exist, but the field value is incorrect
MSR4055	Subscriber is a member of a pool
MSR4056	Field is not updatable
MSR4057	Request only contains one field to update
MSR4058	Data type not found
MSR4059	Data row does not exist
MSR4060	Number of pool members exceeded
MSR4061	Specified pool does not exist
MSR4062	Subscriber is not a member of the pool
MSR4063	Entity cannot be reset
MSR4064	Occurrence constraint violation
MSR4065	Field is not set
MSR4066	Field value exists
MSR4067	Multiple matching rows found
MSR4068	Free system memory is low
MSR4069	At least one key is required
MSR4070	Operation not allowed
MSR4097	Request rejected due to system congestion
MSR4098	Provisioning is disabled

Error Code	Description
MSR4099	Unexpected server error has occurred
MSR4100	Maximum number of Subscribers in a Basic Pool has been exceeded
MSR4101	Enterprise to Basic Pool Conversion failed threshold exceeded
MSR4102	Provisioning Request Timeout because a response was not received from remote UDR
MSR4103	A key is detected in the system for an AE subscriber

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: The examples in the this document do not contain the error text associated with the MSRxxxx because this information varies depending on the entity, key, or field values used.

3. REST INTERFACE MESSAGE DEFINITIONS

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

3.1 Message Conventions

3.1.1 HTTP Method

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

3.1.2 Base URI

The base URI (*baseURI*) that is the prefix for the documented URIs uses this 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 UDR is installed in the network.

For example, if 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
```

3.1.3 REST URL

The REST interface uses the XML conventions (sections 3.1.3.1 to 3.1.3.8) in the REST command URL.

3.1.3.1 *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
```

3.1.3.2 *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 XML blob creating, getting, or deleting the blob.

For example when deleting the Quota data for a subscriber:

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

3.1.3.3 *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
```

3.1.3.4 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
```

3.1.3.5 Transparent Data Row Operations using an Instance Identifier in URL

For transparent data row based operations using an instance identifier the keywords *data* and *row* are 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. The instance identifier specified is the unique identifier used to identify the transparent object.

For example when deleting a row in Quota data for a subscriber using cid as an instance identifier.

```
DELETE {baseURI}/msr/sub/IMSI/302370123456789/data/quota/10GBMonth/row/cid/9223372036854775807
```

3.1.3.6 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
```

3.1.3.7 Transparent Data Row Field Operations using an Instance Identifier in URL

For transparent data row field based operations using an instance identifier the keyword *data* and *row* are 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. The instance identifier specified is the unique identifier used to identify the transparent object.

For example when deleting a row field in Quota data for a subscriber using cid as an instance identifier.

```
DELETE {baseURI}/msr/sub/IMSI/302370123456789/data/quota/10GBMonth/row/cid/9223372036854775807/totalVolume
```

3.1.3.8 Transparent Data Field Operations in URL

For transparent data field based operations only the keyword *data* is used. The data type indicated in the URL can be any valid transparent data type which has fields defined as a name value pair in an element. The data field name and value are also supplied.

For example when deleting a data field in State data for a subscriber.

```
DELETE {baseURI}/msr/sub/IMSI/302370123456789/data/state/mcc/315
```

3.1.4 URL Character Encoding

Restricted character encoding is allowed in the URL using the % (percent) character. Foreexample, the %3B for to indicate a ; (semicolon) character. However, it is not permitted to use double encoding such as %253B in order to first quote the % (percent) character.

3.2 Case Sensitivity

The URL constructs that REST requests are made up of (that is, msr, sub, pool, field, data, multipleFields) are case-sensitive. Exact case must be used for all the commands described in this document, or the request fails.

For example, the URL is valid:

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

But this URL is not:

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

Key names, and entity field names are not case-sensitive, for example *keyName*, *fieldName* and *setFieldName*.

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

Entity names as specified in an *opaqueDataName* or *transparentDataType* are not case sensitive.

Examples:

- When accessing a *fieldName* defined as *inputVolume* in the SEC, then *inputvolume*, *INPUTVOLUME* or *inputVolume* are valid field names. Field names do not have to be specified in a request as they are defined in the SEC.
- When a field is returned in a response, it is returned as defined in the SEC. For example, if the above field is created using the name *INPUTVOLUME*, then it is returned in a response as *inputVolume*.
- When a *fieldValue* is used to find a field (such as when using the **Delete Field Value** command), the field value is case-sensitive. If a multi-value field contained the values *DayPass*, *Weekend*, *Evening* and the **Delete Field Value** command was used to delete the value *WEEKEND*, then this fails.
- When an attribute in the XML blob contains the row identifier name (*rowIdName*), for example for *Quota*, the `<quota name="AggregateLimit">` element contains the attribute called *name* the row identifier name is not case-sensitive.
- When a *rowIdValue* is used to find a row (such as when using the **Get Row** command), the row identifier value is case-sensitive. If an entity contained a row called *DayPass*, and the **Get Row** command was used to get the row *DAYPASS*, then this fails.
- 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* does not find the subscriber.

3.3 XML Comments in a Request

A REST request may not contain XML comments in the request or the content of the body. For example:

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

HTTP Status Code

If a request contains a comment, the request is rejected with the error 400.

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4000">errorText</error>
```

3.4 Request Content in a Request

Some REST requests do not require Request Content to be provided in the body of the HTTP request. If Request Content is provided when it is not required, the Request Content is ignored and the request is processed as normal.

Examples

Request 1

Request URL

```
GET {baseURI}/msr/sub/AccountId/10404723525
```

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</field>
</subscriber>
```

Response 1

The request is successful, and the subscriber was retrieved ignoring the request content.

Request 2

Request URL

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

Request Content

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Weekday">
    <inputVolume>3220</inputVolume>
  </quota>
</usage>
```

Response 2

The request is successful, and *inputVolume* is updated to an empty value ignoring the value specified in the request content.

3.5 List of Messages

Table 6 lists the operations and messages for subscriber data. Each row of the table represents a command. Parameters required for each command are in colored columns. Any blank (uncolored) column represents unused parameter for corresponding command.

Table 6: Summary of Subscriber Commands

Operation Data	Command (Method)	URL	Main Object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	Instance Field Name	Instance Field Value	Field Name	Field Value	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)																							
	Delete Profile (DELETE)												Request Content											
Subscriber Field	Add Field Value (POST)					{Base URL}/msr	sub	{keyName} MSISDN, NAI, IMSI, AccountId	{keyValue}	field/ multipleFields					{fieldName}	{fieldValue}								
	Get Field (GET)																							
	Get Field Value (GET)																							
	Update Field (PUT)																							
	Delete Field (DELETE)																							
	Delete Field Value (DELETE)															{fieldValue}								
Subscriber Opaque Data	Set Opaque Data (PUT)									{Base URL}/msr	sub	{keyName} MSISDN, NAI, IMSI, AccountId	{keyValue}								Request Content			
	Get Opaque Data (GET)																							
	Delete Opaque Data (DELETE)																							
Subscriber Data Row	Set Row (PUT)	{Base URL}/msr	sub	{keyName} MSISDN, NAI, IMSI, AccountId	{keyValue}									data	{opaqueDataType}						Request Content			
	Get Row (GET)																				{instance IdentifierField}	{instance IdentifierValue}		
	Delete Row (DELETE)																				{instance IdentifierField}	{instance IdentifierValue}		
	Reset Quota (POST)																				{instance IdentifierField}	{instance IdentifierValue}		
Subscriber Data Row Field	Get Row Field (GET)					{Base URL}/msr	sub	{keyName} MSISDN, NAI, IMSI, AccountId	{keyValue}															
	Get Row Field Value (GET)																				{instance IdentifierField}	{instance IdentifierValue}	{fieldName}	{FieldValue}
	Update Row Field (PUT)																				{instance IdentifierField}	{instance IdentifierValue}		
																					{instance IdentifierField}	{instance IdentifierValue}		

Operation Data	Command (Method)	URL	Main Object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	Instance Field Name	Instance Field Value	Field Name	Field Value	Additional Input
	Delete Row Field (DELETE)												
									{instance IdentifierField}	{instance IdentifierValue}			
	Delete Row Field Value(DELETE)												
Subscriber Data Field	Set Data Field (PUT or POST)												
	Get Data Field (GET)												
	Delete Data Field (DELETE)												

Table 7 lists operations and messages for pool data. Similar to the previous table, each row of the table represents a command. Parameters required for each command are in colored columns. Any blank/uncolored column represents unused parameter for corresponding command.

Table 7: Summary of Pool Commands

Operation Data	Command (Method)	URL	Main Object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	Instance Field Name	Instance Field Value	Field Name	Field Value	Additional Input
Pool Profile	Create Pool (POST)	{Base URL}/msr	pool	Pool ID	{keyValue}								Request Content
	Get Pool (GET)												
	Update Pool (PUT)												Request Content
	Delete Pool (DELETE)												
Pool Profile Field	Add Field Value(POST)	{Base URL}/msr	pool	Pool ID	{keyValue}	field/ multipleFields					{fieldName}	{fieldValue}	
	Get Field (GET)												
	Get Field Value (GET)												
	Update Field (PUT)											{fieldValue}	
	Delete Field (DELETE)												
	Delete Field Value (DELETE)											{fieldValue}	
Pool Opaque Data	Set Opaque Data (PUT)					data	{opaqueDataType}						Request Content
	Get Opaque Data (GET)												
	Delete Opaque Data (DELETE)												

Operation Data	Command (Method)	URL	Main Object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	Instance Field Name	Instance Field Value	Field Name	Field Value	Additional Input
Pool Data Row	Set Row (PUT)								{instance IdentifierField}	{instance IdentifierValue}			Request Content
	Get Row (GET)								{instance IdentifierField}	{instance IdentifierValue}			
	Delete Row (DELETE)								{instance IdentifierField}	{instance IdentifierValue}			
	Reset Pool Quota (POST)								{instance IdentifierField}	{instance IdentifierValue}			
Pool Data Row Field	Get Row Field (GET)							{rowIdValue}	{instance IdentifierField}	{instance IdentifierValue}			
	Get Row Field Value (GET)								{instance IdentifierField}	{instance IdentifierValue}			
	Update Row Field (PUT)								{instance IdentifierField}	{instance IdentifierValue}			
	Delete Row Field (DELETE)								{instance IdentifierField}	{instance IdentifierValue}			
	Delete Row Field Value(DELETE)								{instance IdentifierField}	{instance IdentifierValue}			
Pool Data Field	Set Data Field (PUT or POST)												
	Get Data Field (GET)												
	Delete Data Field (DELETE)												

4. UDR DATA MODEL

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

The subscriber-related data includes:

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

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

The pool-related data includes:

- **Pool Profile**
Pre-provisioned information that describes a pool.
- **Pool Quota**
- **Information that represents the use of managed resources (quota, pass, top-up, roll-over) for the pool.**
- **Pool State**
Pool-specific properties
- **Pool Dynamic Quota**
Dynamically configured information related to managed resources (pass, top-up).
- **Pool Membership**
List of subscribers that are associated with a pool.

The data architecture supports multiple Network Applications. This flexibility is achieved 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:

- Subscription
 - A subscription exists for every individual subscriber
 - Maps a subscription to the user identities through which it can be accessed
 - Maps an individual subscription to the pool of which they are a member
- Pool Subscription
 - A pool subscription exists for every pool
 - Maps a pool subscription to the pool identity through which it can be accessed
 - Maps a pool subscription to the individual subscriptions of the subscribers that are members of the pool
- User Identities

Use to map a specific user identity to a subscription

 - IMSI, MSISDN, NAI and AccountId map to an individual subscription
 - Pool ID maps to a pool
- Pool Membership

Maps a pool to the list of the individual subscriber members

The Subscription Data Object (SDO):

- An SDO record contains a list of registers, holding a different type of entity data in each register
- An SDO record exists for:
 - Each individual subscriber

Defined entities stored in the registers are:

 - Profile
 - Quota
 - State
 - Dynamic Quota
 - Each pool

Defined entities stored in the registers are:

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

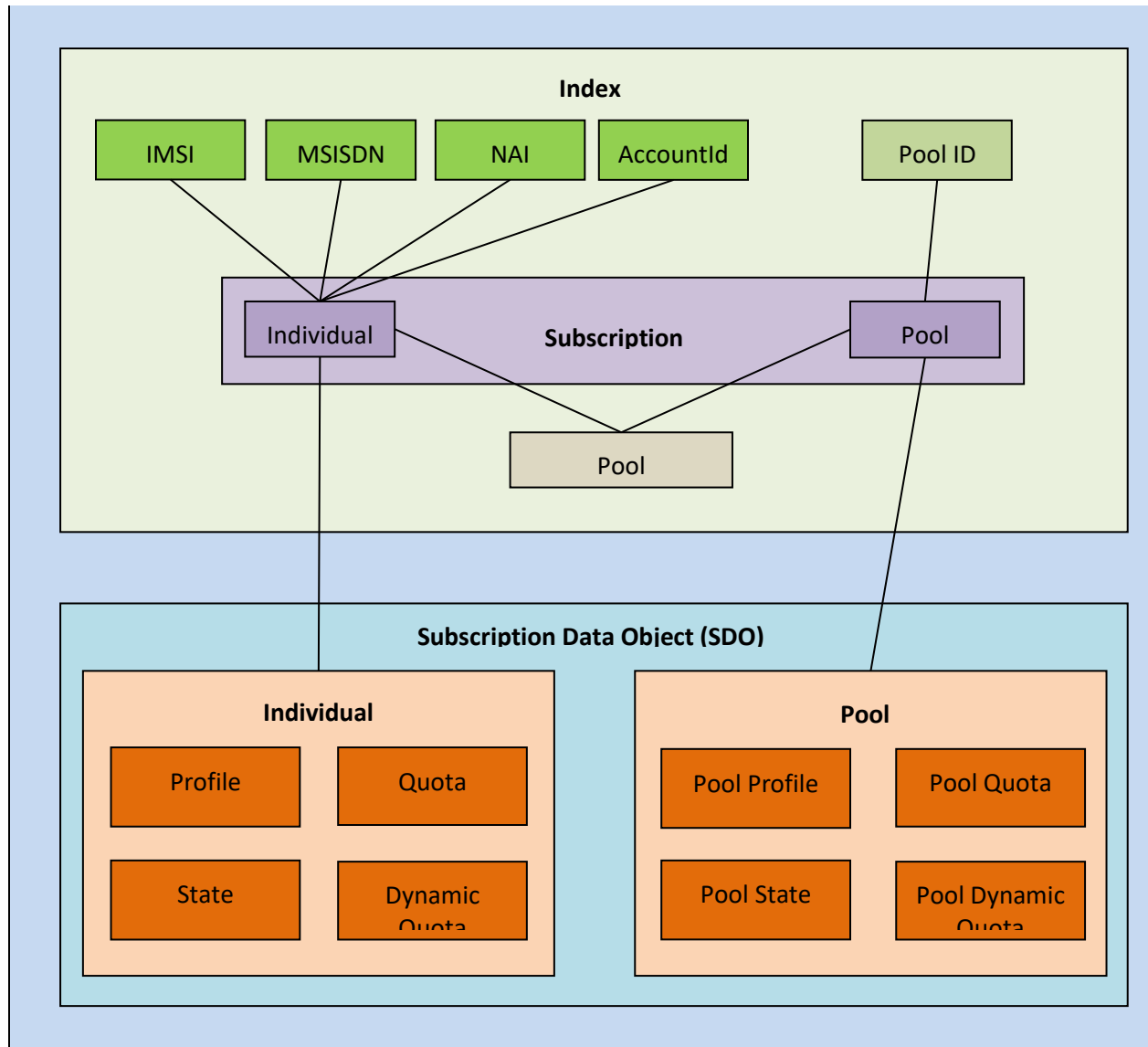
Provisioning applications can create, retrieve, modify, and delete subscriber/pool data. The indexing system allows access to the Subscriber SDO via IMSI, MSISDN, NAI or AccountId. The pool SDO can be accessed via Pool ID.

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

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

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

Figure 2: Data Model



4.1 Subscriber Data

4.1.1 Subscriber Profile

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

The Subscriber Profile supports the sequence of attributes in Table 8. Each record must have at least one of these key values: MSISDN, IMSI, NAI, or AccountId.

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

NOTE: UDR only supports an MSISDN with 8 to 15 numeric digits. A preceding + (plus) symbol is not supported, and is rejected.

Table 8: Subscriber Profile Entity Definition

Name (XML tag)	Type	Description
subscriber	—	Sequence (multiplicity is 1)
MSISDN	String	List of MSISDNs (8 to 15 numeric digits). A separate entry is included for each MSISDN associated with the profile for the subscriber.
IMSI	String	List of IMSIs (10 to 15 numeric digits). A separate entry is included for each IMSI associated with the profile for the subscriber.
NAI	String	<p>User or Domain length is between 0 to 63 characters</p> <p>NOTE: The limitation for 0 to 63 characters is because an NAI beyond 63 characters may not be possible to transfer through all devices. The user needs to ensure the combination of User and Domain does not exceed 63 characters (not including the @ character).</p> <p>List of NAIs (in format user@domain, user, or @domain). A separate entry is included for each NAI associated with the profile for the subscriber.</p> <p>The user or domain can be empty.</p> <p>Allowed characters for the user:</p> <p style="padding-left: 40px;">! % \$ A to Z, a to z, 0 through 9, . - _ / * = ^ ` # ' + ? { } ~</p> <p>Allowed characters for the domain:</p> <p style="padding-left: 40px;">A to Z, a to z, 0 through 9, . - _</p> <p>Example NAI Formats.</p> <p>bob, @privatecorp.example.net</p> <p>fred\$@example.com</p> <p>eng.example.net!nancy@example.net</p> <p>eng%nancy@example.net</p> <p>bob#+ ?@example.net</p>
AccountId	String	<p>Any string that can be used to identify the account for the subscriber (1 to 255 characters).</p> <p>Allowed values are any ASCII printable character, values x20 to x7e.</p>
BillingDay	String	<p>Allowed values are 0 to 31.</p> <p>The day of the month 1 to 31 that the associated quota for the subscriber is reset.</p> <p>0 indicates that the default value configured at the PCRF level is used. This is automatically set in any record where BillingDay is not specified.</p>

Name (XML tag)	Type	Description
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the profile for the subscriber.
Tier	String	Tier for the subscriber.
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.
Custom3	String	Fields used to store customer-specific data.
Custom4	String	Fields used to store customer-specific data.
Custom5	String	Fields used to store customer-specific data.
Custom6	String	Fields used to store customer-specific data.
Custom7	String	Fields used to store customer-specific data.
Custom8	String	Fields used to store customer-specific data.
Custom9	String	Fields used to store customer-specific data.
Custom10	String	Fields used to store customer-specific data.
Custom11	String	Fields used to store customer-specific data.
Custom12	String	Fields used to store customer-specific data.
Custom13	String	Fields used to store customer-specific data.
Custom14	String	Fields used to store customer-specific data.
Custom15	String	Fields used to store customer-specific data.
Custom16	String	Fields used to store customer-specific data.
Custom17	String	Fields used to store customer-specific data.
Custom18	String	Fields used to store customer-specific data.
Custom19	String	Fields used to store customer-specific data.
Custom20	String	Fields used to store customer-specific data.

4.1.2 Quota

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

The Quota entity is associated with a subscriber record and supports the sequence of attributes in Table 9.

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.

Quota entity is generally created, updated, and deleted by Oracle Communications Policy Management, therefore it is advised that operations on the Quota entity via SOAP or REST Provisioning Interfaces be avoided.

The default value in the table is used either:

- When a Quota instance is created, and a value is not supplied for the field. In this case, the field is created with the value indicated
- When a Quota instance is reset using the **Reset Quotas** command. If a field is defined as resettable, and the field exists, then it is set to the value indicated. If the field does not exist in the Quota, it is not created.

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

Table 9: Quota Entity Definition

Name (XML tag)	Type	Default Value	Description	Quota Versions
usage	—	—	Sequence (multiplicity is 1)	1/2/3
version	String	—	Version of the schema.	1/2/3
quota	—	—	Sequence (multiplicity is N)	1/2/3
name	String	—	Quota name (identifier).	1/2/3
cid	String	—	Internal identifier used to identify a quota in a Subscriber Profile.	1/2/3
time	String	Empty string ""	This element tracks the time-based resource consumption for a Quota.	1/2/3
totalVolume	String	0	This element tracks the bandwidth volume-based resource consumption for a Quota.	1/2/3
<i>inputVolume</i>	String	0	This element tracks the upstream bandwidth volume-based resource consumption for a Quota.	1/2/3
outputVolume	String	0	This element tracks the downstream bandwidth volume-based resource consumption for a Quota.	1/2/3
serviceSpecific	String	Empty string ""	This element tracks service-specific resource consumption for a Quota.	1/2/3
nextResetTime	String	Empty string ""	When set, it indicates the time after which the usage counters need to be reset. See section 5.3 for format details.	1/2/3

Name (XML tag)	Type	Default Value	Description	Quota Versions
Type	String	Empty string ""	Type of the resource in use.	2/3
grantedTotalVolume	String	0	Granted Total Volume <ul style="list-style-type: none"> For pool quota, it represents the granted total volume of all the subscribers in the pool. For individual quota, it represents the granted volume to all the PDN connections for that subscriber. 	2/3
grantedInputVolume	String	0	Granted Input Volume.	2/3
grantedOutputVolume	String	0	Granted Output Volume.	2/3
grantedTime	String	Empty string ""	Granted Total Time.	2/3
grantedServiceSpecific	String	Empty string ""	Granted Service Specific Units.	2/3
QuotaState	String	Empty string ""	State of the resource in use.	3
RefInstanceid	String	Empty string ""	Instance-id of the associated provisioned pass, top-up or roll-over.	3

4.1.3 State

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

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

The default fields configured are not:

- Resettable
- Defaultable

The State entity supports the sequence of attributes in Table 10.

Table 10: State Entity Definition

Name (XML tag)	Type	Description
state	—	Sequence (multiplicity is 1)
version	String	Version of the schema.
property	—	Sequence (multiplicity is N)
name	String	The property name.
value	String	Value associated with the property.

4.1.4 Dynamic Quota

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

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

The default fields configured are not:

- Resettable
- Defaultable

The DynamicQuota entity supports the sequence of attributes listed in Table 11.

Table 11: Dynamic Quota Entity Definition

Name (XML tag)	Type	Description
definition	—	Sequence (multiplicity is 1)
version	String	Version of the schema.
DynamicQuota	—	Sequence (multiplicity is N)
Type	String	Identifies the dynamic quota type.
name	String	The class identifier for a pass or top-up. This name is used to match top-ups to quota definitions on the PCRF. This name is used in policy conditions and actions on the PCRF.
InstanceId	String	A unique identifier to identify this instance of a dynamic quota object.
Priority	String	An integer represented as a string. This number allows service providers to specify when one pass or top-up is used before another pass or top-up.
InitialTime	String	An integer represented as a string. The number of seconds initially granted for the pass/top-up.
InitialTotalVolume	String	An integer represented as a string. The number of bytes of total volume initially granted for the pass/top-up.
InitialInputVolume	String	An integer represented as a string. The number of bytes of input volume initially granted for the pass/top-up.
InitialOutputVolume	String	An integer represented as a string. The number of bytes of output volume initially granted for the pass/top-up.
InitialServiceSpecific	String	An integer represented as a string. The number of service specific units initially granted for the pass/top-up.
activationdatetime	String	The date/time after which the pass or top-up may be active. See section 5.3 for format details.
expirationdatetime	String	The date/time after which the pass or top-up is considered to be exhausted

Name (XML tag)	Type	Description
		See section 5.3 for format details.
purchasedatetime	String	The date/time when a pass was purchased See section 5.3 for format details.
Duration	String	The number of seconds after first use in which the pass must be used or expired. If both Duration and expirationdatetime are present, the closest expiration time is used.
InterimReportingInterval	String	The number of seconds after which the GGSN, DPI, or Gateway revalidates quota grants with the PCRF.

4.2 Pool Data

4.2.1 Pool Profile

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

Each Pool Profile must have a unique key value called Pool ID.

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

The Pool profile record consists of the sequence of attributes in Table 12.

Table 12: Pool Profile Entity Definition

Name (XML tag)	Type	Description
pool	—	Sequence (multiplicity is 1)
Pool ID	String	Pool identifier (1 to 22 numeric digits, minimum value 1).
BillingDay	UInt8	The day of the month (1 to 31) when the associated quota for the pool is reset. 0 indicates that the default value configured at the PCRF level is used.
BillingType	String	The billing frequency, monthly, weekly, daily.
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the profile for the pool.
Tier	String	Tier for the pool.
Type	String	Field used to identify an Enterprise Pool. Allowed value is enterprise and is not case-sensitive
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.

Name (XML tag)	Type	Description
Custom3	String	Fields used to store customer-specific data.
Custom4	String	Fields used to store customer-specific data.
Custom5	String	Fields used to store customer-specific data.
Custom6	String	Fields used to store customer-specific data.
Custom7	String	Fields used to store customer-specific data.
Custom8	String	Fields used to store customer-specific data.
Custom9	String	Fields used to store customer-specific data.
Custom10	String	Fields used to store customer-specific data.
Custom11	String	Fields used to store customer-specific data.
Custom12	String	Fields used to store customer-specific data.
Custom13	String	Fields used to store customer-specific data.
Custom14	String	Fields used to store customer-specific data.
Custom15	String	Fields used to store customer-specific data.
Custom16	String	Fields used to store customer-specific data.
Custom17	String	Fields used to store customer-specific data.
Custom18	String	Fields used to store customer-specific data.
Custom19	String	Fields used to store customer-specific data.
Custom20	String	Fields used to store customer-specific data.

4.2.2 Pool Quota

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

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

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

NOTE: Pool Quota entity is generally created, updated and deleted by Oracle Communications Policy Management, therefore it is advised that operations on Pool Quota entity via SOAP or REST Provisioning Interfaces be avoided.

4.2.3 Pool State

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

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

The default fields configured are not:

- Resetable
- Defaultable

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

4.2.4 Pool Dynamic Quota

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

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

The default fields configured are not:

- Resetable
- Defaultable

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

4.3 Date/Timestamp Format

The Date/Timestamp format used by many fields is:

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

This corresponds to either:

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

Where:

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

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

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

5. SUBSCRIBER PROVISIONING

5.1 Subscriber Profile Commands

Table 13: Summary of Subscriber Profile Commands

Command	Description	Keys	Command Syntax
Create Profile	Create a subscriber or Subscriber Profile	MSISDN, NAI, IMSI, AccountId	POST <i>{baseURI}</i> /msr/sub
Get Profile	Get Subscriber Profile data		GET <i>{baseURI}</i> /msr/sub/ <i>keyName</i> / <i>keyValue</i>
Update Profile	Replace an existing Subscriber Profile		PUT <i>{baseURI}</i> /msr/sub/ <i>keyName</i> / <i>keyValue</i>
Delete Profile	Delete all Subscriber Profile data and all opaque data associated with the subscriber		DELETE <i>{baseURI}</i> /msr/sub/ <i>keyName</i> / <i>keyValue</i>

5.1.1 Create Subscriber

Description

This operation creates a Subscriber Profile using the field-value pairs that are specified in the request content.

Unlike other subscriber commands, *keyName* and *keyValue* are not specified in the 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 SEC.

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

Prerequisites

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

Request URL

```
POST {baseURI}/msr/sub
```

Request 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>
```

- *keyNameX*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *fieldNameX*: A user defined field in the Subscriber Profile.
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*.

NOTES:

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

Response Content

None.

Table 14: Create Subscriber 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 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
400	MSR4103	A key is detected in the system for an AE subscriber
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 exists with the *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 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 key values were not 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>
```

Request 6

A subscriber is created, with MSISDN and IMSI keys. The BillingDay and Entitlement fields are set. An AE subscriber exists with the IMSI and enableAEKeyAlreadyExistsErrCode option is set to TRUE.

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 6

The request fails. The error code indicates the key exists for an AE subscriber.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4103">errorText</error>
```

Request 7

A subscriber is created, with MSISDN and IMSI keys. The BillingDay and Entitlement fields are set. An AE subscriber exists with the IMSI and enableAEKeyAlreadyExistsErrCode option is set to FALSE.

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 7

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

HTTP Status Code

400

Response Content

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

5.1.2 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
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.

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>

```

- *keyNameX*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *fieldNameX*: A user defined field in the Subscriber Profile.
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*.

Key/field order in the response is not important

Table 15: Get Profile 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 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 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>

```

5.1.3 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.
- Multi-value fields can be specified by a single *fieldNameX* value with a delimited list of values, or multiple *fieldNameX* fields each containing a single value.

Prerequisites

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

Request URL

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.

Request Content

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

- *keyNameX*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValueX*: Corresponding key field value assigned to *keyNameX*.
- *fieldNameX*: A user defined field in the Subscriber Profile.
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*.

NOTES:

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

Response Content

None.

Table 16: Update Profile 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 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

HTTP Status Code	Error Code	Description
404	MSR4002	Subscriber field is not defined

Examples

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.

5.1.4 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 fails.

Request URL

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.

Request Content

None.

Response Content

None.

Table 17: Delete Profile 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 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 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>
```

5.2 Subscriber Profile Field Commands**Table 18: Summary of Subscriber Profile Field Commands**

Command	Description	Keys	Command Syntax
Add Field Value	Adds a 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/keyName/keyValue/ field/fieldName/fieldValue
Get Field	Retrieve the values for the specified field		GET {baseURI}/msr/sub/keyName/keyValue/ field/fieldname
Get Field Value	Retrieve the single value for the specified field (if set as specified)		GET {baseURI}/msr/sub/keyName/keyValue/ field/fieldName/fieldValue
Update Field Value	Updates field to the specified value		PUT {baseURI}/msr/sub/ keyName/keyValue/field/fieldName/ fieldValue
Update Multiple Fields	Update multiple fields to the specified values		PUT {baseURI}/msr/sub/keyName/keyValue/ multipleFields/fieldName1/ fieldValue1/fieldName2/fieldValue2/...
Delete Field	Delete all the values for the specified field		DELETE {baseURI}/msr/sub/keyName/keyValue/ field/fieldname
Delete Field Value	Delete a value for the specified field		DELETE {baseURI}/msr/sub/keyName/keyValue/ field/fieldName/fieldValue

5.2.1 Add Field Value**Description**

This operation adds one or more values 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 SEC. Any pre-existing values for the field are not affected.

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

If a value being added exists, the request fails.

NOTES:

- If the field to which the value is being added does not exist, it is created.
- The *fieldValue* is case-sensitive. An attempt to add the value a to the current field value of a;b;c fails., but an attempt to add the value A is successful and result in the field value being a;b;c;A

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 be present in the field.

Request URL

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *fieldName*: A user defined field in the Subscriber Profile.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, a;b;c.
- The semicolon between the field values may need to be encoded as %3B for certain clients.

Request Content

None.

Response Content

None.

Table 19: Add Field Value 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

HTTP Status Code	Error Code	Description
400	MSR4064	Occurrence constraint violation
400	MSR4066	Field value 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 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 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>
```

Request 4

A request is made to update by adding two additional MSISDN values. The subscriber only has the MSISDN 15141234567.

Request URL

```
POST {baseURI}/msr/sub/MSISDN/5141234567/field/MSISDN/14161112222; 14505556666
```

Request Content

None.

Response 4

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

HTTP Status Code

200

Response Content

None.

5.2.2 Get Field

Description

This operation retrieves the values of the specified fields 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.

Request URL

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *fieldName*: A user defined field in 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>
```

- *fieldName*: The requested user defined field in the Subscriber Profile.
- *fieldValueX*: Corresponding field value assigned to *fieldname*.

NOTE: For multi-value fields, more than one <field> element may be returned. One element per value.

Table 20: 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

HTTP Status Code	Error Code	Description
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 fails and an error is returned.

HTTP Status Code

404

Response Content

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

5.2.3 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.

NOTES:

- Depending on the field, there may be multiple field-value pairs returned by this operation.
- The *fieldValue* is case-sensitive. An attempt to get value a from the field value of a;b;c is successful, but an attempt to get the value A fails.

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 values supplied in the *fieldValue*.

Request URL

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.

- *keyValue*: Corresponding key field value assigned to *keyName*.
- *fieldName*: A user defined field in the Subscriber Profile.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, a;b;c.
- The semicolon between the field values may need to be encoded as %3B for certain clients.

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>
```

- *fieldNameX*: The requested user defined field in the Subscriber Profile.
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*.

NOTE: For multi-value fields, more than one <field> element may be returned. One element per value.

Table 21: Get Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested field exists for subscriber with value
400	MSR4053	Subscriber and field exist, but values 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>
```

5.2.4 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 set specified is inserted. Adding values to a set is accomplished using Add Field Value.

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
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *fieldName*: A user defined field in the Subscriber Profile.
NOTE: A field name cannot be for a key value—that is, IMSI, MSISDN, NAI, or AccountId.
- *fieldValue*: Corresponding field value assigned to *fieldname*.
NOTE: For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, 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.

Table 22: Update Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	—	Fields 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.

Request 3

A request is made to update the value of the subscribers *MSISDN* to *15145551234*.

Request URL

```
PUT {baseURI}/msr/sub/MSISDN/33123654862/field/MSISDN/15145551234
```

Request Content

None.

Response 3

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

HTTP Status Code

201

Response Content

None.

Request 4

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

Request URL

```
PUT {baseURI}/msr/sub/IMSI/302370123456789/field/IMSI/302370111111111
```

Request Content

None.

Response 4

The request is successful, and the *IMSI* field was updated. The subscriber has a single IMSI of 302370111111111.

HTTP Status Code

201

Response Content

None.

Request 5

A request is made to update the value of the subscribers *NAI* to two values of *mum@foo.com* and *cust514@op.com*.

Request URL

```
PUT {baseURI}/msr/sub/MSISDN/15141234567/field/NAI/mum@foo.com;cust514@op.com
```

Request Content

None.

Response 5

The request is successful, and the *NAI* field was updated. The subscriber has 2 NAIs.

HTTP Status Code

201

Response Content

None.

5.2.5 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 set specified is inserted. Adding values to a 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 field request. Two or three fields can be updated at once. Updating only a single field results 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.

Prerequisites

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

The *fieldNameX* fields must all be valid in the Subscriber Profile.

Request URL

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *fieldNameX*: A user defined field in the Subscriber Profile.
NOTE: A field name cannot be for a key value—that is, IMSI, MSISDN, NAI, or AccountId.
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, a;b;c.
- The semicolon between the field values may need to be encoded as %3B for certain clients.

Request Content

None.

Response Content

None.

Table 23: Update Multiple Fields Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	—	Fields 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

Examples**Request 1**

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 1

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

HTTP Status Code

201

Response Content

None.

Request 2

A request is made to update the *MSISDN* field to *15145551234*, the *Tier* field to *Silver*, and the *NAI* field to *mum@foo.com*.

Request URL

```
PUT {baseURI}/msr/sub/MSISDN/33123654862/multipleFields/
MSISDN/15145551234/Tier/Silver/NAI/mum@foo.com
```

Request Content

None.

Response 2

The request is successful, and the *MSISDN*, *Tier*, and *NAI* fields were all updated.

HTTP Status Code

201

Response Content

None.

5.2.6 Delete Field**Description**

This operation deletes the specified field for the subscriber identified by *keyName* and *keyValue* in the request. If the field is a multi-value field then all values are deleted. Deletion of a field results in removal of the field from the Subscriber Profile. Meaning that the field is not present, not just the value is empty.

NOTES:

- The field being deleted does not need to have a value. It can be empty (deleted), and the request succeeds.
- If the field being deleted is mandatory, and is defined as having a default value, then the field is not removed, but has the default value assigned.
- If a key (IMSI, MSISDN, NAI, or AccountId) field is deleted for a subscriber, then afterwards, the subscriber must still have at least one key type/value remaining or the request fails.

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
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *fieldName*: A user defined field in the Subscriber Profile.

Request Content

None.

Response Content

None.

Table 24: Delete Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Field was successfully deleted
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4069	At least one key is required
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Examples**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/field/Tier
```

Request Content

None.

Response 1

The request is successful, and the field was deleted.

HTTP Status Code

204

Response Content

None.

Request 2

A request is made to delete the IMSI key field. The subscriber has MSISDN and IMSI key fields.

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/15141234567/field/IMSI
```

Request Content

None.

Response 2

The request is successful, and the IMSI key field was deleted.

HTTP Status Code

204

Response Content

None.

Request 3

A request is made to delete the MSISDN key field. The subscriber only has a single MSISDN key field.

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/15145551234/field/MSISDN
```

Request Content

None.

Response 3

The request fails, because the single MSISDN key field is the only existing key.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>  
<error code="MSR4069">errorText</error>
```

Request 4

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

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/15141234567/field/MSISDN
```

Request Content

None.

Response 4

The request is successful, and the MSISDN field is deleted. The subscriber does not have any MSISDN values, and just has an IMSI

HTTP Status Code

204

Response Content

None.

5.2.7 Delete Field Value**Description**

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

This operation can only be used for the fields defined as multi-value field in the SEC.

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

If all values are removed, the field is removed from the Subscriber Profile (an XML element is not present).

NOTE: The *fieldValue* is case-sensitive. An attempt to remove the value a from a field value of a;b;c is successful, but an attempt to remove the value A fails.

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
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *fieldName*: A user defined field in the Subscriber Profile.
NOTE: A field name cannot be for a key value—That is, IMSI, MSISDN, NAI, or AccountId
- *fieldValue*: Corresponding field value assigned to *fieldname*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, a;b;c.
- The semicolon between the field values may need to be encoded as %3B for certain clients.

Request Content

None.

Response Content

None.

Table 25: Delete Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Requested fields were successfully deleted
400	MSR4005	Field does not support multiple values
400	MSR4056	Field is not updatable
400	MSR4069	At least one key is required
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>
```

Request 3

A request is made to delete the MSISDN fields with values of *14161112222* and *15141234567*. The subscriber has 3 MSISDN values, *15141234567*, *14161112222*, and *15145556666*.

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/15141234567/field/MSISDN/14161112222;15141234567
```

Request Content

None.

Response 3

The request is successful, and the MSISDN values *14161112222* and *15141234567* are deleted. The subscriber has a single MSISDN of *15145556666*.

HTTP Status Code

204

Response Content

None.

5.3 Subscriber Opaque Data Commands

The commands listed in Table 26 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 XML blob level. The content of the entity is set, returned, or deleted.

Table 26: Summary of Subscriber Opaque Data Commands

Command	Description	Keys	Command Syntax
Set Opaque Data	Create or 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

Command	Description	Keys	Command Syntax
Delete Opaque Data	Delete opaque data of the specified type		DELETE {baseURI}/msr/sub/keyName/keyValue/data/opaqueDataType

5.3.1 Set Opaque Data

Description

This operation updates (or creates if it not exists) 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
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either quota, state, or 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>
```

- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either quota, state, or dynamicquota.
- *cdataFieldValue*: Contents of the XML data blob.

NOTE: The *opaqueDataType* in the request content is ignored, and is not validated. The *opaqueDataType* in the URL is used to identify the opaque data type.

Response Content

None.

Table 27: Set Opaque Data 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>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </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 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.

5.3.2 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 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}
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either quota, state, or 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>
```

- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either quota, state, or dynamicquota.
- *cdataFieldValue*: Contents of the XML data blob.

Table 28: Get Opaque Data Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested data exists for subscriber

HTTP Status Code	Error Code	Description
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined
404	MSR4053	Data type is not set for this subscriber

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>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</subscriber>
```

Request 2

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

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>
```

5.3.3 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 subscriber, this is not considered an error and a successful result is 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
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either quota, state, or dynamicquota.

Request Content

None.

Response Content

None.

Table 29: Delete Opaque Data 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 State opaque data was not deleted.

HTTP Status Code

204

Response Content

None.

5.4 Subscriber Data Row Commands

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

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

NOTE: Subscriber data row commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an HTTP Status Code 400, with an MSR4070 error being returned.

Table 30: Summary of Subscriber Data Row Commands

Command	Description	Keys	Command Syntax
Set Row	Create or update data row in data of the specified type.	(MSISDN, IMSI, NAI or AccountId) and Row Identifier or (MSISDN, IMSI, NAI or AccountId) and Row Identifier and Instance Identifier	<code>PUT {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue</code>
	Create or update data row in data of the specified type and instance identifier		<code>PUT {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue</code>
Get Row	Retrieve data row from data of the specified type.		<code>GET {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue</code>
	Retrieve data row from data of the specified type and instance identifier		<code>GET {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue</code>
Delete Row	Delete data row in data of the specified type		<code>DELETE {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue</code>
	Delete data row in data of the specified type and instance identifier		<code>DELETE {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue</code>

5.4.1 Set Row

Description

This operation creates a row 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 in the row.

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

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

NOTES:

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

Prerequisites

A subscriber with the 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

Without Instance Identifier

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

With Instance Identifier

```
PUT {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *quota* for the Quota transparent data.
 - Value is *dynamicquota* for the DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the Quota transparent data.
 - Value is *Instanceid* or *Type* for the DynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.

Request Content

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

- *rowValue*: Contents of the XML data blob with the row data.

NOTE: The *rowValue* is the same format as a Quota entity only containing the row being added in a single row.

NOTE: The data in the *rowValue* contains the same *rowIdValue* as specified in the URL. The *rowIdValue* in the URL is ignored, and is not validated. The *rowIdValue* in the request content is solely used to identify the row.

Response Content

None.

Table 31: Set Row 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	MSR4056	Field is not updatable
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

HTTP Status Code	Error Code	Description
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>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
```

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>3</version>
  <quota name="Q1">
    <cid>9223372036854775807</cid>
```

```

    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>

```

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>3</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>3</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 is created.

HTTP Status Code

201

Response Content

None.

Request 5

A request is made to update a data row in the *dynamicquota* transparent data for a subscriber with data row identifier field value *AggregateLimit* and *InstanceId 15678*. The *AggregateLimit* data row exists in the DynamicQuota data, but there are two rows called *AggregateLimit* one with *InstanceId* of *15570*, the other with an *InstanceId* of *15678*. The request is not required in the response.

Request URL

```
PUT {baseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/AggregateLimit/row/InstanceId/15678
```

Request Content

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="AggregateLimit">
    <Type>pass</Type>
    <InstanceId>15678</InstanceId>
    <Priority>4</Priority>
    <InitialTime>135</InitialTime>
    <InitialTotalVolume>2000</InitialTotalVolume>
    <InitialInputVolume>1500</InitialInputVolume>
    <InitialOutputVolume>500</InitialOutputVolume>
    <InitialServiceSpecific>4</InitialServiceSpecific>
```

```

    <activationdatetime>2015-05-22T00:00:00-05:00</activationdatetime>
    <expirationdatetime>2015-05-29T00:00:00-05:00</expirationdatetime>
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
</definition>

```

Response 5

The request is successful, and the data row *AggregateLimit* with *InstanceId* of 15678 was updated.

HTTP Status Code

201

Response Content

None.

5.4.2 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 called *DayPass* exists, then an attempt to get a row called *DayPass* is successful, but an attempt to get a row called *DAYPASS* fails

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 identifier in the transparent data must exist for the subscriber.

Request URL**Without Instance Identifier**

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

With Instance Identifier

```
GET {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *quota* for the Quota transparent data.
 - Value is *dynamicquota* for the DynamicQuota transparent data.

- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the Quota transparent data
 - Value is *InstanceId* or *Type* for the DynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.

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>
```

- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *quota* for the Quota transparent data.
 - Value is *dynamicquota* for the DynamicQuota transparent data.
- *cdatarowValue*: Contents of the XML data blob with the row data.

Table 32: Get Row 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>3</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, as 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>3</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>
```

Request 4

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

Request URL

```
GET {baseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1/InstanceId/11223344
```

Request Content

None.

Response 4

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

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="dynamicquota">
    <![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="DQ1">
    <Type>topup</Type>
    <InstanceId>11223344</InstanceId>
    <Priority>4</Priority>
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
  <DynamicQuota name="DQ1">
    <Type>pass</Type>
    <InstanceId>11223344</InstanceId>
    <Priority>5</Priority>
    <InterimReportingInterval>200</InterimReportingInterval>
    <Duration>20</Duration>
  </DynamicQuota>
</definition>]]>
  </data>
</subscriber>
```

5.4.3 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.

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to delete a row called DayPass is successful, but an attempt to delete a row called DAYPASS fails
- The deletion of a non-existent data row is not considered an error.

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

Without Instance Identifier

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

With Instance Identifier

```
DELETE {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *quota* for the Quota transparent data.
 - Value is *dynamicquota* for the DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the Quota transparent data
 - Value is *InstanceId* or *Type* for the DynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.

Request Content

None.

Response Content

None.

Table 33: Delete Row 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>
```

Request 4

A request is made to delete the *DQ1* data row in the *dynamicquota* transparent data. The *DQ1* data row exists in the DynamicQuota data.

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1
```

Request Content

None.

Response 4

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

HTTP Status Code

204

Response Content

None.

Request 5

A request is made to delete the *DQ1* data row in the *dynamicquota* transparent data with an InstanceId *12345*. The *DQ1* data row with InstanceId *12345* exists in the DynamicQuota data.

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1/row/InstanceId/12345
```

Request Content

None.

Response 5

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

HTTP Status Code

204

Response Content

None.

5.5 Subscriber Data Row Field Commands

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

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

NOTE: Subscriber data row field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in HTTP Status Code 400, with an MSR4070 error being returned.

Table 34: Summary of Subscriber Data Row Field Commands

Command	Description	Keys	Command Syntax
Get Row Field	Retrieve values for the specified field	(MSISDN, IMSI, NAI or AccountId) and Row Identifier and Field name or (MSISDN, IMSI, NAI or AccountId) and Row Identifier, Instance Identifier and Field name	GET {baseURI}/msr/sub/keyName/ keyValue/data/transparentDataType/ rowIdValue/fieldName
	Retrieve values for the specified field and instance identifier		GET {baseURI}/msr/sub/keyName/ keyValue/data/transparentDataType/ rowIdValue/row/instanceFieldName/ instanceFieldValue/fieldName
Get Row Field Value	Retrieve a single value for the specified field		GET {baseURI}/msr/sub/keyName/ keyValue/data/transparentDataType/ rowIdValue/fieldName/fieldValue
	Retrieve a single value for the specified field and instance identifier		GET {baseURI}/msr/sub/keyName/ keyValue/data/transparentDataType/ rowIdValue/row/instanceFieldName/ instanceFieldValue/fieldName/fieldValue
Update Row Field	Update field to the specified value		PUT {baseURI}/msr/sub/keyName/ keyValue/data/transparentDataType/ rowIdValue/fieldName/fieldValue
	Update field to the specified value and instance identifier		PUT {baseURI}/msr/sub/keyName/ keyValue/data/transparentDataType/ rowIdValue/row/instanceFieldName/ instanceFieldValue/fieldName/fieldValue
Delete Row Field	Delete all values for the specified field	DELETE {baseURI}/msr/sub/keyName/ keyValue/data/transparentDataType/ rowIdValue/fieldName	

Command	Description	Keys	Command Syntax
	Delete all values for the specified field and instance identifier		DELETE {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName

5.5.1 Get Row Field

Description

This operation retrieves a field in 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 are 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 called DayPass exists, then an attempt to get a field in a row called DayPass is successful, but an attempt to get a field in a row called DAYPASS fails

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 identifier in the transparent data must exist for the subscriber.

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *quota* for the Quota transparent data.
 - Value is *dynamicquota* for the DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the Quota transparent data

- Value is InstanceId or Type for the DynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.
- *fieldName*: A user defined field in the transparent data row.

Request Content

None.

Response Content

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

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

- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *quota* for the Quota transparent data.
 - Value is *dynamicquota* for the DynamicQuota transparent data.
- *cdataRowFieldValue*: Contents of the XML data blob, with the field from the row data.

Table 35: Get Row Field 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>3</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>3</version>
  <quota name="Weekday">
```

```

    <inputVolume>980</outputVolume>
  </quota>
  <quota name="Weekday">
    <inputVolume>2140</outputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>

```

Request 3

A request is made to get the *InitialInputVolume* field in the *DQ1* data row of the *dynamicquota* transparent data having *InstanceId* value of *11223344*.

Request URL

```

GET {BaseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1/row/InstanceId/11223344/
InitialInputVolume

```

Request Content

None.

Response 3

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="dynamicquota">
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="DQ1">
    <InitialInputVolume>15678</InitialInputVolume>
  </DynamicQuota>
</definition>
]]>
  </data>
</subscriber>

```

5.5.2 Get Row Field Value**Description**

This operation retrieves a field with a value in 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 are 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*.

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to get a field value in a row called DayPass is successful, but an attempt to get a field value in a row called DAYPASS fails
- The *fieldValue* is case-sensitive. An attempt to get the value Data from a field value of Data is successful, but an attempt to get the value DATA fails.

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 identifier in the transparent data must 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**Without Instance Identifier**

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

With Instance Identifier

```
GET {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName/fieldValue
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *quota* for the Quota transparent data.
 - Value is *dynamicquota* for the DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the Quota transparent data
 - Value is *Instanceid* or *Type* for the DynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.
- *fieldName*: A user defined field in the transparent data row.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

Request Content

None.

Response Content

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

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

```

<data name="transparentDataType">
<![CDATA[
cdataRowFieldValue
]]>
</data>
</subscriber>

```

- **transparentDataType:** A user defined type/name for the transparent data.
 - Value is `quota` for the Quota transparent data.
 - Value is `dynamicquota` for the DynamicQuota transparent data.
- **cdataRowFieldValue:** Contents of the XML data blob, with the field from the row data.

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.

Table 36: Get Row Field 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 with the value of *980* 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 with the specified 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>3</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 3220 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>3</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>3</version>
  <quota name="Weekday">
    <inputVolume>980</inputVolume>
  </quota>
</usage>
]]>
```



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

Request 5

A request is made to get the *InitialInputVolume* field with the value of 980 in the *DQ1* data row of the *dynamicquota* transparent data for a subscriber with an *InstanceId* of 345324534. The *InitialInputVolume* field exists, and is set to the value 980.

Request URL

```
GET {BaseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1/row/InstanceId/345324534/
InitialInputVolume/980
```

Request Content

None.

Response 5

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

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="dynamicquota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="DQ1">
    <InitialInputVolume>980</InitialInputVolume>
  </DynamicQuota>
</definition>
]]>
</data>
</subscriber>
```

5.5.3 Update Row Field

Description

This operation updates fields in 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 exist, it is created.

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

NOTE: The *rowIdValue* is case-sensitive. If a row called *DayPass* exists, then an attempt to update a field in a row called *DayPass* is successful, but an attempt to update a field in a row called *DAYPASS* fails

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 identifier in the transparent data must 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

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *quota* for the Quota transparent data.
 - Value is *dynamicquota* for the DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the Quota transparent data.
 - Value is *Instanceid* or *Type* for the DynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.
- *fieldName*: A user defined field in the transparent data row.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

Request Content

None.

Response Content

None.

Table 37 Update Row Field 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

HTTP Status Code	Error Code	Description
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 *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, 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 the *InitialTotalVolume* field in the *DQ1* data row of the *dynamicquota* transparent data for a subscriber with a *Type* value of *pass*.

Request URL

```
PUT {BaseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1/row/Type/pass/
InitialTotalVolume/0
```

Request Content

None.

Response 4

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

HTTP Status Code

201

Response Content

None.

5.5.4 Delete Row Field**Description**

This operation deletes a field in 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 fails.

NOTES:

- If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and row/field data is not deleted.
- If the field with opaque data of the *opaqueDataType* does not exist, this is not considered an error and a successful result is returned.
- If the field being deleted is mandatory, and is defined as having a default value, then the field is not removed, but has the default value assigned.
- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to delete a field in a row called DayPass is successful, but an attempt to delete a field in a row called DAYPASS fails

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 identifier in the transparent data must 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**Without Instance Identifier**

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

With Instance Identifier

```
DELETE
{baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/
instanceFieldValue/fieldName
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.

- Value is `quota` for the Quota transparent data.
- Value is `dynamicquota` for the DynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is `cid` or `Type` for the Quota transparent data.
 - Value is `Instanceid` or `Type` for the DynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.
- *fieldName*: A user defined field in the transparent data row.

Request Content

None.

Response Content

None.

Table 38: Delete Row Field 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
400	MSR4064	Occurrence constraint violation
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 of the *quota* transparent data for a subscriber.

Request URL

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

Request Content

None.

Response 1

The request is successful, and the field in 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 of 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, 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 3

A request is made to delete the *InitialTotalVolume* field in the *DQ1* data row of the *dynamicquota* transparent data for a subscriber.

Request URL

```
DELETE {BaseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1/InitialTotalVolume
```

Request Content

None.

Response 3

The request is successful, and the field in the data row in the *dynamicquota* transparent data was deleted.

HTTP Status Code

204

Response Content

None.

Request 4

A request is made to delete the *InitialTotalVolume* field in the DQ1 data row of the *dynamicquota* transparent data for a subscriber with a *Type* value of *pass*.

Request URL

```
DELETE {BaseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1/row/Type/pass/InitialTotalVolume
```

Request Content

None.

Response 4

The request is successful, and the field in the data row in the *dynamicquota* transparent data was deleted.

HTTP Status Code

204

Response Content

None.

5.6 Subscriber Data Field Commands

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

```
<property>
  <name>X</name>
  <value>Y</value>
</property>
```

The data field commands allow operations (create, retrieve, update, or delete) at the field level. The required field is identified in the request by the `FieldName`.

NOTE: Subscriber data field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an HTTP Status Code 400, with an MSR4070 error being returned.

Table 39: Summary of Subscriber Data Field Commands

Command	Description	Keys	Command Syntax
Set Data Field	Create or update data field in transparent data of the	(MSISDN, IMSI, NAI or	POST {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/fieldName/fieldValue

Command	Description	Keys	Command Syntax
	specified type.	AccountId) and Field Name	<code>PUT {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/fieldName/fieldValue</code>
Get Data Field	Retrieve data field from transparent data of the specified type.		<code>GET {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/fieldName</code>
Delete Data Field	Delete data field in transparent data of the specified type		<code>DELETE {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/fieldName</code>

5.6.1 Set Data Field

Description

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

The field name is specified in *fieldName*, and the field value is specified in *fieldValue*.

If more than one existing fields matches the requested *fieldName*, then the update request fails.

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

NOTES:

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

Prerequisites

A subscriber with the 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

Format 1

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

Format 2

```
POST {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/fieldname/fieldValue
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
Value is *state* for the State transparent data.
- *fieldName*: A user defined field in the transparent data.

- For the State entity, this corresponds to a property in the entity
- The *fieldName* is stored exactly as it is sent in the request. The case of *fieldName* changes if an update is performed using a different case.
- *fieldValue*: Corresponding field value assigned to *fieldName*.

Request Content

None.

Response Content

None.

Table 40: Set Data Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	—	Data field was successfully created/updated
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4067	Multiple matching fields 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 property in the *state* transparent data for a subscriber. The property name is *mcc* and the property value is *315*. The subscriber has an existing State transparent data, but not a State property called *mcc*.

Request URL

```
POST {baseURI}/msr/sub/MSISDN/33123654862/data/state/mcc/315
```

Request Content

None.

Response 1

The request is successful, and the property *mcc* with value *315* was created.

HTTP Status Code

201

Response Content

None.

Request 2

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

Request URL

```
PUT {baseURI}/msr/sub/MSISDN/33123654862/data/state/mcc/315
```

Request Content

None.

Response 2

The request is successful, and the property *mcc* as well as the State entity is created.

HTTP Status Code

201

Response Content

None.

Request 3

A request is made to update a property in the *state* transparent data for a subscriber. The property name is *mcc*. The subscriber has an existing State property called *mcc*.

Request URL

```
POST {baseURI}/msr/sub/MSISDN/33123654862/data/state/mcc/400
```

Request Content

None.

Response 3

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

HTTP Status Code

201

Response Content

None.

Request 4

A request is made to update a property in the *state* transparent data for a subscriber. The property name is *mcc*. Two properties with the name *mcc* exist.

Request URL**Request URL**

```
PUT {baseURI}/msr/sub/MSISDN/33123654862/data/state/mcc
```

Request Content

None.

Response 4

The request fails, as more than one property called *mcc* exists.

HTTP Status Code

400

Response Content

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

5.6.2 Get Data Field**Description**

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

All fields that match the requested *fieldName* are returned.

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

The transparent data field is specified in *fieldName*.

NOTE: The *fieldName* is not case-sensitive. If a field called *mcc* exists, then an attempt to retrieve a field called *MCC* is successful.

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.

A field in the transparent data must exist for the subscriber.

Request URL

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.

- *transparentDataType*: A user defined type/name for the transparent data.
Value is *state* for the State transparent data.
- *fieldName*: A user defined field in the transparent data.
For the State entity this corresponds to a property in the entity.

Request Content

None.

Response Content

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

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

- *transparentDataType*: A user defined type/name for the transparent data.
Value is *state* for the State transparent data.
- *cdataFieldValue*: Contents of the XML data blob with the field from the transparent data.

Table 41: Get Data Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested data field 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 field does not exist

Examples

Request 1

A request is made to get the property *mcc* in the *state* transparent data for a subscriber. The property *mcc* exists.

Request URL

```
GET {BaseURI}/msr/sub/MSISDN/33123654862/data/state/mcc
```

Request Content

None.

Response 1

The request is successful, and the requested property 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>
</state>
]]>
</data>
</subscriber>
```

Request 2

A request is made to get property with name *mcc* in the *state* transparent data for a subscriber. *The* property with name *mcc* does not exist.

Request URL

```
GET {BaseURI}/msr/sub/MSISDN/33123654862/data/state/mcc
```

Request Content

None.

Response 2

The request fails, because the requested property 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 property with name *mcc* in the *state* transparent data for a subscriber. The subscriber has the State entity. Two properties with name *mcc* exist.

Request URL

```
GET {BaseURI}/msr/sub/MSISDN/33123654862/data/state/mcc
```

Request Content

None.

Response 3

The request is successful, and the both the properties are 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>mcc</name>
    <value>400</value>
  </property>
</state>
]]>
</data>
</subscriber>
```

5.6.3 Delete Data Field**Description**

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

The field identifier is specified in *fieldName*.

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

NOTES:

- If the specified field does not exist, this is not considered an error and a successful result is returned.
- The *fieldName* is not case-sensitive. If a field called *mcc* exists, then an attempt to delete a field called *MCC* is successful.

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

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

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *state* for the State transparent data.
- *fieldName*: A user defined field in the transparent data.
For the State entity this corresponds to a property in the entity.

Request Content

None.

Response Content

None.

Table 42: Delete Data Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Requested transparent data field 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 *mcc* property in the *state* transparent data. The *mcc* property exists in the State data.

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/state/mcc
```

Request Content

None.

Response 1

The request is successful, and the property in the State transparent data was deleted.

HTTP Status Code

204

Response Content

None.

Request 2

A request is made to delete the *mcc* property in the *state* transparent data. The *mcc* property does not exist in the state transparent data.

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/state/mcc
```

Request Content

None.

Response 2

The request is successful, even though the *mcc* property does not exist.

HTTP Status Code

204

Response Content

None.

Request 3

A request is made to delete the *mcc* property in the *state* transparent data. The State opaque data is a valid entity, but the requested subscriber does not contain any State opaque data.

Request URL

```
DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/state/mcc
```

Request Content

None.

Response 3

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

HTTP Status Code

404

Response Content

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

5.7 Subscriber Special Operation Commands

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

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

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

Table 43: Summary of Subscriber Special Operation Commands

Command	Description	Keys	Command Syntax
Reset Quota	Reset the fields in the specified Quota	(MSISDN, IMSI, NAI or AccountId) and Row Identifier	POST {BaseURI}/msr/sub/keyName/KeyValue/data/transparentDataType/rowIdValue
	Reset the fields in the specified Quota and instance identifier	or (MSISDN, IMSI, NAI or AccountId), Row Identifier and Instance Identifier	POST {BaseURI}/msr/sub/keyName/KeyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue

5.7.1 Reset Quota

Description

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

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

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

NOTES:

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

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

Without Instance Identifier

```
POST {BaseURI}/msr/sub/keyName/KeyValue/data/transparentDataType/rowIdValue
```

With Instance Identifier

```
POST {BaseURI}/msr/sub/keyName/KeyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- *keyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.
- *transparentDataType*: A user defined type/name for the transparent data.
Value is *quota* for the Quota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* for the Quota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.

Request Content

None.

Response Content

None.

Table 44: Reset Quota 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="MSR4058">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/Q6
```

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, 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 5

A request is made to reset the *Q1* Quota row for a subscriber having *cid* of value *45678*. The subscriber has Quota transparent data, and the Quota transparent data contains a Quota row called *Q1* having *cid* of value *45678*.

Request URL

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

Request Content

None.

Response 5

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

HTTP Status Code

204

Request Content

None.

6. POOL PROVISIONING

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

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

Table 45: Summary of Pool Profile Commands

Command	Description	Keys	Command Syntax
Create Pool	Create a pool/Pool Profile	—	POST <i>{baseURI}/msr/pool</i>
Get Pool	Get Pool Profile data	Pool ID	GET <i>{baseURI}/msr/pool/poolId</i>
Update Pool	Replace an existing Pool Profile		PUT <i>{baseURI}/msr/pool/poolId</i>
Delete Pool	Delete all Pool Profile data and all opaque data associated with the pool		DELETE <i>{baseURI}/msr/pool/poolId</i>

6.1.1 Create Pool

Description

This operation creates a Pool Profile using the field-value pairs that are specified in the request content.

Unlike other pool commands, the key value (Pool ID) is not specified in the URL. Request content includes *poolId*, and field-value pairs, all as specified in the SEC.

NOTES:

- Multi-value fields can be specified by a single *fieldNameX* value with a delimited list of values, or multiple *fieldNameX* fields each containing a single value.
- If the PSO feature is enabled and the Pool ID falls in a range that is maintained by a different UDR instance, then the pool is created as a Non Pool Host UDR pool (remote pool); otherwise the pool is created as a Pool Host UDR pool.
- If the PSO feature is enabled, a pool cannot be provisioned with the Type field on a Non Pool Host UDR system.

Prerequisites

A pool with the supplied Pool ID 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 pool.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="Pool ID">poolId</field>
  [
```

```

<field name="fieldName1">fieldValue1</field>
<field name="fieldName2">fieldValue2</field>
:
<field name="fieldNameN">fieldValueN</field>
]
</pool>

```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *fieldNameX*: A user defined field in the Pool Profile.
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*.

Pool ID/field order in the request is not important

Response Content

None.

Table 46: Create Pool 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 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
400	MSR4070	Operation not allowed

Examples

Request 1

A pool is created, with a Pool ID 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="Pool ID">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 *Pool ID* key. The *BillingDay* and *Entitlement* fields are set. A pool exists with the Pool ID.

Request URL

```
POST {baseURI}/msr/pool
```

Request Content

```

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

```

Response 2

The request fails. The error code indicates the Pool ID exists.

HTTP Status Code

400

Response Content

```

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

```

Request 3

A pool is created, with a *Pool ID* key. PSO feature is enabled. The *Pool ID* falls in a range that is maintained by a different UDR instance.

Request URL

```
POST {baseURI}/msr/pool
```

Request Content

```

<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="Pool ID">100001</field>

```

```

<field name="BillingDay">5</field>
<field name="Entitlement">Weekpass,Daypass</field>
<field name="Type">Enterprise</field>
</pool>

```

Response 3

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

HTTP Status Code

400

Response Content

```

<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4070">errorText</error>

```

6.1.2 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
```

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

Values: 1 to 9999999999999999999999

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="Pool ID">poolId</field>
  [
    <field name="fieldName1">fieldValue1</field>
    <field name="fieldName2">fieldValue2</field>
    :
    <field name="fieldNameN">fieldValueN</field>
  ]
</pool>

```

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

Values: 1 to 9999999999999999999999

- *fieldNameX*: A user defined field in the Pool Profile.

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

Pool ID/field order in the request is not important

Table 47: Get Pool Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Successfully located the pool
404	MSR4001	Could not find the pool by Pool ID

Examples

Request 1

The pool with the Pool ID 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="Pool ID">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 Pool ID is retrieved. The pool does not exist.

Request URL

```
GET {baseURI}/msr/pool/222200
```

Request Content

None.

Response 2

The request fails, as the pool does not exist.

HTTP Status Code

404

Response Content

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

6.1.3 Update Pool**Description**

This operation replaces an existing Subscriber Profile, for the pool identified by *poolId*.

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

NOTES:

- If the Pool ID is included in the content, and it is different from the value specified in the URL, the request fails.
- If the PSO feature is enabled, a pool cannot be updated with the Type field on a Non Pool Host UDR system.

Prerequisites

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

Request URL

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

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

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="Pool ID">poolId</field>
  <field name="fieldName1">fieldValue1</field>
  <field name="fieldName2">fieldValue2</field>
  :
  <field name="fieldNameN">fieldValueN</field>
]
</pool>
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *fieldNameX*: A user defined field in the Pool Profile.

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

Pool ID/field order in the request is not important

Response Content

None.

Table 48: Update Pool 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 Pool ID 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 Pool ID
404	MSR4002	Pool field is not defined
400	MSR4070	Operation not allowed

Examples

Request 1

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 1

The request is successful, and the pool was updated.

HTTP Status Code

204

Response Content

None.

Request 2

A pool is updated with *Type* field being 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="Type">Enterprise</field>
  <field name="Entitlement">Weekpass</field>
  <field name="Entitlement">Daypass</field>
  <field name="Custom15">allo</field>
</pool>
```

Response 1

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

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4070">errorText</error>
```

6.1.4 Delete Pool**Description**

This operation deletes all Pool Profile data and opaque data for the pool that is identified by *poolId*.

Prerequisites

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

The pool must not have any member subscribers, or the request fails.

Request URL

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

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

Request Content

None.

Response Content

None.

Table 49: Delete Pool Response Status/Error Codes

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

Examples**Request 1**

The pool with the Pool ID is deleted. The pool exists and does not have any 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 Pool ID is deleted. The pool exists, but has 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>
```

6.2 Pool Profile Field Commands**Table 50: Summary of Pool Profile Field Commands**

Command	Description	Keys	Command Syntax
Add Field Value	Adds a value to the specified field. This operation does not affect any pre-existing values for the field	Pool ID	<code>POST {baseURI}/msr/pool/poolId/field/fieldName/fieldValue</code>
Get Field	Retrieve values for the specified field		<code>GET {baseURI}/msr/pool/poolId/field/fieldName</code>
Get Field Value	Retrieve the single value for the specified field (if set as specified)		<code>GET {baseURI}/msr/pool/poolId/field/fieldName/fieldValue</code>
Update Field Value	Update field to the specified value		<code>PUT {baseURI}/msr/pool/poolId/field/fieldName/fieldValue</code>
Update Multiple Fields	Update multiple fields to the specified values		<code>PUT {baseURI}/msr/pool/poolId/multipleFields/fieldName1/fieldValue1/fieldName2/fieldValue2/...</code>
Delete Field	Delete all values for the specified field		<code>DELETE {baseURI}/msr/pool/poolId/field/fieldName</code>
Delete Field Value	Delete a value for the specified field		<code>DELETE {baseURI}/msr/pool/poolId/field/fieldName/fieldValue</code>

6.2.1 Add Field Value**Description**

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

This operation can only be used for the fields defined as multi-value field in the SEC. Any pre-existing values for the field are not affected.

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

If a value being added exists, the request fails.

NOTES:

- If the field to which the value is being added does not exist, it is created.
- The *fieldValue* is case-sensitive. An attempt to add the value a to a field value of a;b;c fails., but an attempt to add the value A is successful and result in the field value being a;b;c;A

Prerequisites

A pool with the Pool ID of the *poolId* supplied must exist.

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

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

Request URL

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

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *fieldName*: A user defined field in the Pool Profile.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, a;b;c.
- The semicolon between the field values may need to be encoded as %3B for certain clients.

Request Content

None.

Response Content

None.

Table 51: Add Field Value 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 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 present in the *Entitlement* field.

Request URL

```
POST {baseURI}/msr/pool/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 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.

6.2.2 Get Field

Description

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

NOTE: Depending on the field, there may be multiple field-value pairs returned by this operation.

Prerequisites

A pool with the Pool ID of the *poolId* supplied must exist.

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

Request URL

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

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *fieldName*: A user defined field in the Pool Profile.

Request Content

None.

Response Content

A `<pool>` element that contains a `<field>` element for every value defined for the specified field in 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>
```

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

Table 52: Get Field 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

Examples**Request 1**

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 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="Entitlement">Weekpass</field>
  <field name="Entitlement">Daypass</field>
</pool>
```

6.2.3 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, return success, regardless of how many other values may exist in the Pool Profile. If any or all of the values are not present as part of the Pool Profile, an error is returned.

NOTES:

- Depending on the field, there may be multiple field-value pairs returned by this operation.
- The *fieldValue* is case-sensitive. An attempt to get the value a from a field value of a;b;c is successful, but an attempt to get the value A fails.

Prerequisites

A pool with the Pool ID of the *poolId* supplied must exist.

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

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

Request URL

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

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 9999999999999999999999
- *fieldName*: A user defined field in the Pool Profile.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, a;b;c.

- 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>
```

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

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested field exists for pool
400	MSR4053	Pool and field exist, but values 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

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/pool/300000/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"?>
<pool>
  <field name="Entitlement">DayPass</field>
  <field name="Entitlement">HighSpeedData</field>
</pool>
```

6.2.4 Update Field**Description**

This operation updates a field to the specified value for the pool identified by the specified *poolId*.

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 set specified is inserted. Adding values to a current set is accomplished using Add Field Value.

NOTE: This command cannot be used to update the Pool ID.

Prerequisites

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

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

Request URL

```
PUT {baseURI}/msr/pool/poolId/field/fieldName/fieldValue
```

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

- *fieldName*: A user defined field in the Pool Profile.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, a;b;c.
- The semicolon between the field values may need to be encoded as %3B for certain clients.

Request Content

None.

Response Content

None.

Table 53: Update Field 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

Examples**Request 1**

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
```

Request Content

None.

Response 1

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

HTTP Status Code

201

Response Content

None.

6.2.5 Update Multiple Fields**Description**

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

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 set specified is inserted. Adding values to a current set is accomplished using Add Field Value.

This command updates multiple fields in a single command for pool data. ALL fields that can be modified in the single field request can also be modified in the multiple field request. Two or three fields can be updated at once. Updating only a single field results 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 the Pool ID.

Prerequisites

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

The fields *fieldNameX* must all be valid fields in the Pool Profile.

Request URL

```
PUT {baseURI}/msr/pool/poolId/multipleFields/fieldName1/fieldValue1/
fieldName2/fieldValue2/[fieldName3/fieldValue3]
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 9999999999999999999999
- *fieldNameX*: A user defined field in the Pool Profile.
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, a;b;c.
- The semicolon between the field values may need to be encoded as %3B for certain clients.

Request Content

None.

Response Content

None.

Table 54: Update Multiple Fields 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
400	MSR4057	Request only contains one field to update
404	MSR4001	Pool does not exist
404	MSR4002	Pool field is not defined

Examples**Request 1**

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 1

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

HTTP Status Code

201

Response Content

None.

6.2.6 Delete Field**Description**

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

If the field is multi-value field then all values are deleted. Deletion of a field results in the removal of the field from the Pool Profile. The field is not present, not just the value is empty.

NOTES:

- The field being deleted does not need to have a current value. It can be empty (deleted), and the request succeeds.
- This command cannot be used to delete the Pool ID.
- If the field being deleted is mandatory, and is defined as having a default value, then the field is not removed, but has 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
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *fieldName*: A user defined field in the Subscriber Profile.

Request Content

None.

Response Content

None.

Table 55: Delete Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Field was successfully deleted
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
404	MSR4001	Pool does not exist
404	MSR4002	Pool field is not defined
400	MSR4101	Enterprise to Basic Pool Conversion failed threshold exceeded

Examples**Request 1**

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 1

The request is successful, and the field was deleted.

HTTP Status Code

204

Response Content

None.

Request 2

A request is made to delete the *Type* field from an enterprise pool that has > the maximum members allowed for a basic pool. (Note: deleting the *Type* field triggers a conversion from an enterprise pool to a basic pool).

Request URL

```
DELETE {BaseURI}/msr/pool/100000/field/Type
```

Request Content

None.

Response 2

The request fails, because the enterprise to basic pool conversion failed because the pool has more members than the maximum threshold for a basic pool.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4101">errorText</error>
```

6.2.7 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 used for the fields defined as multi-value field in the SEC.

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

with *c* as the value. If the profile contains *a;b;c* and a request is made to delete *c;d* the request succeeds and the profile is left with *a;b* as the value.

If all values are removed, the field is removed from the Pool Profile (an XML element is not present).

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

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
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *fieldName*: A user defined field in the Pool Profile.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

NOTES:

- For multi-value fields, the value contains a semicolon separated list of values on a single line. For example, *a;b;c*.
- The semicolon between the field values may need to be encoded as *%3B* for certain clients.

Request Content

None.

Response Content

None.

Table 56: Delete Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Requested fields 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, and 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

Request Content

None.

Request 2

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/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.

6.3 Pool Opaque Data Commands

Table 57: Summary of Pool Opaque Data Commands

Command	Description	Keys	Command Syntax
Set Opaque Data	Create or update opaque data of the specified type	Pool ID	<code>PUT {baseURI}/msr/pool/poolId/data/opaqueDataType</code>
Get Opaque Data	Retrieve opaque data of the specified type		<code>GET {baseURI}/msr/pool/poolId/data/opaqueDataType</code>
Delete Opaque Data	Delete opaque data of the specified type		<code>DELETE {baseURI}/msr/pool/poolId/data/opaqueDataType</code>

6.3.1 Set Opaque Data

Description

This operation updates (or creates if it not exists) the 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.

NOTES:

- 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.
- If the PSO feature is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

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

Request URL

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

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either *poolquota*, *poolstate*, or *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>

```

- *opaqueDataType*: A user defined type/name for the opaque data. Value is either poolquota, poolstate, or pooldynamicquota.
- *CDATAFieldValue*: Contents of the XML data blob.

NOTE: The *opaqueDataType* in the request content is currently ignored, and is not validated. The *opaqueDataType* in the URL is solely used to identify the opaque data type.

Response Content

None.

Table 58: Set Opaque Data Response Status/Error Codes

Set Opaque Data 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	Pool is not found
404	MSR4049	Data type is not defined

Example

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>3</version>
<quota name="AggregateLimit">
  <cid>9223372036854775807</cid>
  <time>3422</time>
  <totalVolume>1000</totalVolume>
  <inputVolume>980</inputVolume>
  <outputVolume>20</outputVolume>
  <serviceSpecific>12</serviceSpecific>
  <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
</quota>
</usage>
]]>
</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 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>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]>
</data>
</pool>

```


Response 2

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

HTTP Status Code

201

Response Content

None.

6.3.2 Get Opaque Data**Description**

This operation retrieves the opaque data of the specified *opaqueDataType* for the pool identified by the *poolId* in the request.

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

NOTE: If the PSO feature is enabled, this operation on the Non Pool Host UDR for the specified pool would return success with empty entity data as the pool entity data is only stored on the Pool Host UDR.

Prerequisites

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

The *opaqueDataType* must reference a valid pooled 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
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999999999999999
- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either poolquota, poolstate, or 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>
```

- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either poolquota, poolstate, or pooldynamicquota.
- *cdataFieldValue*: Contents of the XML data blob.

Table 59: Get Opaque Data Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested opaque data exists for pool
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4053	Data type is not set for this pool
204	—	No Content returned

Example**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>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
```

```

    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </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>

```

Request 3

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

Request URL

```
GET {baseURI}/msr/pool/100000/data/poolquota
```

Request Content

None.

Response 3

The request is successful, and the specified pool on NPHO does not return poolquota data.

HTTP Status Code

204

Response Content

None.

6.3.3 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.

NOTES:

- If the opaque data of the *opaqueDataType* does not exist for the pool, this is not considered an error and a successful result is returned.
- If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

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

Request URL

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

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *opaqueDataType*: A user defined type/name for the opaque data.
Value is either poolquota, poolstate, or pooldynamicquota.

Request Content

None.

Response Content

None.

Table 60: Delete Opaque Data Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Opaque data was successfully deleted
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined

Example**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 is successful although PoolState opaque data was not deleted.

HTTP Status Code

204

Response Content

None.

6.4 Pool Data Row Commands

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

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

NOTE: Pool data row commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an HTTP Status Code 400, with an MSR4070 error being returned.

Table 61: Summary of Pool Data Row Commands

Command	Description	Keys	Command Syntax
Set Row	Create or update data row in data of the specified type.	Pool ID and Row Identifier Or Pool ID and Row Identifier and Instance Identifier	<code>PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue</code>
	Create or update data row in data of the specified type and instance identifier.		<code>PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue</code>
Get Row	Retrieve data row from data of the specified type.		<code>GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue</code>
	Retrieve data row from data of the specified type and instance identifier.		<code>GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue</code>
Delete Row	Delete data row in data of the specified type		<code>DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue</code>
	Delete data row in data of the specified type and instance identifier		<code>DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue</code>

6.4.1 Set Row**Description**

This operation creates a data row or updates an existing data row for the pool identified by the *poolId*.

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

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

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

NOTES:

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

- If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

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

Request URL

Without Instance Identifier

```
PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue
```

With Instance Identifier

```
PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *poolquota* for the PoolQuota transparent data.
 - Value is *pooldynamicquotafor* for the PoolDynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the data blob.
There is not a *rowIdValue* for State transparent data.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the PoolQuota transparent data.
 - Value is *InstanceId* or *Type* for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.

Request Content

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

- *rowValue*: Contents of the XML data blob with the row data.

NOTE: The *rowValue* is the same format as a PoolQuota entity, the single row being added

NOTE: The data in *rowValue* contains 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.

Table 62: Set Row 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	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4067	Multiple matching rows found
404	MSR4001	Pool 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 *poolquota* transparent data for a pool. The data row identifier field value is *AggregateLimit*. The pool does not have an existing PoolQuota row called *AggregateLimit*.

Request URL

```
PUT {baseURI}/msr/pool/100000/data/poolquota/AggregateLimit
```

Request Content

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
```

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 *poolquota* transparent data for a pool. The data row identifier field value is *PQ1*. The pool has an existing PoolQuota row called *PQ1*.

Request URL

```
PUT {baseURI}/msr/pool/100000/data/poolquota/PQ1
```

Request Content

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="PQ1">
    <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 *PQ1* was updated.

HTTP Status Code

201

Response Content

None.

Request 3

A request is made to update a data row in the *pooldynamicquota* transparent data for a pool. The data row identifier field value is *AggregateLimit*. The *AggregateLimit* data row exists in the PoolDynamicQuota data, but there are two rows called *AggregateLimit* one with *InstanceId* of 15570, the other with a *InstanceId* of 15678. The request is not required in the response.

Request URL

```
PUT {baseURI}/msr/pool/100000/data/pooldynamicquota/AggregateLimit/row/InstanceId/15678
```

Request Content

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
```

```

<DynamicQuota name="AggregateLimit">
  <Type>pass</Type>
  <InstanceId>15678</InstanceId>
  <Priority>4</Priority>
  <InitialTime>135</InitialTime>
  <InitialTotalVolume>2000</InitialTotalVolume>
  <InitialInputVolume>1500</InitialInputVolume>
  <InitialOutputVolume>500</InitialOutputVolume>
  <InitialServiceSpecific>4</InitialServiceSpecific>
  <activationdatetime>2015-05-22T00:00:00-05:00</activationdatetime>
  <expirationdatetime>2015-05-29T00:00:00-05:00</expirationdatetime>
  <InterimReportingInterval>100</InterimReportingInterval>
  <Duration>10</Duration>
</DynamicQuota>
</definition>

```

Response 3

The request is successful, and the data row *AggregateLimit* with *InstanceId* of *15678* was updated.

HTTP Status Code

201

Response Content

None.

6.4.2 Get Row**Description**

This operation retrieves a transparent data row for the pool identified by the *poolId*. 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*.

NOTES:

- The *rowIdValue* is case-sensitive. If a row called *DayPass* exists, then an attempt to get a row called *DayPass* is successful, but an attempt to get a row called *DAYPASS* fails
- If PSO is enabled, this operation on the Non Pool Host UDR for the specified pool would return success with empty data as the pool entity data is only stored on the Pool Host UDR.

Prerequisites

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

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

A data row with the identifier in the transparent data must exist for the pool.

Request URL**Without Instance Identifier**

```
GET {baseURI}/msr/pool/{poolId}/data/{transparentDataType}/rowIdValue
```

With Instance Identifier

```
GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- **poolId:** Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- **transparentDataType:** A user defined type/name for the transparent data.
 - Value is `poolquota` for the PoolQuota transparent data.
 - Value is `pooldynamicquota` for the PoolDynamicQuota transparent data.
- **rowIdValue:** The row name value that identifies the row in the transparent data blob.
- **instanceFieldName:** A user defined field in the data row that is used to define a unique row instance.
 - Value is `cid` or `Type` for the PoolQuota transparent data.
 - Value is `Instanceid` or `Type` for the PoolDynamicQuota transparent data.
- **instanceFieldValue:** Corresponding field value assigned to *instanceFieldName*.

Request Content

None.

Response Content

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

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

- **transparentDataType:** A user defined type/name for the transparent data.
 - Value is `poolquota` for the PoolQuota transparent data.
 - Value is `pooldynamicquota` for the PoolDynamicQuota transparent data.
- **cdataRowValue:** Contents of the XML data blob with the row data.

Table 63: Get Row Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested data row exists for pool
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found

HTTP Status Code	Error Code	Description
404	MSR4059	Data row does not exist
204	—	No Content returned

Examples

Request 1

A request is made to get the *PQ1* data row from the *poolquota* transparent data for a pool. The pool has the PoolQuota entity, and the *PQ1* data row exists.

Request URL

```
GET {baseURI}/msr/pool/100000/data/poolquota/PQ1
```

Request Content

None.

Response 1

The request is successful, and the PoolQuota transparent data row requested 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>3</version>
  <quota name="PQ1">
    <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>
</pool>
```

Request 2

A request is made to get the Weekend data row from the poolquota transparent data for a pool. The pool has the PoolQuota entity, but and the Weekend data row does not exist.

Request URL

```
GET {baseURI}/msr/pool/100000/data/poolquota/Weekend
```

Request Content

None.

Response 2

The request fails, as 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 *poolquota* transparent data for a pool. The pool has the PoolQuota entity. Two instances of the *Weekday* data row exist.

Request URL

```
GET {baseURI}/msr/pool/100000/data/poolquota/Weekday
```

Request Content

None.

Response 3

The request is successful, and the PoolQuota transparent data rows requested are 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>3</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>
</pool>

```

Request 4

A request is made to get the *PDQ1* data row from the *pooldynamicquota* transparent data for a pool with the *InstanceId* value of *11223344*. The *PoolDynamicQuota* data contains four rows called *PDQ1*. Two with *<InstanceId>* of *11223344*, one with an *InstanceId* of *99887766*, and one with an *InstanceId* of *55556666*. The request is not required in the response.

Request URL

```
GET {baseURI}/msr/pool/10000/data/pooldynamicquota/PDQ1/InstanceId/11223344
```

Request Content

None.

Response 4

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

HTTP Status Code

200

Response Content

```

<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="pooldynamicquota">
    <![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="PDQ1">
    <Type>pass</Type>
    <InstanceId>11223344</InstanceId>
    <Priority>4</Priority>
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
  <DynamicQuota name="PDQ1">
    <Type>topup</Type>
    <InstanceId>11223344</InstanceId>
    <Priority>4</Priority>
    <InterimReportingInterval>200</InterimReportingInterval>
    <Duration>20</Duration>
  </DynamicQuota>
</definition>]]>
  </data>
</pool>

```

6.4.3 Delete Row

Description

This operation deletes a transparent data row for the pool identified by the *poolId*.

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.

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to delete a row called DayPass is successful, but an attempt to delete a row called DAYPASS fails
- The deletion of a non-existent data row is not considered an error.
- If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

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

Request URL

Without Instance Identifier

```
DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue
```

With Instance Identifier

```
DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *poolquota* for the PoolQuota transparent data.
 - Value is *pooldynamicquota* for the PoolDynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the PoolQuota transparent data.
 - Value is *InstanceId* or *Type* for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.

Request Content

None.

Response Content

None.

Table 64: Delete Row Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Data row was successfully deleted
400	MSR4064	Occurrence constraint violation
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 delete the *PQ1* data row in the *poolquota* transparent data. The *PQ1* data row exists in the PoolQuota data.

Request URL

```
DELETE {baseURI}/msr/pool/100000/data/poolquota/PQ1
```

Request Content

None.

Response 1

The request is successful, and the data row in the PoolQuota 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 *poolquota* transparent data. The *Weekend* data row does not exist in the PoolQuota transparent data.

Request URL

```
DELETE {baseURI}/msr/pool/100000/data/poolquota/Weekend
```

Request Content

None.

Response 2

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

HTTP Status Code

204

Response Content

None.

Request 3

A request is made to delete the *PDQ1* data row in the *pooldynamicquota* transparent data. The *PDQ1* data row exists in the PoolDynamicQuota data.

Request URL

```
DELETE {baseURI}/msr/pool/10000/data/pooldynamicquota/PDQ1
```

Request Content

None.

Response 3

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

HTTP Status Code

204

Response Content

None.

6.5 Pool Data Row Field Commands

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

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

NOTE: Pool data row field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an HTTP Status Code 400, with an MSR4070 error being returned.

Table 65: Summary of Pool Data Row Field Commands

Command	Description	Keys	Command Syntax
Get Row Field	Retrieve values for the specified field	Pool ID and Row Identifier and Field name	GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName
	Retrieve values for the specified field and instance identifier		GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName

Command	Description	Keys	Command Syntax
Get Row Field Value	Retrieve a single value for the specified field	Or Pool ID and Row Identifier, Instance Identifier and Field name	GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName/fieldValue
	Retrieve a single value for the specified field and instance identifier		GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName/fieldValue
Update Row Field	Update field to the specified value		PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName/fieldValue
	Update field to the specified value and instance identifier		PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName/fieldValue
Delete Row Field	Delete all values for the specified field		DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName
	Delete all values for the specified field and instance identifier		DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName

6.5.1 Get Row Field

Description

This operation retrieves a field in a transparent data row for the pool identified by the *poolId*.

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

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

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

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to get a field in a row called DayPass is successful, but an attempt to get a field in a row called DAYPASS fails
- If PSO is enabled, this operation on the Non Pool Host UDR for the specified pool would return success with empty data as the pool entity data is only stored on the Pool Host UDR.

Prerequisites

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

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

A data row with the identifier in the transparent data must exist for the pool.

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

Request URL

Without Instance Identifier

```
GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName
```

With Instance Identifier

```
GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName
```

- **poolId**: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 9999999999999999999999
- **transparentDataType**: A user defined type/name for the transparent data.
 - Value is `poolquota` for the PoolQuota transparent data.
 - Value is `pooldynamicquota` for the PoolDynamicQuota transparent data.
- **rowIdValue**: The row name value that identifies the row in the transparent data blob.
- **instanceFieldName**: A user defined field in the data row that is used to define a unique row instance.
 - Value is `cid` or `Type` for the PoolQuota transparent data.
 - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data.
- **instanceFieldValue**: Corresponding field value assigned to *instanceFieldName*.
- **fieldName**: A user defined field in the transparent data row.

Request Content

None.

Response Content

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

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

- **transparentDataType**: A user defined type/name for the transparent data.
 - Value is `poolquota` for the PoolQuota transparent data.
 - Value is `pooldynamicquota` for the PoolDynamicquota transparent data.
- **cdataRowFieldValue**: Contents of the XML data blob, with the field from the row data.

Table 66: Get Row Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested data row field exists for pool

HTTP Status Code	Error Code	Description
404	MSR4001	Pool 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
204	—	No Content returned

Examples

Request 1

A request is made to get the *inputVolume* field in the *PQ1* data row of the *poolquota* transparent data for a pool.

Request URL

```
GET {BaseURI}/msr/pool/100000/data/poolquota/PQ1/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"?>
<pool>
<data name="poolquota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
<version>3</version>
<quota name="PQ1">
<inputVolume>980</inputVolume>
</quota>
</usage>
]]>
</data>
</pool>
```

Request 2

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

Request URL

```
GET {BaseURI}/msr/pool/100000/data/poolquota/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"?>
<pool>
<data name="poolquota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Weekday">
    <inputVolume>980</outputVolume>
  </quota>
  <quota name="Weekday">
    <inputVolume>2140</outputVolume>
  </quota>
</usage>
]]>
</data>
</pool>
```

Request 3

A request is made to get the *InitialInputVolume* field in the *PDQ1* data row of the *pooldynamicquota* transparent data having an *InstanceId* of value *11223344* for a pool.

Request URL

```
GET {BaseURI}/msr/pool/100000/data/pooldynamicquota/PDQ1/row/InstanceId/11223344/InitialInputVolume
```

Request Content

None.

Response 3

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

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
<data name="pooldynamicquota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="PDQ1">
    <InitialInputVolume>15678</InitialInputVolume>
  </DynamicQuota>
</definition>
]]>
</data>
</pool>
```

6.5.2 Get Row Field Value**Description**

This operation retrieves a field with a value in a transparent data row for the pool identified by the *poolId*.

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

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to get a field value in a row called DayPass is successful, but an attempt to get a field value in a row called DAYPASS fails
- The *fieldValue* is case-sensitive. An attempt to get the value Data from a current field value of Data is successful, but an attempt to get the value DATA fails.
- If PSO is enabled, this operation on the Non Pool Host UDR for the specified pool would return success with empty data as the pool entity data is only stored on the Pool Host UDR.

Prerequisites

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

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

A data row with the identifier in the transparent data must exist for the pool.

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**Without Instance Identifier**

```
GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName/fieldValue
```

With Instance Identifier

```
GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName/fieldValue
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *poolquota* for the PoolQuota transparent data.
 - Value is *pooldynamicquota* for the PoolDynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the PoolQuota transparent data.
 - Value is *InstanceId* or *Type* for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.
- *fieldName*: A user defined field in the transparent data row.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

Request Content

None.

Response Content

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

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

- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *poolquota* for the PoolQuota transparent data.
 - Value is *pooldynamicquota* for the PoolDynamicQuota transparent data.
- *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.

Table 67: Get Row Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested data row field/value exists for pool
400	MSR4053	Data row field value does not match
404	MSR4001	Pool is not found
404	MSR4002	Field is not defined for this data type

HTTP Status Code	Error Code	Description
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist
204	—	No Content returned

Examples

Request 1

A request is made to get the *inputVolume* field with the value of *980* in the *PQ1* data row of the *poolquota* transparent data for a pool. The *inputVolume* field exists, and is set to the value *980*.

Request URL

```
GET {BaseURI}/msr/pool/100000/data/poolquota/PQ1/inputVolume/980
```

Request Content

None.

Response 1

The request is successful, and the requested field with the specified value 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>3</version>
<quota name="PQ1">
<inputVolume>980</inputVolume>
</quota>
</usage>
]]>
</data>
</pool>
```

Request 2

A request is made to get the *outputVolume* field with the value of *2000* in the *PQ4* data row of the *poolquota* transparent data for a pool. The *outputVolume* field exists, but is set to the value *1500*.

Request URL

```
GET {BaseURI}/msr/pool/100000/data/poolquota/PQ1/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 3220 in the *Weekday* data row of the *poolquota* transparent data for a pool. 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/pool/100000/data/poolquota/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"?>
<pool>
<data name="poolquota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Weekday">
    <inputVolume>3220</inputVolume>
  </quota>
  <quota name="Weekday">
    <inputVolume>3220</inputVolume>
  </quota>
</usage>
]]>
</data>
</pool>
```

Request 4

A request is made to get the *InitialTotalVolume* field with the value of *980* in the *PDQ1* data row of the *pooldynamicquota* transparent data for a pool. The *InitialTotalVolume* field exists, and is set to the value *980*.

Request URL

```
GET {BaseURI}/msr/pool/33123654862/data/pooldynamicquota/PDQ1/row/InitialTotalVolume/2000
```

Request Content

None.

Response 4

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

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
<data name="pooldynamicquota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="PDQ1">
    <InitialTotalVolume>2000</InitialTotalVolume>
  </DynamicQuota>
</denition>
]]>
</data>
</pool>
```

6.5.3 Update Row Field**Description**

This operation updates a fields in a transparent data row for the pool identified by the *poolId*.

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 is created.

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

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to update a field in a row called DayPass is successful, but an attempt to update a field in a row called DAYPASS fails
- If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

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

A data row with the identifier in the transparent data exists for the pool.

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

Request URL

Without Instance Identifier

```
PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName/fieldValue
```

With Instance Identifier

```
PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue/fieldName/fieldValue
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 9999999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *poolquota* for the PoolQuota transparent data.
 - Value is *pooldynamicquota* for the PoolDynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the PoolQuota transparent data.
 - Value is *Instanceid* or *Type* for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.
- *fieldName*: A user defined field in the transparent data row.
- *fieldValue*: Corresponding field value assigned to *fieldname*.

Request Content

None.

Response Content

None.

Table 68: Update Row Field 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

HTTP Status Code	Error Code	Description
404	MSR4001	Pool 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 *inputVolume* field in the *PQ1* data row of the *poolquota* transparent data for a pool.

Request URL

```
PUT {BaseURI}/msr/pool/100000/data/poolquota/PQ1/inputVolume/0
```

Request Content

None.

Response 1

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

HTTP Status Code

201

Response Content

None.

Request 2

A request is made to update the *cid* field in the *PQ1* data row in the *poolquota* transparent data. The *cid* field is not allowed to be updated.

Request URL

```
PUT {BaseURI}/msr/pool/100000/data/poolquota/PQ1/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 pool. Two instances of the *Weekday* data row exist.

Request URL

```
PUT {BaseURI}/msr/pool/100000/data/poolquota/Weekday/inputVolume/0
```

Request Content

None.

Request 4

A request is made to update the *InitialTotalVolume* field in the *PDQ1* data row of the *PoolDynamicQuota* transparent data for a pool.

Request URL

```
PUT {BaseURI}/msr/pool/33123654862/data/pooldynamicquota/PDQ1/row/InitialTotalVolume/2000
```

Request Content

None.

Response 4

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

HTTP Status Code

201

Response Content

None.

6.5.4 Delete Row Field**Description**

This operation deletes a field in a transparent data row for the pool identified by the *poolId*.

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 fails.

NOTES:

- If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and row/field data is not deleted.
- If the field with opaque data of the *opaqueDataType* does not exist, this is not considered an error and a successful result is returned.
- If the field being deleted is mandatory, and is defined as having a default value, then the field is not removed, but has the default value assigned.
- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to delete a field in a row called DayPass is successful, but an attempt to delete a field in a row called DAYPASS fails
- If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

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

A data row with the identifier in the transparent data must exist for the pool.

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

Request URL**Without Instance Identifier**

```
DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName
```

With Instance Identifier

```
DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/row/instanceFieldName/instanceFieldValue/rowIdValue/fieldName
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
 - Value is *poolquota* for the PoolQuota transparent data.
 - Value is *pooldynamicquota* for the PoolDynamicQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* or *Type* for the PoolQuota transparent data.
 - Value is *InstanceId* or *Type* for the PoolDynamicQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.
- *fieldName*: A user defined field in the transparent data row.

Request Content

None.

Response Content

None.

Table 69: Delete Row Field 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
400	MSR4064	Occurrence constraint violation
404	MSR4001	Pool 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 *PQ1* data row of the *poolquota* transparent data for a pool.

Request URL

```
DELETE {BaseURI}/msr/pool/100000/data/poolquota/PQ1/inputVolume
```

Request Content

None.

Response 1

The request is successful, and the field in the data row in the PoolQuota 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 of the *poolquota* transparent data for a pool. Two instances of the *Weekday* data row exist.

Request URL

```
DELETE {BaseURI}/msr/pool/100000/data/poolquota/Weekday/inputVolume
```

Request Content

None.

Response 2

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 3

A request is made to delete the *InitialTotalVolume* field in the *PDQ1* data row of the *pooldynamicquota* transparent data for a pool.

Request URL

```
DELETE {BaseURI}/msr/pool/33123654862/data/pooldynamicquota/PDQ1/InitialTotalVolume
```

Request Content

None.

Response 3

The request is successful, and the field in the data row in the *PoolDynamicQuota* transparent data was deleted.

HTTP Status Code

204

Response Content

None.

6.6 Pool Data Field Commands

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

```
<property>
  <name>X</name>
```



```
<value>Y</value>
</property>
```

The data field commands allow operations (create, retrieve, update, or delete) at the field level. The required field is identified in the request by the `FieldName`.

NOTE: Pool data field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an HTTP Status Code 400, with an MSR4070 error being returned.

Table 70: Summary of Pool Data Field Commands

Command	Description	Keys	Command Syntax
Set Data Field	Create or update data field in transparent data of the specified type.	Pool ID and Field Name	<code>POST {baseURI}/msr/pool/poolId/data/transparentDataType/fieldName/fieldValue</code>
			<code>PUT {baseURI}/msr/pool/poolId/data/transparentDataType/fieldName/fieldValue</code>
Get Data Field	Retrieve data field from transparent data of the specified type.		<code>GET {baseURI}/msr/pool/poolId/data/transparentDataType/fieldName</code>
Delete Data Field	Delete data field in transparent data of the specified type		<code>DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/fieldName</code>

6.6.1 Set Data Field

Description

This operation creates a field or updates an existing field in a transparent data for the pool identified by the `poolId`.

The field name is specified in `fieldName`, and the field value is specified in `fieldValue`.

If more than one existing fields matches the requested `fieldName`, then the update request fails.

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

NOTES:

- The `fieldName` is not case-sensitive. If a field called `mcc` exists, then an attempt to update an existing field called `MCC` is successful.
- If the transparent entity specified in `entityName` does not exist for the pool, it is created.
- If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

The `transparentDataType` must reference a valid pooled transparent entity in the Interface Entity Map table in the SEC.

Request URL**Format 1**

```
PUT {baseURI}/msr/pool/poolId/data/transparentDataType/fieldName/fieldValue
```

Format 2

```
POST {baseURI}/msr/pool/poolId/data/transparentDataType/fieldName/fieldValue
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
Value is *poolstate* for the PoolState transparent data.
- *fieldName*: A user defined field in the transparent data.
 - For the PoolState entity this corresponds to a property in the entity.
 - The *fieldName* is stored exactly as it is sent in the request. The case of *fieldName* changes if an update is performed using a different case.
- *fieldValue*: Corresponding field value assigned to *fieldName*.

Request Content

None.

Response Content

None.

Table 71: Set Data Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	—	Data field was successfully created/updated
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4067	Multiple matching fields found
404	MSR4001	Pool 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 property in the *poolstate* transparent data for a pool. The property name is *mcc* and the property value is *315*. The pool does not have an existing PoolState property called *mcc*.

Request URL

```
POST {baseURI}/msr/pool/10000/data/poolstate/mcc/315
```

Request Content

None.

Response 1

The request is successful, and the property *mcc* with value *315* was created.

HTTP Status Code

201

Response Content

None.

Request 2

A request is made to create a property in the *poolstate* transparent data for a pool. The property name is *mcc* and the property value is *315*. The pool does not have an existing PoolState property called *mcc*. The pool does not have the PoolState transparent data.

Request URL

```
PUT {baseURI}/msr/pool/10000/data/poolstate/mcc/315
```

Request Content

None.

Response 2

The request is successful, and the property *mcc* as well as the PoolState entity is created.

HTTP Status Code

201

Response Content

None.

Request 3

A request is made to update a property in the *poolstate* transparent data for a pool. The property name is *mcc*. The pool has an existing PoolState property called *mcc*.

Request URL

```
POST {baseURI}/msr/pool/10000/data/poolstate/mcc/400
```

Request Content

None.

Response 3

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

HTTP Status Code

201

Response Content

None.

Request 4

A request is made to update a property in the *poolstate* transparent data for a pool. The property name is *mcc*. Two properties with the name *mcc* exist.

Request URL**Request URL**

```
PUT {baseURI}/msr/pool/10000/data/poolstate/mcc
```

Request Content

None.

Response 4

The request fails, as more than one property called *mcc* exists.

HTTP Status Code

400

Response Content

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

6.6.2 Get Data Field**Description**

This operation retrieves a data field in a transparent data for the pool identified by the *poolId*.

All fields that match the requested *fieldName* are returned.

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

The transparent data field is specified in *fieldName*.

NOTES:

- The *fieldName* is not case-sensitive. If a field called *mcc* exists, then an attempt to retrieve a field called *MCC* is successful.
- If the PSO feature is enabled, this operation on the Non Pool Host UDR for the specified pool would return success with empty entity data as the pool entity data is only stored on the Pool Host UDR.

Prerequisites

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

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

A field in the transparent data must exist for the pool.

Request URL

```
GET {baseURI}/msr/pool/poolId/data/transparentDataType/fieldName
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
Value is *poolstate* for the PoolState transparent data.
- *fieldName*: A user defined field in the transparent data.

NOTE: For the PoolState entity this corresponds to a property in the entity.

Request Content

None.

Response Content

A `<pool>` element that contains a `<data>` element, which contains the specified transparent data field (if it exists) for the identified pool.

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

- *transparentDataType*: A user defined type/name for the transparent data.
Value is *poolstate* for the PoolState transparent data.
- *cdataFieldValue*: Contents of the XML data blob with the field from the transparent data.

Table 72: Get Data Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Requested data field exists for pool

HTTP Status Code	Error Code	Description
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data field does not exist
204	—	No Content returned

Examples

Request 1

A request is made to get the property *mcc* in the *poolstate* transparent data for a pool. The property *mcc* exists.

Request URL

```
GET {BaseURI}/msr/pool/10000/data/poolstate/mcc
```

Request Content

None.

Response 1

The request is successful, and the requested property 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>P3</name>
    <value>200</value>
  </property>
</state>
]]>
</data>
</pool>
```

Request 2

A request is made to get property with name *mcc* in the *poolstate* transparent data for a pool. *The* property with name *mcc* does not exist.

Request URL

```
GET {BaseURI}/msr/pool/10000/data/poolstate/mcc
```

Request Content

None.

Response 2

The request fails, because the requested property does not exist.

HTTP Status Code

400

Response Content

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

6.6.3 Delete Data Field**Description**

This operation deletes a data field in a transparent data for the pool identified by the *poolId*.

The field identifier is specified in *fieldName*.

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

NOTES:

- If the specified field does not exist, this is not considered an error and a successful result is returned.
- The *fieldName* is not case-sensitive. If a field called *mcc* exists, then an attempt to delete a field called *MCC* is successful.
- If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

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

Request URL

```
DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/fieldName
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 9999999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
Value is *poolstate* for the PoolState transparent data.
- *fieldName*: A user defined field in the transparent data.

Note: For the PoolState entity this corresponds to a property in the entity.

Request Content

None.

Response Content

None.

Table 73: Delete Data Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Requested transparent data field was successfully deleted
400	MSR4067	Multiple matching fields found
400	MSR4064	Occurrence constraint violation
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 delete the *mcc property* in the *poolstate* transparent data. The *mcc* property exists in the PoolState data.

Request URL

```
DELETE {baseURI}/msr/pool/10000/data/poolstate/mcc
```

Request Content

None.

Response 1

The request is successful, and the property in the PoolState transparent data was deleted.

HTTP Status Code

204

Response Content

None.

Request 2

A request is made to delete the *mcc property* in the *poolstate* transparent data. The *mcc* property does not exist in the PoolState transparent data.

Request URL

```
DELETE {baseURI}/msr/pool/10000/data/poolstate/mcc
```

Request Content

None.

Response 2

The request is successful, even though the *mcc* property does not exist.

HTTP Status Code

204

Response Content

None.

Request 3

A request is made to delete the *mcc* property in the *poolstate* transparent 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/10000/data/poolstate/mcc
```

Request Content

None.

Response 3

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>
```

6.7 Additional Pool Commands

Table 74: Summary of Additional Pool Commands

Command	Description	Keys	Command Syntax
Add Member to Pool	Add subscriber to a Pool	Pool ID 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

Command	Description	Keys	Command Syntax
Get Pool Members	Retrieve pool member subscribers by Pool ID	Pool ID	GET {BaseURI}/msr/pool/ <i>poolId</i> /member
Get Pool ID	Retrieve Pool ID for specified member subscriber	(MSISDN, IMSI, NAI or AccountId)	GET {BaseURI}/msr/sub/ <i>subKeyName</i> / <i>subKeyValue</i> /pool
Get All Pool Members	Retrieve pool member subscribers from all local or local/remote systems by Pool ID	Pool ID	GET {BaseURI}/msr/pool/ <i>poolId</i> /allmembers/AddressList/ <i>addresslist</i> /PSO/ <i>pso</i>

6.7.1 Add Member to Pool

Description

This operation adds a Subscriber to a Pool.

When the PSO flag is enabled, a pool member added to a pool on a Non Pool Host UDR is not treated as a pool member until the pool is created or Pool Profile updated on the Pool Host UDR when the connection between the two is active. Subscribers are only treated as a pool member for REST behavior, not for signaling.

Prerequisites

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

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

The subscriber must not 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*

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *subKeyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *subKeyValue*: Corresponding key field value assigned to *keyName*.

Request Content

None.

Response Content

None.

Table 75: Add Member to Pool Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	—	Subscriber successfully added to pool
400	MSR4100	Maximum number of Subscribers in a Basic Pool has been exceeded
404	MSR4001	Subscriber is not found
404	MSR4061	Specified pool does not exist
409	MSR4055	Subscriber is a member of a pool

Examples**Request 1**

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

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 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 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 basic pool. Both the pool and the subscriber exist. The subscriber is not a member of a pool. The basic pool has the maximum number of members allowed.

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="MSR4100">errorText</error>
```

6.7.2 Remove Member from Pool**Description**

This operation removes a Subscriber from a Pool.

Prerequisites

A pool with the key of 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
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 9999999999999999999999
- *subKeyName*: A key field in the Subscriber Profile.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *subKeyValue*: Corresponding key field value assigned to *keyName*.

Request Content

None.

Response Content

None.

Table 76: Remove Member from Pool 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 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 is 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>
```

6.7.3 Get Pool Members**Description**

This operation gets the list of Subscriber members of a Pool by *poolId*. This operation only gets the list of subscribers and addresses for a local pool on the UDR instance where the request was received, regardless if the pool is a PSO pool or not.

Prerequisites

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

Request URL

```
GET {BaseURI}/msr/pool/poolId/member
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 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 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>

```

- *keyNameX*: A key field for the member subscriber.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValueX*: Corresponding key field value assigned to *keyNameX*.

Table 77: Get Pool Members 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 3 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>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  
```



```

</member>
<member>
  <id><name>MSISDN</name><value>380561234777</value></id>
  <id><name>IMSI</name><value>311480100000999</value></id>
</member>
<member>
  <id><name>NAI</name><value>joe@wireless.com</value></id>
  <id><name>NAI</name><value>p12321@mynet.com</value></id>
</member>
</members>

```

Request 2

A request is made to get the list of subscribers for a pool. The pool exists but does not have any member subscribers.

Request URL

```
GET {BaseURI}/msr/pool/200000/member
```

Request Content

None.

Response 2

The request is successful and member subscribers are not 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>
```

6.7.4 Get Pool ID

Description

This operation gets the Pool ID related to a subscriber, based on the given user identity of the subscriber.

Prerequisites

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

The subscriber must be a member of a pool.

Request URL

```
GET {BaseURI}/msr/sub/keyName/keyValue/pool
```

- *keyName*: A key field for the member subscriber.
Value is either IMSI, MSISDN, NAI, or AccountId.
- *keyValue*: Corresponding key field value assigned to *keyName*.

Request Content

None.

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="Pool ID">poolId</field>
</pool>
```

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

Table 78: Get Pool ID Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Subscriber pool membership successfully returned
404	MSR4001	Subscriber is not found
404	MSR4062	Subscriber is not a member of a pool

Examples

Request 1

A request is made to get the Pool ID 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 Pool ID value was returned.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="Pool ID">100000</field>
</pool>
```

Request 2

A request is made to get the Pool ID 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>
```

6.7.5 Get All Pool Members**Description**

This operation has the option to get the list of Subscriber members of a Pool by *poolId*, from the local or from the local and all remote UDR systems. The Pool Spanning UDRs feature needs to be enabled in order to retrieve remote pool members. This command can be used on a single local Pool Host UDR as well, when the Pool Spanning UDRs feature is not enabled.

Prerequisites

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

Request URL

```
GET {BaseURI}/msr/pool/poolId/allmembers/AddressList/addressList/PSO/pso
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 99999999999999999999
- *addressList*: The requested address type of the subscriber for a pool is returned in the response
Values:
 - IMSI
Only pool members with an IMSI address type is returned in the response
 - All
All pool members are returned in the response (default).
- *pso*: Indicates whether the response includes pool members across all UDR instances in the pool network or only the UDR instance where the request is received.
Values.
 - yes
Response includes pool members across all UDR instances in the pool network.
 - No
Response only includes pool members on the local UDR instance (default).

NOTES:

- If a timeout occurs while waiting for a response from a remote UDR instance, then UDR fails the request with a REQUEST_TIMEOUT error.
- If a request fails to be sent to a remote UDR instance, then UDR fails the provisioning request with a REQUEST_TIMEOUT.
- This request can be used even when the Pool Spanning UDRs feature is not enabled.

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 requested user identities for that subscriber, one user identity per `<id>` element. There can be one or many `<id>` elements per `<member>` element.

The response includes pool members across all UDR instances in the pool network if the `pso` parameter was set to `yes` in the request and the PSO feature is enabled. Otherwise, only pool members on the local UDR instance are returned.

The response includes only the address types specified in the `addresslist` parameter in the request. (that is, those where `keyNameX` is in the list). Only the requested address types for a subscriber is returned in the response. If a subscriber is a pool member but does not have any of the requested address types, then the pool member is not included in the response.

If the PSO feature is enabled, then each pool member returned also includes the `udrId` corresponding to the UDR instance where they reside in the pool network.

```

<members>
[
  <member>
[
  <udrId>udrId</udrId> ]
  <id><name>keyName1</name><value>keyValue1</value></id>
[
  <id><name>keyName2</name><value>keyValue2</value></id>
  :
  <id><name>keyNameN</name><value>keyValueN</value></id>
]
</member>
]
[
  <member>
[
  <udrId>udrId</udrId> ]
  <id><name>keyName1</name><value>keyValue1</value></id>
[
  <id><name>keyName2</name><value>keyValue2</value></id>
  :
  <id><name>keyNameN</name><value>keyValueN</value></id>
]
</member>
:
  <member>
[
  <udrId>udrId</udrId> ]
  <id><name>keyName1</name><value>keyValue1</value></id>
[
  <id><name>keyName2</name><value>keyValue2</value></id>
  :
  <id><name>keyNameN</name><value>keyValueN</value></id>
]
</member>
]
</members>

```

- **udrId** (optional): A subscriber key range that is hosted by each UDR in the remote pool network. The response only includes the *udrId* if the Pools Spanning UDRs feature is enabled.
Value: 1 to 10 digits.
- **keyNameX**: A key field for the member subscriber.
Value is either IMSI, MSISDN, NAI, or AccountId.
- **keyValueX**: Corresponding key field value assigned to *keyNameX*.

Table 79 Get All Pool Members Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	—	Pool exists, and membership returned OK
404	MSR4061	Specified pool does not exist
404	MSR4102	Provisioning Request Timeout, a response was not received from the remote UDR

Examples**Request 1**

A request is made to get the list of subscriber members for a pool. The PSO feature is not enabled.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/no
```

Request Content

None.

Response 1

The request is successful, and the 3 subscriber members are returned. Only pool members from the local UDR instance 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>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  </member>
  <member>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
    <id><name>NAI</name><value>joe@wireless.com</value></id>
    <id><name>NAI</name><value>p12321@my.net.com</value></id>
  </member>
</members>
```

Request 2

A request is made to get the list of subscriber members for a pool. The pool exists, but does not have any subscriber members. The PSO feature is enabled.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/no
```

Request Content

None.

Response 2

The request is successful and subscriber members are not 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 subscriber members for a pool. The pool does not exist (local or any remote UDR instances). The PSO feature is enabled and the *pso* parameter is set to *yes*.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None.

Response 3

The request fails. The error value indicates that the pool does not exist on all queried UDR instances (local or remote).

HTTP Status Code

404

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4061">errorText</error>
```

Request 4

A request is made to get the list of subscriber members for a pool across all UDR instances in the pool network. The PSO feature is enabled and the *pso* parameter is set to *yes*.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None.

Response 4

The request is successful, and the 3 member subscribers are returned. The response includes pool members across all UDR instances in the pool network.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <udrId>1</udrId>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  </member>
  <member>
    <udrId>1</udrId>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
    <udrId>2</udrId>
    <id><name>NAI</name><value>joe@wireless.com</value></id>
    <id><name>NAI</name><value>p12321@mynet.com</value></id>
  </member>
</members>
```

Request 5

A request is made to get the list of pool members across all UDR instances in the pool network. The *psd* parameter is set to *yes*. The PSO feature is enabled. There is a connection issue between the local and remote UDR systems.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None.

Response 5

The request fails. The error value indicates a provisioning request timeout, the request could not be sent to the remote UDR due to the connection being down.

HTTP Status Code

404

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4102">errorText</error>
```

Request 6

A request is made to get the list of subscribers for a pool on all UDRs. The PSO feature is enabled. For this case, a response is not received from the remote UDR.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```


Request Content

None.

Response 6

The request fails. The error value indicates a provisioning request timeout; a response was not received from the remote UDR.

HTTP Status Code

404

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4102">errorText</error>
```

Request 7

A request is made to get the list of subscriber members for a pool when the PSO feature is disabled and the *pso* parameter is set to *yes*.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None.

Response 7

The request is successful, and the 3 member subscribers are returned. The response includes local pool members only.

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>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  </member>
  <member>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
    <id><name>NAI</name><value>joe@wireless.com</value></id>
    <id><name>NAI</name><value>p12321@mynet.com</value></id>
  </member>
</members>
```

Request 8

A request is made to get the list of subscriber members for a pool. The PSO feature is enabled. The *psd* parameter is set to *yes* and *addressList* is set to *IMSI*.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/IMSI/PSO/yes
```

Request Content

None.

Response 8

The request is successful, and the 2 member subscribers are returned. The response includes only IMSI keys for all pool members.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <udrId>1</udrId>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>IMSI</name><value>311480100532432</value></id>
  </member>
  <member>
    <udrId>2</udrId>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
</members>
```

Request 9

A request is made to get the list of subscribers for a pool on all UDRs. The PSO feature is enabled and activated on 4 UDRs. For this case, a response is not received from one remote UDR.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None.

Response 9

The request fails. The error value indicates a provisioning request timeout; a response was not received from a remote UDR.

HTTP Status Code

404

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4102">errorText</error>
```

6.8 Pool Special Operation Commands

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

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

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

Table 80: Summary of Pool Special Operation Commands

Command	Description	Keys	Command Syntax
Reset Pool Quota	Reset the fields in the specified Pool Quota	Pool ID and Row Identifier	POST {BaseURI}/msr/pool/keyName/data/transparentDataType/rowIdValue
	Reset the fields in the specified Pool Quota and instance identifier	or Pool ID, Row Identifier and Instance Identifier	POST {BaseURI}/msr/pool/keyName/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue

6.8.1 Reset Pool Quota**Description**

This operation resets a particular quota row in the PoolQuota transparent data associated with a pool.

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

If the pool has PoolQuota transparent data, then the configured values in the specified quota row are reset to the configured reset values.

NOTES:

- The *rowIdValue* is case-sensitive. If a row called DayPass exists, then an attempt to reset a quota row called DayPass is successful, but an attempt to reset a quota row called DAYPASS fails.
- When a PoolQuota instance is reset using the **Pool Reset Quota** command, each resettable field is set to its defined reset value. If the field does not currently exist, it is not created. But, if a resettable field does not exist, and the field has a default value, then the field is created with the default value.
- If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

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

The PoolQuota transparent data must exist for the pool.

The specified quota row must exist in the PoolQuota transparent data.

Request URL**Without Instance Identifier**

```
POST {BaseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue
```

With Instance Identifier

```
POST {BaseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- *poolId*: Pool ID value of the pool. Numeric value, 1 to 22 digits in length.
Values: 1 to 999999999999999999999999999999
- *transparentDataType*: A user defined type/name for the transparent data.
Value is *poolquota* for the PoolQuota transparent data.
- *rowIdValue*: The row name value that identifies the row in the transparent data blob.
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance.
 - Value is *cid* for the PoolQuota transparent data.
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*.

Request Content

None.

Response Content

None.

Table 81: Reset Pool Quota 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	Pool ID 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 *PQ1* PoolQuota row for a pool. The pool has PoolQuota transparent data, and the PoolQuota transparent data contains a PoolQuota row called *PQ1*.

Request URL

```
POST {baseURI}/msr/pool/10000/data/poolquota/PQ1
```

Request Content

None.

Response 1

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

HTTP Status Code

204

Response Content

None.

Request 2

A request is made to reset the *PQ1* PoolQuota row for a pool. The pool does not have PoolQuota transparent data.

Request URL

```
POST {baseURI}/msr/pool/10000/data/poolquota/PQ1
```

Request Content

None.

Response 2

The request fails because the pool does not have PoolQuota transparent data.

HTTP Status Code

404

Response Content

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

Request 3

A request is made to reset the *PQ6* PoolQuota row for a pool. The pool has PoolQuota transparent data, but the PoolQuota transparent data does not contain a PoolQuota row called *PQ6*.

Request URL

```
POST {baseURI}/msr/pool/10000/data/poolquota/PQ6
```

Request Content

None.

Response 3

The request fails, because the *PQ6* 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* PoolQuota row for a pool. The pool has PoolQuota transparent data, and the PoolQuota transparent data contains two rows called *Weekday*.

Request URL

```
POST {baseURI}/msr/pool/10000/data/poolquota/Weekday
```

Request Content

None.

Response 4

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 5

A request is made to reset the *PQ1* PoolQuota row for a pool having *cid* of value 45678. The pool has PoolQuota transparent data, and the PoolQuota transparent data contains a PoolQuota row called *PQ1* having *cid* of value 45678.

Request URL

```
POST {baseURI}/msr/pool/10000/data/poolquota/PQ1/row/cid/45678
```

Request Content

None.

Response 5

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

HTTP Status Code

204

Response Content

None.

APPENDIX A. REST INTERFACE SYSTEM VARIABLES

The REST interface has a set of system variables that affect its operation as it runs. REST interface variables (Table 28) can be set via the UDR GUI and can be changed at runtime to effect dynamic server reconfiguration.

Table 82: REST Interface System variables

Parameter	Description
REST Interface Port	REST Interface TCP Listening Port. NOTE: Changes to the TCP listening port do not take effect until the udrprov process is restarted. Also, you must specify a different port than the SOAP interface. Default is 8787; range is 0 to 65535.
REST Interface Idle Timeout	The maximum time (in seconds) that an open REST connection remains active without a request being sent, before the connection is dropped. Default is 1200; range is 1 to 86400.
Maximum REST Connections	Maximum number of simultaneous REST Interface client connections. Default is 100; range is 1 to 100.
Allow REST Connections	Whether or not to allow incoming provisioning connections on the REST Interface. Default is UNCHECKED.
REST Secure Mode	Whether the REST Interface operates in secure mode (using TLS), or unsecure mode (plain text). NOTE: Changes to the Secure Mode do not take effect until the udrprov process is restarted. Default is 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 is resent to ensure that the request was committed. Default is 5; range is 2 to 3600.
Compatibility Mode*	Indicates whether backwards compatibility is enabled. NOTE: Change to Compatibility Mode may cause the existing provisioning connections to be dropped. Default is R10.0+

APPENDIX B. LEGACY SPR COMPATIBILITY MODE

UDR can be configured to run in a compatibility mode with the legacy SPR.

When the Compatibility Mode system option (see Appendix A), which is configurable by the UDR GUI, is set to R10.0+ then UDR behaves as described in the main body of this document. When Compatibility Mode is set to R9.x, then the differences contained in this appendix apply.

Enabling this configuration option results in the format of some requests and responses being different to the default UDR behavior. This appendix lists the different requests and responses that enabling the option applies to.

B.1 Get Row Response Format

UDR returns a data row in the format (Field Based or Element Based) how it is defined or stored. 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, in 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>3</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>
```

NOTE: 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>
```

APPENDIX C. MY ORACLE SUPPORT

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

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

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

You are connected to a live agent who can assist you with My Oracle Support registration and opening a support ticket.

My Oracle Support is available 24 hours a day, 7 days a week, 365 days a year.

APPENDIX D. LOCATE PRODUCT DOCUMENTATION ON THE ORACLE HELP CENTER SITE

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

1. Access the Oracle Help Center site at <http://docs.oracle.com>
2. Click **Industries**.
3. Under the Oracle Communications subheading, click the **Oracle Communications documentation** link.
4. The Communications Documentation page opens. Most products covered by these documentation sets are under the headings Network Session Delivery and Control Infrastructure or Platforms.
5. Click on your product and then the release number.

A list of the documentation set for the selected product and release displays.

6. To download a file to your location, right-click the **PDF** link, select **Save target as** (or similar command based on your browser), and save to a local folder.