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 A.
# Table of Contents

1 **INTRODUCTION** ................................................................................................................ 10

1.1 Purpose and Scope ....................................................................................................... 10
1.2 External 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 ................................................................................................. 17

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 ......................................................................................................... 18
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 Rest Conventions ........................................................................................................ 20

2.12.1 HTTP(S) Request Headers ............................................................................................... 20
2.12.1.1 HTTP version .................................................................................................... 20
2.12.1.2 Accept Header .................................................................................................. 20
2.12.1.3 Transfer-Encoding Header ......................................................................................... 20
2.12.1.4 Requests with body content ................................................................. 20
2.12.2 HTTP(S) Status Codes and Error Messages ............................................. 21

3 REST INTERFACE MESSAGE DEFINITIONS ............................................... 24
3.1 Message Conventions ..................................................................................... 24
  3.1.1 HTTP Method .............................................................................................. 24
  3.1.2 Base URI ..................................................................................................... 24
  3.1.3 REST URL .................................................................................................. 24
    3.1.3.1 Subscriber or Pool in URL ................................................................. 24
  3.1.3.2 Opaque Data Operations in URL ............................................................ 24
  3.1.3.3 Field in URL ............................................................................................ 25
  3.1.3.4 Transparent Data Row Operations in URL ............................................. 25
  3.1.3.5 Transparent Data Row Field Operations in URL .................................... 25
  3.1.4 URL Character Encoding ............................................................................. 25
3.2 Case Sensitivity ............................................................................................... 25
3.3 XML Comments in a Request ......................................................................... 26
3.4 List of Messages .............................................................................................. 26

4 UDR DATA MODEL ............................................................................................. 30
4.1 Subscriber Data ............................................................................................... 32
  4.1.1 Subscriber Profile ....................................................................................... 32
  4.1.2 Quota ......................................................................................................... 34
  4.1.3 State ........................................................................................................... 35
  4.1.4 Dynamic Quota ........................................................................................... 36
4.2 Pool Data .......................................................................................................... 37
  4.2.1 Pool Profile ................................................................................................ 37
  4.2.2 Pool Quota .................................................................................................. 38
  4.2.3 Pool State .................................................................................................... 38
  4.2.4 Pool Dynamic Quota ................................................................................... 39
4.3 Date/Timestamp Format .................................................................................. 39

5 SUBSCRIBER PROVISIONING .......................................................................... 40
5.1 Subscriber Profile Commands ....................................................................... 40
  5.1.1 Create Subscriber ...................................................................................... 40
  5.1.2 Get Profile .................................................................................................. 45
  5.1.3 Update Profile ............................................................................................ 47
  5.1.4 Delete Profile ............................................................................................ 49
5.2 Subscriber Profile Field Commands ................................................................ 50
  5.2.1 Add Field Value ........................................................................................ 51
6.3.3 Delete Opaque Data ........................................................................................................ 126

6.4 Pool Data Row Commands ............................................................................................ 128
   6.4.1 Set Row .................................................................................................................. 128
   6.4.2 Get Row .................................................................................................................... 131
   6.4.3 Delete Row .............................................................................................................. 134

6.5 Pool Data Row Field Commands ............................................................................... 136
   6.5.1 Get Row Field ......................................................................................................... 136
   6.5.2 Get Row Field Value .............................................................................................. 139
   6.5.3 Update Row Field ................................................................................................... 142
   6.5.4 Delete Row Field .................................................................................................... 144

6.6 Additional Pool Commands ......................................................................................... 146
   6.6.1 Add Member to Pool ............................................................................................... 147
   6.6.2 Remove Member from Pool ..................................................................................... 150
   6.6.3 Get Pool Members ................................................................................................. 151
   6.6.4 Get PoolID ............................................................................................................. 154
List of Figures

Figure 1: User Data Repository High Level Architecture ......................................................................................... 13
Figure 2: Data Model............................................................................................................................................... 32
### 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 ............................................................................................. 16
Table 4: HTTP(S) Status Codes ........................................................................................................... 21
Table 5: Error Codes ............................................................................................................................ 21
Table 6: Summary of Subscriber Commands ....................................................................................... 27
Table 7: Summary of Pool Commands .................................................................................................. 28
Table 8: Subscriber Profile Entity Definition ..................................................................................... 33
Table 9: Quota Entity Definition .......................................................................................................... 34
Table 10: State Entity Definition ........................................................................................................ 35
Table 11: Dynamic Quota Entity Definition ......................................................................................... 36
Table 12: Pool Profile Entity Definition .............................................................................................. 37
Table 13: Summary of Subscriber Profile Commands .......................................................................... 40
Table 14: Create Subscriber Response Status/Error Codes ................................................................. 41
Table 15: Get Profile Response Status/Error Codes ............................................................................ 46
Table 16: Update Profile Response Status/Error Codes ........................................................................ 48
Table 17: Delete Profile Response Status/Error Codes ......................................................................... 49
Table 18: Summary of Subscriber Profile Field Commands ................................................................. 50
Table 19: Add Field Value Response Status/Error Codes .................................................................... 52
Table 20: Get Field Response Status/Error Codes ............................................................................. 54
Table 21: Get Field Value Response Status/Error Codes ..................................................................... 57
Table 22: Update Field Response Status/Error Codes ......................................................................... 59
Table 23: Update Multiple Fields Response Status/Error Codes ......................................................... 62
Table 24: Delete Field Response Status/Error Codes ......................................................................... 64
Table 25: Delete Field Value Response Status/Error Codes ............................................................... 66
Table 26: Summary of Subscriber Opaque Data Commands ............................................................... 68
Table 27: Set Opaque Data Response Status/Error Codes .................................................................. 69
Table 28: Get Opaque Data Response Status/Error Codes ................................................................. 72
Table 29: Delete Opaque Data Response Status/Error Codes ............................................................. 74
Table 30: Summary of Subscriber Data Row Commands .................................................................... 76
Table 31: Set Row Response Status/Error Codes ............................................................................... 77
Table 32: Get Row Response Status/Error Codes ............................................................................... 81
Table 33: Delete Row Response Status/Error Codes .......................................................................... 84
Table 34: Summary of Subscriber Data Row Field Commands .......................................................... 85
Table 35: Get Row Field Response Status/Error Codes ...................................................................... 87
Table 36: Get Row Field Value Response Status/Error Codes .......................................................... 89
Table 37: Update Row Field Response Status/Error Codes ............................................................... 93
Table 38: Delete Row Field Response Status/Error Codes ............................................................... 96
Table 39: Summary of Subscriber Special Operation Commands ..................................................... 97
Table 40: Reset Quota Response Status/Error Codes ....................................................................... 98
Table 41: Summary of Pool Profile Commands ................................................................................... 101
Table 42: Create Pool Response Status/Error Codes ......................................................................... 102
Table 43: Get Pool Response Status/Error Codes ............................................................................. 104
Table 44: Update Pool Response Status/Error Codes ......................................................................... 106
Table 45: Delete Pool Response Status/Error Codes .......................................................................... 107
Table 46: Summary of Pool Profile Field Commands .......................................................................... 108
Table 47: Add Field Value Profile Field Commands ............................................................................ 109
Table 48: Get Field Response Status/Error Codes ................................................................................................. 111
Table 49: Get Field Value Response Status/Error Codes ........................................................................................ 113
Table 50: Update Field Response Status/Error Codes ............................................................................................ 115
Table 51: Update Multiple Fields Response Status/Error Codes ........................................................................... 116
Table 52: Delete Field Response Status/Error Codes ............................................................................................ 118
Table 53: Delete Field Value Response Status/Error Codes .................................................................................. 119
Table 54: Summary of Pool Opaque Data Commands .......................................................................................... 121
Table 55: Set Opaque Data Response Status/Error Codes .................................................................................... 122
Table 56: Get Opaque Data Response Status/Error Codes ................................................................................... 124
Table 57: Delete Opaque Data Response Status/Error Codes .............................................................................. 127
Table 58: Summary of Pool Data Row Commands ................................................................................................ 128
Table 59: Set Row Response Status/Error Codes ................................................................................................. 129
Table 60: Get Row Response Status/Error Codes ................................................................................................. 132
Table 61: Delete Row Response Status/Error Codes ............................................................................................. 135
Table 62: Summary of Pool Data Row Field Commands ........................................................................................ 136
Table 63: Request URL Response Status/Error Codes .......................................................................................... 137
Table 64: Get Row Field Value Response Status/Error Codes ................................................................................ 140
Table 65: Update Row Field Response Status/Error Codes ................................................................................... 143
Table 66: Delete Row Field Response Status/Error Codes .................................................................................... 145
Table 67: Summary of Additional Pool Commands ............................................................................................... 146
Table 68: Add Member to Pool Table 69 Response Status/Error Codes ............................................................... 147
Table 70: Remove Member from Pool Response Status/Error Codes ................................................................. 150
Table 71: Get Pool Members Response Status/Error Codes .................................................................................. 152
Table 72: Get PoolID Response Status/Error Codes .............................................................................................. 155
Table 73: Bulk Import/Export variables ................................................................................................................. 156
1 INTRODUCTION

1.1 Purpose and Scope
This document presents the REST Provisioning interface to be used by local and remote provisioning client applications to administer the Provisioning Database of the Oracle Communications User Data Repository (UDR) system. Through REST interfaces, an external provisioning system supplied and maintained by the network operator can 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 External References
The following external document references capture the source material used to create this document.

[1] IMS Sh interface; Signalling flows and message contents, 3GPP TS 29.328, Release 11
[3] User Data Convergence (UDC); Technical realization and information flows; Stage 2, 3GPP TS 23.335, Release 11

1.3 Glossary
This section lists terms and acronyms specific to this document.

Table 1: Glossary

<table>
<thead>
<tr>
<th>Acronym/Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID</td>
<td>Atomic, Consistent, Isolatable, Durable</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary Large Object</td>
</tr>
<tr>
<td>CFG</td>
<td>Configuration Data—data for components and system identification and configuration</td>
</tr>
<tr>
<td>CPS</td>
<td>Customer Provisioning System</td>
</tr>
<tr>
<td>DP</td>
<td>Database Processor</td>
</tr>
<tr>
<td>FRS</td>
<td>Feature Requirements Specification</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IMSI</td>
<td>International Mobile Subscriber Identity, or IMSI [im-zee]</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>MEAL</td>
<td>Measurements, Events, Alarms, and Logs</td>
</tr>
<tr>
<td>Acronym/Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>MP</td>
<td>Message Processor</td>
</tr>
<tr>
<td>MSISDN</td>
<td>Mobile Subscriber ISDN Number</td>
</tr>
<tr>
<td>NA</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>NE</td>
<td>Network Element</td>
</tr>
<tr>
<td>NPA</td>
<td>Numbering Plan Area (Area Code)</td>
</tr>
<tr>
<td>OAMP</td>
<td>Operations, Administration, Maintenance, and Provisioning</td>
</tr>
<tr>
<td>NOAMP</td>
<td>Network OAMPProvisioning</td>
</tr>
<tr>
<td>PCRF</td>
<td>Policy Charging and Rules Function</td>
</tr>
<tr>
<td>PS</td>
<td>Provisioning System</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SDO</td>
<td>Subscriber Data Object</td>
</tr>
<tr>
<td>SEC</td>
<td>Subscriber Entity Configuration</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SOAM</td>
<td>System Operation, Administration, and Maintenance</td>
</tr>
<tr>
<td>SPR</td>
<td>Subscriber Profile Repository</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UTC</td>
<td>Coordinated Universal Time</td>
</tr>
<tr>
<td>VIP</td>
<td>Virtual IP</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
2 SYSTEM ARCHITECTURE

2.1 Overview

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

The data stored in UDR can be permanent and temporary data. Permanent data is subscription data and relates to the required information the system needs to know to perform the service. User identities (MSISDN, IMSI, NAI and 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 (transparent data stored by application servers for service execution, user status, usage, and so on).

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

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

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

Figure 1 shows that Oracle Communications User Data Repository consists of several functional blocks. The Message Processors (MP) provide support for a variety of protocols that entail the front-end signaling to peer network nodes. The back-end UDR database resides on the N-OAMP servers. This release focuses on the development of the Sh messaging interface for use with the UDR application.

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

The Network level OAMP server (NOAMP) 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 NOAMP and in turn replicates the data to the message processors.
MP functions as the client-side of the network application, provides the network connectivity and hosts network stack such as Diameter, SOAP, LDAP, SIP and SS7.

Figure 1: User Data Repository High Level Architecture

2.2 Provisioning Interface

The REST provisioning interface provides following data manipulation commands:

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

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

2.3 REST Application Server (RAS)
The application within the provisioning process interfacing to REST provisioning clients runs on every active
NOAMP 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 the Customer Provisioning Systems (CPS). These are independent information
systems supplied and maintained by the network operator to be used for provisioning the UDR system. Through
the RAS, 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 NOAMP using the VIP of the Primary NOAMP.

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

Provisioning clients must run a timeout for the response to a request, if a response is not sent. If a response is
not received, a client should drop the connection and re-establish a connection before trying again.

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

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

• Client server IP Address white list
• Secure Connections using 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 following 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 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 operating RAS mode configured (secure or unsecure), provisioning clients can connect using either 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).

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 contains 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/Tekelec 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 under /usr/TKLC/udr/ssl. These files are copied to the server running the provisioning client.
Table 2: TLS X.509 Certificate and Key PEM-encoded Files

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

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/signed by Oracle Certification Authority (CA), it cannot establish a secure connection with the RAS.

2.5.2.2 Supported TLS Cipher Suites

A cipher suite is a set/combo of lower-level algorithms that a TLS-enabled connection uses to do authentication, key exchange, and stream encryption. The following table 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 known security vulnerabilities.

Table 3: TLS Supported Cipher Suites

<table>
<thead>
<tr>
<th>Cipher Suite (RFC)</th>
<th>OpenSSL Equivalent</th>
<th>Key Exchange</th>
<th>Signing/Authentication</th>
<th>Encryption (Bits)</th>
<th>MAC (Hash) Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
<td>AES256-SHA</td>
<td>RSA</td>
<td>RSA</td>
<td>AES (256)</td>
<td>SHA-1</td>
</tr>
<tr>
<td>TLS_RSA_WITH_3DES_EDE_CBC_SHA</td>
<td>DES-CBC3-SHA</td>
<td>RSA</td>
<td>RSA</td>
<td>3DES(168)</td>
<td>SHA-1</td>
</tr>
<tr>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
<td>AES128-SHA</td>
<td>RSA</td>
<td>RSA</td>
<td>AES(128)</td>
<td>SHA-1</td>
</tr>
<tr>
<td>Cipher Suite (RFC)</td>
<td>OpenSSL Equivalent</td>
<td>Key Exchange</td>
<td>Signing/Authentication</td>
<td>Encryption (Bits)</td>
<td>MAC (Hash) Algorithms</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>--------------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>-------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>TLS_KRB5_WITH_RC4_128_SHA</td>
<td>KRB5-RC4-SHA</td>
<td>KRB5</td>
<td>KRB5</td>
<td>RC4(128)</td>
<td>SHA-1</td>
</tr>
<tr>
<td>TLS_RSA_WITH RC4_128_SHA</td>
<td>RC4-SHA</td>
<td>RSA</td>
<td>RSA</td>
<td>RC4(128)</td>
<td>SHA-1</td>
</tr>
<tr>
<td>TLS_KRB5_WITH_3DES_EDE_CBC_SHA</td>
<td>KRB5-DES-CBC3-SHA</td>
<td>KRB5</td>
<td>KRB5</td>
<td>3DES(168)</td>
<td>SHA-1</td>
</tr>
</tbody>
</table>

### 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 should send a Connection header including the connection-token close. If either the client or the server sends the close token in the Connection header, that request becomes the last one for the connection.

The provisioning client establishes a 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.

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 6.6.4 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.

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

### 2.8 Database Transactions

Each create, update, or delete request coming from REST interface triggers a unique database transaction, such as 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 within that transaction which were performed successfully. If any database errors are encountered while committing the transaction, all updates are rolled back and the database is restored to its previous state.

2.8.1.2  Consistency

Data across all requests performed inside a transaction is consistent.

2.8.1.3  Isolation

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

2.8.1.4  Durability

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

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

Data durability signifies the replication of the provisioned data to different parts of the system before a response is provided for a provisioning transaction. The following 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/limit the REST provisioning interface in a number of different ways.

2.9.1  Connections Allowed

The configuration variable {allow resto provisioning connections} (see 6.6.4 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. Alarm 13000) is raised. Any attempts to connect are rejected.

When {allow resto provisioning connections} is set back to ALLOWED, the alarm is cleared, and connections are accepted again.
2.9.2 Disable Provisioning

When the Oracle Communications User Data Repository GUI option to disable provisioning is selected, existing connections remain up, and new connections are allowed. But, any provisioning request 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 5.1.1.

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 (such as no requests 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.

The HTTP Status Code 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 some requests (based on the HTTP method, see section 3.1.1).

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

The HTTP Status Code 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>

REST Interface Description

Oracle Communications User Data Repository 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/HTTPS interface that is based on RESTful concepts. This section defines the operations that can be performed using the REST interface.

2.12 Rest Conventions

The REST interface uses the following RESTful concepts:

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

2.12.1 HTTP(S) Request Headers

The following HTTP(S) requirements must be followed.

2.12.1.1 HTTP version

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

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

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

```plaintext
Request Version : HTTPS TLS v1
```

2.12.1.2 Accept Header

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

```plaintext
Accept: application/camiant-msr-v1+xml      <- version number is 1 or 2.0
```

or

```plaintext
Accept: application/camiant-msr-v2.0+xml
```

or

```plaintext
Accept :/*/*
```

or

```plaintext
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 Oracle Communications User Data Repository support both versions 1 and 2.0.

The Oracle Communications User Data Repository response to an incorrect MIME version is a Bad Request, for example, with error code `Invalid Accept: application/camiant-msr-v1+xml`.

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

2.12.1.3 Transfer-Encoding Header

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

```plaintext
Transfer-Encoding: chunked
```

2.12.1.4 Requests with body content

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

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

## 2.12.2 HTTP(S) Status Codes and Error Messages

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

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200—OK</td>
<td>Indicates the successful completion of request processing.</td>
</tr>
<tr>
<td>201—Created</td>
<td>Used for new entities.</td>
</tr>
<tr>
<td>204—No Content</td>
<td>The request completed successfully and no response content body is sent back to the client.</td>
</tr>
<tr>
<td>400—Bad Request</td>
<td>This indicates that there is a problem with how the request is formatted or that the data in the request caused a validation error.</td>
</tr>
<tr>
<td>404—Not Found</td>
<td>Indicates that the client tried to operate on a resource that did not exist.</td>
</tr>
<tr>
<td>409—Conflict</td>
<td>Indicated that the client tried to operate on a resource where the operation was not appropriate for that resource.</td>
</tr>
<tr>
<td>4xx—Other</td>
<td>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.</td>
</tr>
<tr>
<td>500—Internal Server Error</td>
<td>This error and other errors in the 5xx range indicate server problems.</td>
</tr>
<tr>
<td>503—Service Unavailable</td>
<td>Indicates that the client tried to send a provisioning request when provisioning was disabled.</td>
</tr>
<tr>
<td>507—Insufficient Storage</td>
<td>Indicates that free system memory is low, and the database cannot store any new data.</td>
</tr>
</tbody>
</table>

Besides the HTTP status codes, following additional error codes are provided for the 4xx and 5xx range of Status Codes. Note that the “Description” column is for reference only, it is not included in the HTTP response. Additional text may be included in the HTTP response in some cases, for some responses.

### Table 5: Error Codes

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSR4000</td>
<td>Invalid content request data supplied</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MSR4001</td>
<td>Subscriber/pool not found</td>
</tr>
<tr>
<td>MSR4002</td>
<td>Subscriber/pool/data field is not defined</td>
</tr>
<tr>
<td>MSR4003</td>
<td>A key is detected to be already in the system for another subscriber/pool</td>
</tr>
<tr>
<td>MSR4004</td>
<td>Unique key not found for subscriber/pool</td>
</tr>
<tr>
<td>MSR4005</td>
<td>Field does not support multiple values and value for field exists</td>
</tr>
<tr>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>MSR4050</td>
<td>Unknown key, the key provided in the request is invalid</td>
</tr>
<tr>
<td>MSR4051</td>
<td>The value provided for the field is invalid</td>
</tr>
<tr>
<td>MSR4053</td>
<td>Subscriber/pool exist, but the field value is incorrect</td>
</tr>
<tr>
<td>MSR4055</td>
<td>Subscriber is a member of a pool</td>
</tr>
<tr>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>MSR4057</td>
<td>Request only contains one field to update</td>
</tr>
<tr>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
<tr>
<td>MSR4060</td>
<td>Number of pool members exceeded</td>
</tr>
<tr>
<td>MSR4061</td>
<td>Specified pool does not exist</td>
</tr>
<tr>
<td>MSR4062</td>
<td>Subscriber is not a member of the pool</td>
</tr>
<tr>
<td>MSR4063</td>
<td>Entity cannot be reset</td>
</tr>
<tr>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>MSR4065</td>
<td>Field is not set</td>
</tr>
<tr>
<td>MSR4066</td>
<td>Field value exists</td>
</tr>
<tr>
<td>MSR4067</td>
<td>Multiple matching rows found</td>
</tr>
<tr>
<td>MSR4068</td>
<td>Free system memory is low</td>
</tr>
<tr>
<td>MSR4069</td>
<td>At least one key is required</td>
</tr>
<tr>
<td>MSR4097</td>
<td>Request rejected due to system congestion</td>
</tr>
<tr>
<td>MSR4098</td>
<td>Provisioning is disabled</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MSR4099</td>
<td>Unexpected server error has occurred</td>
</tr>
<tr>
<td>MSR4103</td>
<td>A key is detected to be already in the system for an AE subscriber</td>
</tr>
</tbody>
</table>

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

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

In the examples, the error text associated with the MSRxxxx is not included because the 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 the following syntax:

\[ \text{http(s)}://\{\text{DNS Name or IP address}\}:<\text{IP Port}>/rs \]

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 Oracle Communications User Data Repository 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 following XML conventions 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.

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 it 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 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

3.1.4 URL Character Encoding

It is allowed to encode restricted characters in the URL using the % (percent) character, such as %3B for a ; (semicolon), but 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 (such as, msr, sub, pool, field, data, or multipleFields) are case-sensitive. Exact case must be followed for all the commands described in this document, or the request fails.

For example, the following is valid:

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

But the following is not:

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

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 would fail.

• When an attribute in the XML blob contains the row identifier name—aka `rowIdName` (for example for Quota, the element `<quota name="AggregateLimit">` contains the attribute called “name”) the row identifier name is not case-sensitive.

• When a `rowIdValue` is used to find a row (such as when using the Get Row command), the row identifier value is case-sensitive. If an entity contained a row called DayPass, and the Get Row command is used to get the row DAYPASS, the command 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 within the request or the content body, such as:

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

If a request contains a comment, the request is rejected with the following error:

**HTTP Status Code**

400

**Response Content**

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

3.4 List of Messages

The following table provides a list of operations/messages for subscriber data. Each row of the table represents a command. Parameters required for each command are in the colored column. Any blank/uncolored column represents unused parameter for corresponding command.
### Table 6: Summary of Subscriber Commands

<table>
<thead>
<tr>
<th>Operation Data</th>
<th>Command (Method)</th>
<th>URL</th>
<th>Main Object</th>
<th>Key Name</th>
<th>Key Value</th>
<th>subObject Type</th>
<th>subObject Name</th>
<th>subObject Value</th>
<th>Field Name</th>
<th>Field Value</th>
<th>Additional Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscriber Profile</td>
<td>Create Profile (POST)</td>
<td>{Base URL}/msr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Get Profile (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Update Profile (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Delete Profile (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td>Subscriber Field</td>
<td>Add Field Value (POST)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>field/multipleFields</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Get Field (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Get Field Value (GET)</td>
<td>{Base URL}/msr</td>
<td>sub</td>
<td>{keyName}</td>
<td>{keyValue}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Update Field (PUT)</td>
<td></td>
<td></td>
<td>MSISDN, NAI, IMSI, AccountId</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Delete Field (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Delete Field Value (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td>Subscriber Opaque Data</td>
<td>Set Opaque Data (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>data</td>
<td></td>
<td>{opaqueDataType}</td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Get Opaque Data (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Delete Opaque Data (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td>Subscriber Data Row</td>
<td>Set Row (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Get Row (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
</tbody>
</table>
### Table 7: Summary of Pool Commands

<table>
<thead>
<tr>
<th>Operation Data</th>
<th>Command (Method)</th>
<th>URL</th>
<th>Main Object</th>
<th>Key Name</th>
<th>Key Value</th>
<th>subObject Type</th>
<th>subObject Name</th>
<th>subObject Value</th>
<th>Field Name</th>
<th>Field Value</th>
<th>Additional Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pool Profile</td>
<td>Create Pool (POST)</td>
<td>{Base URL} /msr</td>
<td>pool</td>
<td>PoolID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Get Pool (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Request Content</td>
</tr>
<tr>
<td></td>
<td>Update Pool (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delete Pool (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Profile</td>
<td>Add Field Value (POST)</td>
<td>field/multipleFields</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Get Field (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following table provides a list of operations/messages for pool data. Similar to the previous table, each row of the table represents a command. Parameters required for each command are in the colored column. Any blank (uncolored) column represents unused parameter for corresponding command.
<table>
<thead>
<tr>
<th>Operation Data</th>
<th>Command (Method)</th>
<th>URL</th>
<th>Main Object</th>
<th>Key Name</th>
<th>Key Value</th>
<th>subObject Type</th>
<th>subObject Name</th>
<th>subObject Value</th>
<th>Field Name</th>
<th>Field Value</th>
<th>Additional Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pool Opaque Data</td>
<td>Get Field Value (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Update Field (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delete Field (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Opaque Data</td>
<td>Delete Field Value (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row</td>
<td>Set Opaque Data (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Get Opaque Data (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row</td>
<td>Delete Opaque Data (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Set Row (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Get Row (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Delete Row (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Get Row Field (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Get Row Field Value (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Update Row Field (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Delete Row Field (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Delete Row Field Value (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Provisioning**

Request Content

<table>
<thead>
<tr>
<th>Operation Data</th>
<th>Command (Method)</th>
<th>URL</th>
<th>Main Object</th>
<th>Key Name</th>
<th>Key Value</th>
<th>subObject Type</th>
<th>subObject Name</th>
<th>subObject Value</th>
<th>Field Name</th>
<th>Field Value</th>
<th>Additional Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pool Opaque Data</td>
<td>Get Field Value (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Update Field (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delete Field (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Opaque Data</td>
<td>Delete Field Value (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row</td>
<td>Set Opaque Data (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Get Opaque Data (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row</td>
<td>Delete Opaque Data (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Set Row (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Get Row (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Delete Row (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Get Row Field (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Get Row Field Value (GET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Update Row Field (PUT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Delete Row Field (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool Data Row Field</td>
<td>Delete Row Field Value (DELETE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Provisioning**

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

- **Quota**
  Information that represents the use of managed resources (quota, pass, top-up, and roll-over) by 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, roll-over). This data may be created or updated by either the provisioning interface or the Sh interface.

- **Pool Membership**
  The pool to which the subscriber is associated. The current implementation allows a subscriber to be associated with a single pool, although the intention is to extend this to multiple pools in the future.

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

The pool-related data includes:

- **Pool Profile**
  Pre-provisioned information that describes a pool

- **Pool Quota**
  Information that represents managed resources used by the pool (quota, pass, top-up, and roll-over)

- **Pool State**
  Pool-specific properties

- **Pool Dynamic Quota**
  Dynamically configured information related to managed resources (pass, top-up, and roll-over)

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

The Index contains information on the following:

- **Subscription**
  - A subscription exists for every individual subscriber
  - Maps a subscription to the user identities through which it can be accessed
  - Maps an individual subscription to the pool of which they are a member
- **Pool Subscription**
  - A pool subscription exists for every pool
  - Maps a pool subscription to the pool identity through which it can be accessed
  - Maps a pool subscription to the individual subscriptions of the subscribers that are members of the pool
- **User Identities**
  - Use to map a specific user identity to a subscription
  - IMSI, MSISDN, NAI and AccountId map to an individual subscription
- **PoolID**
  - Maps a pool to a pool
  - Maps a pool to the list of the individual subscriber members

The Subscription Data Object (SDO):

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

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

A field within an entity can be defined as mandatory, or optional. A mandatory field must exist, and cannot be deleted.

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

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

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.
Provisioning

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

UDR 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

<table>
<thead>
<tr>
<th>Name (XML tag)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriber</td>
<td>—</td>
<td>Sequence (multiplicity = 1)</td>
</tr>
<tr>
<td>MSISDN</td>
<td>String</td>
<td>List of MSISDNs (8 to 15 numeric digits). A separate entry is included for each MSISDN associated with the profile for the subscriber.</td>
</tr>
<tr>
<td>IMSI</td>
<td>String</td>
<td>List of IMSIs (10 to 15 numeric digits). A separate entry is included for each IMSI associated with the profile for the subscriber.</td>
</tr>
<tr>
<td>NAI</td>
<td>String</td>
<td>List of NAI(s) in format “user@domain”). A separate entry is included for each NAI associated with the profile for the subscriber.</td>
</tr>
<tr>
<td>AccountId</td>
<td>String</td>
<td>Any string that can be used to identify the account for the subscriber (1 to 255 characters).</td>
</tr>
<tr>
<td>BillingDay</td>
<td>String</td>
<td>Allowed values are (0 to 31). The day of the month (1 to 31) when the associated quota for the subscriber is reset. 0 indicates that the default value configured at the PCRF level is used. This is automatically set in any record where BillingDay is not specified.</td>
</tr>
<tr>
<td>Entitlement</td>
<td>String</td>
<td>List of entitlements. A separate entry is included for each entitlement associated with the profile for the subscriber.</td>
</tr>
<tr>
<td>Tier</td>
<td>String</td>
<td>Tier for the subscriber.</td>
</tr>
<tr>
<td>Custom1</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom2</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom3</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom4</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom5</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom6</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom7</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom8</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom9</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom10</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom11</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
</tbody>
</table>
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 following sequence of attributes.

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.

The default value given in the table is used either:

- When a Quota instance is created, and no value is supplied for the field. In this case, the field is created with the value indicated
- When a Quota instance is reset using the “Reset Quota” command. 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 gets created with the value indicated

Table 9: Quota Entity Definition

<table>
<thead>
<tr>
<th>Name (XML tag)</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
<th>Quota Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>usage</td>
<td>—</td>
<td>—</td>
<td>Sequence (multiplicity = 1)</td>
<td>1/2/3</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
<td>—</td>
<td>Version of the schema.</td>
<td>1/2/3</td>
</tr>
<tr>
<td>quota</td>
<td>—</td>
<td>—</td>
<td>Sequence (multiplicity = N)</td>
<td>1/2/3</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>---</td>
<td>Quota name (identifier).</td>
<td>1/2/3</td>
</tr>
<tr>
<td>cid</td>
<td>String</td>
<td>---</td>
<td>Internal identifier used to identity a quota within a subscriber profile.</td>
<td>1/2/3</td>
</tr>
<tr>
<td>time</td>
<td>String</td>
<td>Empty string</td>
<td>This element tracks the time-based resource consumption for a Quota.</td>
<td>1/2/3</td>
</tr>
<tr>
<td>Name (XML tag)</td>
<td>Type</td>
<td>Default Value</td>
<td>Description</td>
<td>Quota Versions</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>totalVolume</td>
<td>String</td>
<td>“0”</td>
<td>This element tracks the bandwidth volume-based resource consumption for a Quota.</td>
<td>1/2/3</td>
</tr>
<tr>
<td>inputVolume</td>
<td>String</td>
<td>“0”</td>
<td>This element tracks the upstream bandwidth volume-based resource consumption for a Quota.</td>
<td>1/2/3</td>
</tr>
<tr>
<td>outputVolume</td>
<td>String</td>
<td>“0”</td>
<td>This element tracks the downstream bandwidth volume-based resource consumption for a Quota.</td>
<td>1/2/3</td>
</tr>
<tr>
<td>serviceSpecific</td>
<td>String</td>
<td>Empty string</td>
<td>This element tracks service-specific resource consumption for a Quota.</td>
<td>1/2/3</td>
</tr>
<tr>
<td>nextResetTime</td>
<td>String</td>
<td>Empty string</td>
<td>When set, it indicates the time after which the usage counters need to be reset.</td>
<td>1/2/3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>See section 4.3 for format details.</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
<td>Empty string</td>
<td>Type of the resource in use.</td>
<td>2/3</td>
</tr>
<tr>
<td>grantedTotalVolume</td>
<td>String</td>
<td>“0”</td>
<td>Granted Total Volume represents the granted total volume of all the subscribers in the pool for pool quota. For individual quota, it represents the granted volume to all the PDN connections for that subscriber.</td>
<td>2/3</td>
</tr>
<tr>
<td>grantedInputVolume</td>
<td>String</td>
<td>“0”</td>
<td>Granted Input Volume.</td>
<td>2/3</td>
</tr>
<tr>
<td>grantedOutputVolume</td>
<td>String</td>
<td>“0”</td>
<td>Granted Output Volume.</td>
<td>2/3</td>
</tr>
<tr>
<td>grantedTime</td>
<td>String</td>
<td>Empty string</td>
<td>Granted Total Time.</td>
<td>2/3</td>
</tr>
<tr>
<td>grantedServiceSpecific</td>
<td>String</td>
<td>Empty string</td>
<td>Granted Service Specific Units.</td>
<td>2/3</td>
</tr>
<tr>
<td>QuotaState</td>
<td>String</td>
<td>Empty string</td>
<td>State of the resource in use.</td>
<td>3</td>
</tr>
<tr>
<td>RefinstanceId</td>
<td>String</td>
<td>Empty string</td>
<td>Instance-id of the associated provisioned pass, top-up or roll-over.</td>
<td>3</td>
</tr>
</tbody>
</table>

### 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 State entity supports the following sequence of attributes:

**Table 10: State Entity Definition**

<table>
<thead>
<tr>
<th>Name (XML tag)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>—</td>
<td>Sequence (multiplicity is 1)</td>
</tr>
</tbody>
</table>
### Dynamic Quota

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

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

The DynamicQuota entity supports the following sequence of attributes:

**Table 11: Dynamic Quota Entity Definition**

<table>
<thead>
<tr>
<th>Name (XML tag)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>String</td>
<td>Version of the schema.</td>
</tr>
<tr>
<td>property</td>
<td>—</td>
<td>Sequence (multiplicity is N)</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>The property name.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>Value associated with the given property.</td>
</tr>
<tr>
<td>definition</td>
<td>—</td>
<td>Sequence (multiplicity is 1)</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
<td>Version of the schema.</td>
</tr>
<tr>
<td>DynamicQuota</td>
<td>—</td>
<td>Sequence (multiplicity is N)</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
<td>Identifies the dynamic quota type.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>The class identifier for a pass or top-up. This name is used to match top-ups</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to quota definitions on the PCRF. This name is used in policy conditions and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>actions on the PCRF.</td>
</tr>
<tr>
<td>InstanceId</td>
<td>String</td>
<td>A unique identifier to identify this instance of a dynamic quota object.</td>
</tr>
<tr>
<td>Priority</td>
<td>String</td>
<td>An integer represented as a string. This number allows service providers to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specify when one pass or top-up is used before another pass or top-up.</td>
</tr>
<tr>
<td>InitialTime</td>
<td>String</td>
<td>An integer represented as a string. The number of seconds initially granted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for the pass/top-up.</td>
</tr>
<tr>
<td>InitialTotalVolume</td>
<td>String</td>
<td>An integer represented as a string. The number of bytes of total volume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>initially granted for the pass/top-up.</td>
</tr>
<tr>
<td>InitialInputVolume</td>
<td>String</td>
<td>An integer represented as a string. The number of bytes of input volume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>initially granted for the pass/top-up.</td>
</tr>
<tr>
<td>InitialOutputVolume</td>
<td>String</td>
<td>An integer represented as a string. The number of bytes of output volume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>initially granted for the pass/top-up.</td>
</tr>
<tr>
<td>InitialServiceSpecific</td>
<td>String</td>
<td>An integer represented as a string. The number of service specific units</td>
</tr>
<tr>
<td></td>
<td></td>
<td>initially granted for the pass/top-up.</td>
</tr>
<tr>
<td>activationdatetime</td>
<td>String</td>
<td>The date/time after which the pass or top-up may be active.</td>
</tr>
</tbody>
</table>
### Pool Data

#### Pool Profile

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

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

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

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

#### Table 12: Pool Profile Entity Definition

<table>
<thead>
<tr>
<th>Name (XML tag)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool</td>
<td>—</td>
<td>Sequence (multiplicity is 1)</td>
</tr>
<tr>
<td>PoolID</td>
<td>String</td>
<td>Pool identifier (1 to 22 numeric digits, minimum value of 1).</td>
</tr>
</tbody>
</table>
| BillingDay     | Uint8      | The day of the month (1 to 31) when the associated quota pool is reset.  
<pre><code>              | 0 indicates that the default value configured at the PCRF level is used.      |
</code></pre>
<p>| 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.                                                           |
| 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.                                 |</p>
<table>
<thead>
<tr>
<th>Name (XML tag)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom5</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom6</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom7</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom8</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom9</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom10</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom11</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom12</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom13</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom14</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom15</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom16</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom17</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom18</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom19</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
<tr>
<td>Custom20</td>
<td>String</td>
<td>Fields used to store customer-specific data.</td>
</tr>
</tbody>
</table>

### 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 may 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 4.1.2.

### 4.2.3 Pool State

The PoolState entity is written by the PCRF to store the state of various properties managed as a part of the 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 may be present based on the version number value of the entity being accessed. In UDR, there is only one version number of 1.

The PoolState entity attributes are the same as defined for the State entity in section 4.1.3.
4.2.4 Pool Dynamic Quota

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

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

The PoolDynamicQuota entity attributes are the same as defined for the DynamicQuota entity in section 4.1.4.

4.3 Date/Timestamp Format

The Date/Timestamp format used by many fields is:

\[
\text{CCYY-MM-DDThh:mm:ss[<Z|<+|->hh:mm>]}\]

This corresponds to either:

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

Where:

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

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

- \text{2015-06-04T15:43:00} (local time)
- \text{2015-06-04T15:43:00Z} (UTC time)
- \text{2015-06-04T15:43:00+02:00} (positive offset from UTC)
- \text{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

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

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 Subscriber Entity Configuration.

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 new subscriber.

```xml
<?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="fieldNameN">fieldValueN</field>
  ]
</subscriber>
```
• keyNameX: A key field within the Subscriber Profile
  Value is either IMSI, MSISDN, NAI, or AccountId
• keyValueX: Corresponding key field value assigned to keyNameX
• fieldNameX: A user defined field within the Subscriber Profile
• fieldValueX: Corresponding field value assigned to fieldNameX

One key is mandatory. Any combination of key types are allowed. More than one occurrence of each key type (such as, 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

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Successfully created</td>
</tr>
<tr>
<td>400</td>
<td>MSR4000</td>
<td>Invalid content request data supplied</td>
</tr>
<tr>
<td>400</td>
<td>MSR4003</td>
<td>A key is detected to be already in the system for another subscriber</td>
</tr>
<tr>
<td>400</td>
<td>MSR4004</td>
<td>The field list does not contain at least one unique key</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>400</td>
<td>MSR4103</td>
<td>A key is detected to be already in the system for an AE subscriber</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Subscriber field is not defined</td>
</tr>
</tbody>
</table>

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>
</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 given IMSI.

Request URL
POST {baseURI}/msr/sub

Request Content
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="MSISDN">5141112223334</field>
  <field name="IMSI">184126781612121</field>
  <field name="BillingDay">2</field>
  <field name="Entitlement">DayPass</field>
</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 no key values were supplied.

HTTP Status Code
400

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

Request 5
A subscriber is created, with MSISDN and IMSI keys. The BillingDay and Custom15 fields are set.

NOTE: 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 given 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 given 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 within 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 within the Subscriber Profile
  Value is either IMSI, MSISDN, NAI, or AccountId
• **keyValueX**: Corresponding key field value assigned to keyNameX
• **fieldNameX**: A user defined field within 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

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Successfully located the subscriber</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Could not find the subscriber by key</td>
</tr>
</tbody>
</table>

### Examples

**Request 1**
The subscriber with the given AccountId is retrieved. The subscriber exists.

**Request URL**
GET {baseURI}/msr/sub/AccountId/10404723525

**Request Content**
None

**Response 1**
The request is successful, and the subscriber was retrieved.

**HTTP Status Code**
200

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

**Request 2**
The subscriber with the given IMSI is retrieved. The subscriber does not exist.

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

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

- `keyName`: A key field within 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="keyNameN">keyValueN</field>
  ...
</subscriber>
```
Provisioning

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

One key is mandatory. Any combination of key types are allowed. More than one occurrence of each key type (such as 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

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

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
<?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 within 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

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>The subscriber was successfully deleted</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Could not find the subscriber by key</td>
</tr>
<tr>
<td>409</td>
<td>MSR4055</td>
<td>Cannot delete, subscriber belongs to a pool</td>
</tr>
</tbody>
</table>

Examples

Request 1

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

Request URL

DELETE {baseURI}/msr/sub/MSISDN/33123654862
Request Content
None

Response 1
The request is successful.

HTTP Status Code
204

Response Content
None.

Request 2
The subscriber with the given NAI is deleted. The subscriber exists. The subscriber is a member of a pool.

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

Request Content
None

Response 2
The request fails, because the subscriber is a member of a pool.

HTTP Status Code
409

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

5.2 Subscriber Profile Field Commands

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

### 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 performed for the fields defined as multi-value field in the Subscriber Entity Configuration. Any pre-existing values for the field are not affected.

All existing values are retained, and the new 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 would have the value “a;b;c;d”.

If a value being added exists, the request fails.

If the value is being added to a filed that does not exist, it is created.

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

**Prerequisites**

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

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

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

**Request URL**

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

- `keyName`: A key field within the Subscriber Profile
  
  Value is either IMSI, MSISDN, NAI, or AccountId

- `keyValue`: Corresponding key field value assigned to `keyName`

- `fieldName`: A user defined field within the Subscriber Profile

- `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 19: Add Field Value Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Successfully added field values</td>
</tr>
<tr>
<td>400</td>
<td>MSR4005</td>
<td>Field does not support multiple values</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>400</td>
<td>MSR4066</td>
<td>Field value exists</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Subscriber field is not defined</td>
</tr>
</tbody>
</table>

Examples

Request 1
A request is made to add the value DayPass to the Entitlement field. The Entitlement field is a valid multi-value field. The DayPass value is not already present in the Entitlement field.

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

Request Content
None

Response 1
The request is successful, and the value was added to the Entitlement field.

HTTP Status Code
200

Response Content
None.

Request 2
A request is made to add the values DayPass and HighSpeedData to the Entitlement field. The Entitlement field is a valid multi-value field. The DayPass and HighSpeedData values are not already present in the Entitlement field.

Request URL
POST {baseURI}/msr/sub/NAI/dad@op.com/field/Entitlement/DayPass;HighSpeedData
Provisioning

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 to add two additional MSISDN values. Currently, the subscriber only has the MSISDN 15141234567.

Request URL
POST {baseURI}/msr/sub/MSISDN/5141234567/field/MSISDN/
1416112222; 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 for 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 within the Subscriber Profile
  Value is either IMSI, MSISDN, NAI, or AccountId
- keyValue: Corresponding key field value assigned to keyName
- fieldName: A user defined field within the Subscriber Profile

Request Content
None.

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

```xml
<?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 within the Subscriber Profile
- fieldValueX: Corresponding field value assigned to fieldName

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

Table 20: Get Field Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested field exists for subscriber</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Subscriber field is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4065</td>
<td>Field is not set</td>
</tr>
</tbody>
</table>
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
Provisioning

Request Content
None

Response 3
The request is successful, and an empty value is returned.

HTTP Status Code
404

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

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.

Depending on the field entered, 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 current field value of “a;b;c” would be successful, but an attempt to get the value “A” would fail

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

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

The requested field must contain the values supplied in the fieldValue.

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

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

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

```xml
<?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 within the Subscriber Profile
- `fieldValueX`: Corresponding field value assigned to `fieldNameX`

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

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>-</td>
<td>Requested field exists for subscriber with given value</td>
</tr>
<tr>
<td>400</td>
<td>MSR4053</td>
<td>Subscriber and field exist, but values do not match</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Subscriber field is not defined</td>
</tr>
</tbody>
</table>

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
<?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 the values of the field, which means that any existing values for the field are deleted first. For multi-value fields, all previous values are erased and the new set of values is inserted. Adding values to a current 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 within the Subscriber Profile
  Value is either IMSI, MSISDN, NAI, or AccountId
- keyValue: Corresponding key field value assigned to keyName
- fieldName: A user defined field within the Subscriber Profile
- fieldValue: Corresponding field value assigned to fieldName

NOTE: A field name cannot be for a key value—such as IMSI, MSISDN, NAI, or AccountId

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**

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Fields were successfully updated</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>The value provided for the field is invalid</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Subscriber field is not defined</td>
</tr>
</tbody>
</table>

**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 two 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 new set of values is inserted. Adding values to a current set is accomplished using Add Field Value.

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

ALL fields that can be modified in the "single field" request can also be modified in the "multiple fields" request. Two or three fields can be updated at once. Updating only a single field 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 fields fieldNameX must all be valid fields in the Subscriber Profile.

Request URL

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

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

- keyValue: Corresponding key field value assigned to keyName

- fieldNameX: A user defined field within the Subscriber Profile

  NOTE: A field name cannot be for a key value—such as IMSI, MSISDN, NAI, or AccountId

- fieldValueX: Corresponding field value assigned to fieldNameX

  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 23: Update Multiple Fields Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Fields were successfully updated</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>The value provided for the field is invalid</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4057</td>
<td>Request only contains one field to update</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Subscriber field is not defined</td>
</tr>
</tbody>
</table>

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 the removal of the field from the subscriber profile. The field is not present, not just the value is empty.

The field being deleted does not need to have a current 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 (such as 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 within the Subscriber Profile
  Value is either IMSI, MSISDN, NAI, or AccountId
- keyValue: Corresponding key field value assigned to keyName
- fieldName: A user defined field within the Subscriber Profile

Request Content

None.

Response Content

None.
Table 24: Delete Field Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Field was successfully deleted</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>400</td>
<td>MSR4069</td>
<td>At least one key is required</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Subscriber field is not defined</td>
</tr>
</tbody>
</table>

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 performed for the fields defined as multi-value field in the Subscriber Entity Configuration.
Each individual value is removed from the Subscriber Profile. If a supplied value does not exist, then it is ignored. For example, if a profile contains values "a;b;c" and a request to delete "a;b" is made, this succeeds and the profile is left with "c" as the value. If the profile contains "a;b;c" and a request is made to delete "c;d" the request succeeds and the profile is left with "a;b" as the value.

If all values are removed, the field is removed from the subscriber profile (there is no XML element present).

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

**Prerequisites**

A 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 within the Subscriber Profile
  
  Value is either IMSI, MSISDN, NAI, or AccountId

- **keyValue**: Corresponding key field value assigned to keyName

- **fieldName**: A user defined field within the Subscriber Profile

  **NOTE**: A field name cannot be for a key value—such as 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 25: Delete Field Value Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Requested fields were successfully deleted</td>
</tr>
<tr>
<td>400</td>
<td>MSR4005</td>
<td>Field does not support multiple values</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4069</td>
<td>At least one key is required</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Subscriber field is not defined</td>
</tr>
</tbody>
</table>
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 following commands perform opaque data operations. They can be used on entities defined as either opaque or transparent. The opaque data operation operates on the entity at the XML blob level. The contents of the entity is set, returned, or deleted.

Table 26: Summary of Subscriber Opaque Data Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Opaque Data</td>
<td>Create/update opaque data of the specified type</td>
<td>MSISDN, IMSI, NAI or AccountId</td>
<td>PUT {baseURI}/msr/sub/keyName/keyValue/data/opaqueDataType</td>
</tr>
<tr>
<td>Get Opaque Data</td>
<td>Retrieve opaque data of the specified type</td>
<td></td>
<td>GET {baseURI}/msr/sub/keyName/keyValue/data/opaqueDataType</td>
</tr>
<tr>
<td>Delete Opaque Data</td>
<td>Delete opaque data of the specified type</td>
<td></td>
<td>DELETE {baseURI}/msr/sub/keyName/keyValue/data/opaqueDataType</td>
</tr>
</tbody>
</table>

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.

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

- keyValue: A key field within 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
<?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”

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

Response Content
None.

Table 27: Set Opaque Data Response Status/Error Codes

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

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 already has an existing State entity.

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

Request Content
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="state">
    <![CDATA[
    <?xml version="1.0" encoding="UTF-8"?>
    <state>
      <version>1</version>
      <property>
        <name>mcc</name>
        <value>315</value>
      </property>
      <property>
        <name>expire</name>
        <value>2010-02-09T11:20:32</value>
      </property>
    </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 keyCode and keyValue in the request.

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

Prerequisites
A subscriber with the key of the keyCode/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

• keyCode: A key field within the Subscriber Profile
  Value is either IMSI, MSISDN, NAI, or AccountId
• keyValue: Corresponding key field value assigned to keyCode
• 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

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested data exists for subscriber</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4053</td>
<td>Data type is not set for this subscriber</td>
</tr>
</tbody>
</table>

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
<?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>
      </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[
      <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.
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 within 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

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Data was successfully deleted</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
</tbody>
</table>

**Examples**

**Request 1**

A request is made to delete the quota opaque data.

**Request URL**

DELETE {baseURI}/msr/sub/MSISDN/33123654862/data/quota

**Request Content**

None

**Response 1**

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

**HTTP Status Code**

204
Response Content
None.

Request 2
A request is made to delete the state opaque data.

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

Request Content
None

Response 2
The request is successful, and the State opaque data was deleted.

HTTP Status Code
204

Response Content
None.

Request 3
A request is made to delete the state opaque data. The subscriber does not have any State opaque data.

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

Request Content
None

Response 3
The request is successful, although no State opaque data was deleted.

HTTP Status Code
204

Response Content
None.

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 within the Quota entity.

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

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 MSR4099 error being returned.
Table 30: Summary of Subscriber Data Row Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Row</td>
<td>Create/update data row in data of the specified type.</td>
<td>(MSISDN, IMSI, NAI or AccountId) and Row Identifier</td>
<td>PUT {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue</td>
</tr>
<tr>
<td>Get Row</td>
<td>Retrieve data row from data of the specified type.</td>
<td></td>
<td>GET {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue</td>
</tr>
<tr>
<td>Delete Row</td>
<td>Delete data row within data of the specified type</td>
<td></td>
<td>DELETE {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue</td>
</tr>
</tbody>
</table>

5.4.1 Set Row

Description
This operation creates or updates an existing data row for the subscriber identified by the `keyName` and `keyValue`.

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

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

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

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

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

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

- `keyName`: A key field within 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 within the data blob

Request Content

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

- `rowValue`: Contents of the XML data “blob”, with the row data

**NOTE:** the `rowValue` is in the same format as the Quota entity, just containing a single row, the row being added
The data contained within 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

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Data row was successfully created/updated</td>
</tr>
<tr>
<td>400</td>
<td>MSR4000</td>
<td>Request content is not valid</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>400</td>
<td>MSR4067</td>
<td>Multiple matching rows found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
</tbody>
</table>

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
<?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>
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 /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
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.

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

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

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

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

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

- keyName: A key field within 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 within the transparent data blob

Request Content
None.

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

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

- transparentDataType: A user defined type/name for the transparent data
  Value is quota for the Quota transparent data
- cdataRowValue: Contents of the XML data “blob”, with the row data
Table 32: Get Row Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>-</td>
<td>Requested data row exists for subscriber</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
</tbody>
</table>

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 and 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>
      </usage>
    ]]></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.

The `rowIdValue` is case-sensitive. If a row existed called “DayPass”, then an attempt to delete a row called “DayPass” would be successful, but an attempt to delete a row called “DAYPASS” would fail.

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

- `keyName`: A key field within 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 within the transparent data blob

Request Content
None.

Response Content
None.
Table 33: Delete Row Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Data row was successfully deleted</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
</tbody>
</table>

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>

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 within the Quota entity.

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

Subscriber data row field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an HTTP Status Code 400, with an MSR4099 error being returned.

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

Description

This operation retrieves a field within 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`.

The `rowIdValue` is case-sensitive. If a row existed called “DayPass”, then an attempt to get a field in a row called “DayPass” would be successful, but an attempt to get a field in a row called “DAYPASS” would fail.

Prerequisites

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

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

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

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

Request URL

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

- `keyName`: A key field within 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 within the transparent data blob
- `fieldName`: A user defined field within the transparent data row

Request Content

None.

Response Content

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

```xml
<?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

- cdataRowFieldValue: Contents of the XML data “blob”, with the field from the row data

### Table 35: Get Row Field Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested data row field exists for subscriber</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4065</td>
<td>Field is not set</td>
</tr>
</tbody>
</table>

### 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
<?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
<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>
```

5.5.2 Get Row Field Value

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

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

The `rowIdValue` is case-sensitive. If a row existed called “DayPass”, then an attempt to get a field value in a row called “DayPass” would be successful, but an attempt to get a field value in a row called “DAYPASS” would fail.

The `fieldValue` is case-sensitive. An attempt to get the value “Data” from a current field value of “Data” would be successful, but an attempt to get the value “DATA” would fail.
Prerequisites
A subscriber with the key of the keyName/keyValue supplied must exist.

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

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

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

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

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

- keyName: A key field within 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 within the transparent data blob
- fieldName: A user defined field within 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
- 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

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

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
<?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 2330 in the Weekday data row of the quota transparent data for a subscriber. Two instances of the Weekday data row exist. The inputVolume field exists in both rows, and is set to the value 3220 in both rows.

Request URL
GET {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday/inputVolume/3220

Request Content
None

Response 3
The request is successful, and the field from two matching Weekday rows are returned.

HTTP Status Code
200

Response Content
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
.getVersion>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
<?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>
```

5.5.3 Update Row Field

Description
This operation updates a fields within a transparent data row for the subscriber identified by the `keyName` and `keyValue`.

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

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

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

The `rowIdValue` is case-sensitive. If a row already existed called “DayPass”, then an attempt to update a field in a row called “DayPass” would be successful, but an attempt to update a field in a row called “DAYPASS” would fail

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

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

A data row with the given identifier within the transparent data should exist for the subscriber.
The field name specified must be a valid field for the Entity as defined in the SEC. The field must be updatable.

**Request URL**

```
PUT (baseURI)/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/fieldName/fieldVal
ue
```

- **keyName**: A key field within 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 within the transparent data blob

- **fieldName**: A user defined field within 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**

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Requested transparent data row field was successfully created</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4067</td>
<td>Multiple matching rows found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
</tbody>
</table>
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>

5.5.4 Delete Row Field

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

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

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

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 existed called “DayPass”, then an attempt to delete a field in a row called “DayPass” would be successful, but an attempt to delete a field in a row called “DAYPASS” would fail.

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

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

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

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

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

- **keyName**: A key field within 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 within the transparent data blob
- **fieldName**: A user defined field within the transparent data row

Request Content
None.
Response Content
None.

Table 38: Delete Row Field Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Requested transparent data row field was successfully deleted</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4067</td>
<td>Multiple matching rows found</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
</tbody>
</table>

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>

5.6 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 within the Quota entity.

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

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

Table 39: Summary of Subscriber Special Operation Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Quota</td>
<td>Reset the fields within the specified Quota</td>
<td>(MSISDN, IMSI, NAI or AccountId) and Row Identifier</td>
<td>POST {BaseURI}/msr/sub/keyName/KeyValue/data/transparentDataType/rowIdValue</td>
</tr>
</tbody>
</table>

5.6.1 Reset Quota

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

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

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

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

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.
Provisioning

Request URL

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

- keyName: A key field within 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 within the transparent data blob

Request Content

None.

Response Content

None.

Table 40: Reset Quota Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Requested transparent data row was successfully reset</td>
</tr>
<tr>
<td>400</td>
<td>MSR4067</td>
<td>Multiple matching rows found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
<tr>
<td>409</td>
<td>MSR4063</td>
<td>Entity cannot be reset</td>
</tr>
</tbody>
</table>

Examples

Request 1

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

Request URL

POST \{baseURI\}/msr/sub/MSISDN/33123654862/data/quota/Q1

Request Content

None

Response 1

The request is successful, and the specified Quota row was reset.
HTTP Status Code
204

Response Content
None.

Request 2
A request is made to reset the Q1 Quota row for a subscriber. The subscriber does not have Quota transparent data.

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

Request Content
None

Response 2
The request fails because the subscriber does not have Quota transparent data.

HTTP Status Code
404

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

Request 3
A request is made to reset the Q6 Quota row for a subscriber. The subscriber has Quota transparent data, but the Quota transparent data does not contain a Quota row called Q6.

Request URL
POST {baseURI}/msr/sub/MSISDN/33123654862/data/quota/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>
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 PoolID value associated with the pool.

Table 41: Summary of Pool Profile Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Pool</td>
<td>Create a pool profile</td>
<td>—</td>
<td>POST {baseURI}/msr/pool</td>
</tr>
<tr>
<td>Get Pool</td>
<td>Get pool profile data</td>
<td>PoolID</td>
<td>GET {baseURI}/msr/pool/poolId</td>
</tr>
<tr>
<td>Update Pool</td>
<td>Replace an existing pool profile</td>
<td>PoolID</td>
<td>PUT {baseURI}/msr/pool/poolId</td>
</tr>
<tr>
<td>Delete Pool</td>
<td>Delete all pool profile data and all opaque data associated with the pool</td>
<td></td>
<td>DELETE {baseURI}/msr/pool/poolId</td>
</tr>
</tbody>
</table>

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

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 pool with the supplied PoolID must not exist.

Request URL

POST {baseURI}/msr/pool

Request Content

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

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

- poolId: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999
Provisioning

- fieldNameX: A user defined field within the Pool Profile
- fieldValueX: Corresponding field value assigned to fieldNameX

PoolID/field order in the request is not important

**Response Content**

None.

**Table 42: Create Pool Response Status/Error Codes**

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

**Examples**

**Request 1**

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

**Request URL**

POST {baseURI}/msr/pool

**Request Content**

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

**Response 1**

The request is successful, and the pool was created.

**HTTP Status Code**

201

**Response Content**

None.
Request 2
A pool is created, with a PoolID key. The BillingDay and Entitlement fields are set. A pool exists with the given PoolID.

Request URL
POST {baseURI}/msr/pool

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

Response 2
The request fails. The error code indicates the PoolID exists.

HTTP Status Code
400

Response Content
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4004">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: PoolID 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="PoolID">poolId</field>
  [ <field name="fieldName1">fieldValue1</field> ]
<field name="fieldName2">fieldValue2</field>
:
<field name="fieldNameN">fieldValueN</field>
]
</pool>

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

PoolID/field order in the request is not important

Table 43: Get Pool Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Successfully located the pool</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Could not find the pool by PoolID</td>
</tr>
</tbody>
</table>

Examples

Request 1
The pool with the given PoolID is retrieved. The pool exists.

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

Request Content
None

Response 1
The request is successful, and the pool was retrieved.

HTTP Status Code
200

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

Request 2
The pool with the given PoolID is retrieved. The pool does not exist.
Request URL
GET {baseURI}/msr/pool/222200

Request Content
None

Response 2
The request fails, 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 PoolID, all existing data for the pool is completely removed and replaced by the request content. Therefore, it is not necessary to include the PoolID from the URI in the request content (although it is not an error if it is included).

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

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

Request URL
PUT {baseURI}/msr/pool/poolId

• poolId: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999

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

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

• poolId: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999

• fieldNameX: A user defined field within the Pool Profile
• fieldValueX: Corresponding field value assigned to fieldNameX

PoolID/field order in the request is not important

Response Content
None.

Table 44: Update Pool Response Status/Error Codes

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

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.
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: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999

Request Content
None.

Response Content
None.

Table 45: Delete Pool Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>The pool was successfully deleted</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Could not find the pool by PoolID</td>
</tr>
<tr>
<td>409</td>
<td>MSR4055</td>
<td>The pool could not be deleted as it has member subscribers</td>
</tr>
</tbody>
</table>

Examples

Request 1
The pool with the given PoolID is deleted. The pool exists, and has no member subscribers.

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

Request Content
None

Response 1
The request is successful.

HTTP Status Code
204

Response Content
None.
Request 2
The pool with the given PoolID is deleted. The pool exists, 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 46: Summary of Pool Profile Field Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Field Value</td>
<td>Adds a value to the specified field. This operation does not affect any pre-existing values for the field</td>
<td>POST {baseURI}/msr/pool/poolId/field/fieldName/fieldValue</td>
<td></td>
</tr>
<tr>
<td>Get Field</td>
<td>Retrieve values for the specified field</td>
<td>GET {baseURI}/msr/pool/poolId/field/fieldName</td>
<td></td>
</tr>
<tr>
<td>Get Field Value</td>
<td>Retrieve the single value for the specified field (if set as specified)</td>
<td>GET {baseURI}/msr/pool/poolId/field/fieldName/fieldValue</td>
<td></td>
</tr>
<tr>
<td>Update Field Value</td>
<td>Update field to the specified value</td>
<td>PUT {baseURI}/msr/pool/poolId/field/fieldName/fieldValue</td>
<td></td>
</tr>
<tr>
<td>Update Multiple Fields</td>
<td>Update multiple fields to the specified values</td>
<td>PUT {baseURI}/msr/pool/poolId/multipleFields/fieldName1/fieldValue1/fieldName2/fieldValue2/…</td>
<td></td>
</tr>
<tr>
<td>Delete Field</td>
<td>Delete all values for the specified field</td>
<td>DELETE {baseURI}/msr/pool/poolId/field/fieldName</td>
<td></td>
</tr>
<tr>
<td>Delete Field Value</td>
<td>Delete a value for the specified field</td>
<td>DELETE {baseURI}/msr/pool/poolId/field/fieldName/fieldValue</td>
<td></td>
</tr>
</tbody>
</table>
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 performed for the fields defined as multi-value field in the Subscriber Entity Configuration. Any pre-existing values for the field are not affected.

All existing values are retained, and the new 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 would have the value “a;b;c;d”.

If a value being added exists, the request fails.

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 current field value of “a;b;c” would fail, but an attempt to add the value “A” would be successful and result in the field value being “a;b;c;A”

Prerequisites
A pool with the PoolID 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 already be present in the field.

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

- poolId: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999
- fieldName: A user defined field within the Pool Profile
- 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 47: Add Field Value Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Successfully added field values</td>
</tr>
<tr>
<td>400</td>
<td>MSR4005</td>
<td>Field does not support multiple values</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
</tbody>
</table>
### HTTP Status Code Table

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4066</td>
<td>Field value exists</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Pool field is not defined</td>
</tr>
</tbody>
</table>

### Examples

**Request 1**
A request is made to add the value *DayPass* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* value is not already present in the *Entitlement* field.

**Request URL**

```
POST {baseURI}/msr/pool/100000/field/Entitlement/DayPass
```

**Request Content**

None

**Response 1**
The request is successful, and the value was added to the *Entitlement* field.

**HTTP Status Code**

200

**Response Content**

None.

**Request 2**
A request is made to add the values *DayPass* and *HighSpeedData* to the *Entitlement* field. The *Entitlement* field is a valid multi-value field. The *DayPass* and *HighSpeedData* values are not already present in the *Entitlement* field.

**Request URL**

```
POST {baseURI}/msr/pool/200000/field/Entitlement/DayPass;HighSpeedData
```

**Request Content**

None

**Response 2**
The request is successful, and the values were added to the *Entitlement* field.

**HTTP Status Code**

200

**Response Content**

None.
6.2.2 Get Field

Description
This operation retrieves the values for the specified field for the pool identified by the `poolId`.
Depending on the field entered, there may be multiple field-value pairs returned by this operation.

Prerequisites
A pool with the PoolID 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`: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999
- `fieldName`: A user defined field within the Pool Profile

Request Content
None.

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

```xml
<?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 within the Pool Profile
- `fieldValueX`: Corresponding field value assigned to `fieldName`

Table 48: Get Field Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested field exists for pool</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Pool field is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4065</td>
<td>Field is not set</td>
</tr>
</tbody>
</table>
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.

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 current field value of “a;b;c” would be successful, but an attempt to get the value “A” would fail

Prerequisites
A pool with the PoolID 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: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999

- fieldName: A user defined field within the Pool Profile
• 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**

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

```xml
<?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 within the Pool Profile

• fieldValueX: Corresponding field value assigned to fieldName

**Table 49: Get Field Value Response Status/Error Codes**

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested field exists for pool</td>
</tr>
<tr>
<td>400</td>
<td>MSR4053</td>
<td>Pool and field exist, but values do not match</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Pool field is not defined</td>
</tr>
</tbody>
</table>

**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 a set of values for a field, which means that any existing values for the field are deleted first. For multi-value fields, all previous values are erased and the new set of values is inserted. Adding values to a current set is accomplished using Add Field Value.

This command cannot be used to update the PoolID.

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: PoolID value of the pool. Numeric value, 1 to 22 digits in length
Values: 1 to 9999999999999999999999

- **fieldName**: A user defined field within the Pool Profile
- **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 50: Update Field Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Field was successfully updated</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>The value provided for the field is invalid</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Pool field is not defined</td>
</tr>
</tbody>
</table>

### 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 the value set of the field, which means that any existing values for the field are deleted first. For multi-value fields, all previous values are erased and the new set of values 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 fields" 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.

This command cannot be used to update the PoolID.

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

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 51: Update Multiple Fields Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Field was successfully updated</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>The value provided for the field is invalid</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
</tbody>
</table>
### HTTP Status Code Table

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>MSR4057</td>
<td>Request only contains one field to update</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Pool field is not defined</td>
</tr>
</tbody>
</table>

### 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 removal of the field from the pool profile. The field is not present, not just the value is empty.

The field being deleted does not need to have a current value. It can be empty (deleted) already, and the request succeeds.

This command cannot be used to delete the PoolID.

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: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 99999999999999999999
  • fieldName: A user defined field within the Subscriber Profile

Request Content
None.

Response Content
None.

Table 52: Delete Field Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Field was successfully deleted</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Pool field is not defined</td>
</tr>
</tbody>
</table>

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.
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 performed for the fields defined as multi-value field in the Subscriber Entity Configuration.

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

If all values are removed, the field is removed from the Pool Profile (there is no XML element present).

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

Prerequisites
A pool with the key of the `poolId` supplied must exist.

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

Request URL
DELETE {baseURI}/msr/pool/poolId/field/fieldName/fieldValue

- `poolId`: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999
- `fieldName`: A user defined field within the Pool Profile
- `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 53: Delete Field Value Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Requested fields were successfully deleted</td>
</tr>
<tr>
<td>400</td>
<td>MSR4005</td>
<td>Field does not support multiple values</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
</tbody>
</table>
### 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**

```http
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**

```http
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 54: Summary of Pool Opaque Data Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Opaque Data</td>
<td>Create/update opaque data of the specified type</td>
<td>PoolID</td>
<td>PUT {baseURI}/msr/pool/poolId/data/opaqueDataType</td>
</tr>
<tr>
<td>Get Opaque Data</td>
<td>Retrieve opaque data of the specified type</td>
<td></td>
<td>GET {baseURI}/msr/pool/poolId/data/opaqueDataType</td>
</tr>
<tr>
<td>Delete Opaque Data</td>
<td>Delete opaque data of the specified type</td>
<td></td>
<td>DELETE {baseURI}/msr/pool/poolId/data/opaqueDataType</td>
</tr>
</tbody>
</table>

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.

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

Prerequisites
A pool with the 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: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  
  Values: 1 to 9999999999999999999999

- 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
<?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”

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

Response Content
None.

Table 55: Set Opaque Data Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Data was successfully created/updated</td>
</tr>
<tr>
<td>400</td>
<td>MSR4000</td>
<td>Request content is not valid</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
</tbody>
</table>

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
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="poolquota">
    <![CDATA[
      <?xml version="1.0" encoding="UTF-8"?>
      <usage>
        <quota name="AggregateLimit">
          <cid>9223372036854775807</cid>
          <time>3422</time>
          <totalVolume>1000</totalVolume>
          <inputVolume>980</inputVolume>
          <outputVolume>20</outputVolume>
          <serviceSpecific>12</serviceSpecific>
        </quota>
        <version>3</version>
      </usage>
    ]]>
  </data>
</pool>
```
Provisioning

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

Response 1
The request is successful, and the PoolQuota opaque data was created.

HTTP Status Code
201

Response Content
None.

Request 2
A request is made to update the poolstate opaque data. The pool already has an existing PoolState entity.

Request URL
PUT {baseURI}/msr/pool/100002/data/poolstate

Request Content
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="poolstate">
    <![CDATA[
      <?xml version="1.0" encoding="UTF-8"?>
      <state>
        <version>1</version>
        <property>
          <name>mcc</name>
          <value>315</value>
        </property>
        <property>
          <name>expire</name>
          <value>2010-02-09T11:20:32</value>
        </property>
        <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.

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: PoolID 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
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

Table 56: Get Opaque Data Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested opaque data exists for pool</td>
</tr>
<tr>
<td>HTTP Status Code</td>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4053</td>
<td>Data type is not set for this pool</td>
</tr>
</tbody>
</table>

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

**Request Content**
None

**Response 2**
The request is successful, and the PoolState opaque data is returned.

**HTTP Status Code**
200

**Response Content**

```xml
<?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>
```

### 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.

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.

**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: PoolID value of the pool. Numeric value, 1 to 22 digits in length
   Values: 1 to 9999999999999999999999
• opaqueDataType: A user-defined type/name for the opaque data
   Value is either poolquota, poolstate, or pooldynamicquota

Request Content
None.

Response Content
None.

Table 57: Delete Opaque Data Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Opaque data was successfully deleted</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
</tbody>
</table>

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 no PoolState opaque data was 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 within the PoolQuota entity.

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

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 MSR4099 error being returned.

Table 58: Summary of Pool Data Row Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Row</td>
<td>Create/update data row in data of the specified type.</td>
<td>PoolID and Row Identifier</td>
<td>PUT {baseURI}/msr/pool/poolId/data/ transparentDataType/rowIdValue</td>
</tr>
<tr>
<td>Get Row</td>
<td>Retrieve data row from data of the specified type.</td>
<td></td>
<td>GET {baseURI}/msr/pool/poolId/data/ transparentDataType/rowIdValue</td>
</tr>
<tr>
<td>Delete Row</td>
<td>Delete data row within data of the specified type</td>
<td></td>
<td>DELETE {baseURI}/msr/pool/poolId/data/ transparentDataType/rowIdValue</td>
</tr>
</tbody>
</table>

6.4.1 Set Row

Description
This operation creates 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 within 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.

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

If the transparent entity specified in entityName does not exist for the pool, it is created.

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

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

- poolId: PoolID 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
- rowIdValue: The row name value that identifies the row within the data blob

Request Content

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

- rowValue: Contents of the XML data “blob”, with the row data

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

The data contained within 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 59: Set Row Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Data row was successfully created/updated</td>
</tr>
<tr>
<td>400</td>
<td>MSR4000</td>
<td>Request content is not valid</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>400</td>
<td>MSR4067</td>
<td>Multiple matching rows found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
</tbody>
</table>
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
<?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
<?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.

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.

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

Prerequisites
A 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 given identifier within the transparent data should exist for the pool.

Request URL
GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue

- poolId: PoolID 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
- rowIdValue: The row name value that identifies the row within the transparent data blob

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>
transientDataType: A user defined type/name for the transparent data
Value is poolquota for the PoolQuota transparent data
• cdataRowValue: Contents of the XML data “blob”, with the row data

Table 60: Get Row Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested data row exists for pool</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
</tbody>
</table>

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>
      </quota>
    </usage>
  </data>
</pool>
Provisioning

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

Response 2
The request fails, as the data row does not exist.

HTTP Status Code
404

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

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[
      <usage>
        <inputVolume>0</inputVolume>
        <outputVolume>0</outputVolume>
        <serviceSpecific>12</serviceSpecific>
        <nextResetTime>2010-05-12T16:00:00-05:00</nextResetTime>
      </usage>
    </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.

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

The deletion of a non-existent data row is not considered an error.

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/rowIdValue

- poolId: PoolId value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 999999999999999999999
- 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 within the transparent data blob
Request Content
None.

Response Content
None.

Table 61: Delete Row Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Data row was successfully deleted</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
</tbody>
</table>

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.

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 within 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.

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 MSR4099 error being returned.

Table 62: Summary of Pool Data Row Field Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Row Field</td>
<td>Retrieve values for the specified field</td>
<td>PoolID and Row Identifier and Field name</td>
<td>GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName</td>
</tr>
<tr>
<td>Get Row Field Value</td>
<td>Retrieve a single value for the specified field</td>
<td></td>
<td>GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName/fieldValue</td>
</tr>
<tr>
<td>Update Field</td>
<td>Update field to the specified value</td>
<td></td>
<td>PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName/fieldValue</td>
</tr>
<tr>
<td>Delete Field</td>
<td>Delete all values for the specified field</td>
<td></td>
<td>DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName</td>
</tr>
</tbody>
</table>

6.5.1 Get Row Field

Description
This operation retrieves a field within 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.

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

Prerequisites
A 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 given identifier within the transparent data should exist for the pool.
The field name specified must be a valid field for the Entity as defined in the SEC.

Request URL

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

• poolId: PoolID 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
• rowIdValue: The row name value that identifies the row within the transparent data blob
• fieldName: A user defined field within the transparent data row

Request Content

None.

Response Content

A &lt;pool&gt; element that contains a &lt;data&gt; element, which contains the specified transparent data row field (if it exists) for the identified pool.

<?xml version="1.0" encoding="UTF-8"?>
&lt;pool&gt;
  &lt;data name="transparentDataType">
    <![CDATA[
cdataRowFieldValue
    ]]>)
  &lt;/data&gt;
&lt;/pool&gt;

• transparentDataType: A user defined type/name for the transparent data
  Value is poolquota for the PoolQuota transparent data
• cdataRowFieldValue: Contents of the XML data “blob”, with the field from the row data

Table 63: Request URL Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested data row field exists for pool</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4065</td>
<td>Field is not set</td>
</tr>
</tbody>
</table>
Examples

Request 1
A request is made to get the \textit{inputVolume} field in the \textit{PQ1} data row of the \textit{poolquota} transparent data for a pool.

Request URL
\texttt{GET} {\text{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
\begin{verbatim}
<?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>
\end{verbatim}

Request 2
A request is made to get the \textit{outputVolume} field in the \textit{Weekday} data row of the \textit{poolquota} transparent data for a pool. Two instances of the \textit{Weekday} data row exist.

Request URL
\texttt{GET} {\text{BaseURI}}/msr/pool/100000/data/poolquota/Weekday/outputVolume

Request Content
None

Response 2
The request is successful, and the field from two matching \textit{Weekday} rows are returned.

HTTP Status Code
200

Response Content
\begin{verbatim}
<?xml version="1.0" encoding="UTF-8"?>
<pool>
</pool>
\end{verbatim}
6.5.2 Get Row Field Value

Description
This operation retrieves a field with a given value, within 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.

The rowIdValue is case-sensitive. If a row existed called “DayPass”, then an attempt to get a field value in a row called “DayPass” would be successful, but an attempt to get a field value in a row called “DAYPASS” would fail

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

Prerequisites
A 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 given identifier within the transparent data should 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
GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName/fieldValue

- poolId: PoolID 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
- rowIdValue: The row name value that identifies the row within the transparent data blob
- fieldName: A user defined field within the transparent data row
- fieldValue: Corresponding field value assigned to fieldName
Provisioning

**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
<?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
- **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 64: Get Row Field Value Response Status/Error Codes**

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Requested data row field/value exists for pool</td>
</tr>
<tr>
<td>400</td>
<td>MSR4053</td>
<td>Data row field value does not match</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
</tbody>
</table>

**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 2330 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>

6.5.3 Update Row Field

Description
This operation updates a fields within 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 exist, it is created.

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

The rowIdValue is case-sensitive. If a row already existed called “DayPass”, then an attempt to update a field in a row called “DayPass” would be successful, but an attempt to update a field in a row called “DAYPASS” would fail.

Prerequisites
A 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 given identifier within the transparent data should 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
PUT {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName/fieldValue

- poolId: PoolID 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

- rowIdValue: The row name value that identifies the row within the transparent data blob
- fieldName: A user defined field within the transparent data row
- fieldValue: Corresponding field value assigned to fieldName

**Request Content**

None.

**Response Content**

None.

**Table 65: Update Row Field Response Status/Error Codes**

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>—</td>
<td>Requested transparent data row field was successfully created</td>
</tr>
<tr>
<td>400</td>
<td>MSR4051</td>
<td>Invalid value for a field</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4067</td>
<td>Multiple matching rows found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
</tbody>
</table>

**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

6.5.4 Delete Row Field

Description
This operation deletes a field within 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.

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 existed called “DayPass”, then an attempt to delete a field in a row called “DayPass” would be successful, but an attempt to delete a field in a row called “DAYPASS” would fail
Prerequisites
A 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 given identifier within the transparent data should 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
DELETE {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldName

- poolId: PoolID 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
- rowIdValue: The row name value that identifies the row within the transparent data blob
- fieldName: A user defined field within the transparent data row

Request Content
None.

Response Content
None.

Table 66: Delete Row Field Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Requested transparent data row field was successfully deleted</td>
</tr>
<tr>
<td>400</td>
<td>MSR4056</td>
<td>Field is not updatable</td>
</tr>
<tr>
<td>400</td>
<td>MSR4067</td>
<td>Multiple matching rows found</td>
</tr>
<tr>
<td>400</td>
<td>MSR4064</td>
<td>Occurrence constraint violation</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Pool is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4002</td>
<td>Field is not defined for this data type</td>
</tr>
<tr>
<td>404</td>
<td>MSR4049</td>
<td>Data type is not defined</td>
</tr>
<tr>
<td>404</td>
<td>MSR4058</td>
<td>Data type not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4059</td>
<td>Data row does not exist</td>
</tr>
</tbody>
</table>
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>

6.6 Additional Pool Commands

Table 67: Summary of Additional Pool Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Keys</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Member to Pool</td>
<td>Add subscriber to a Pool</td>
<td>PoolID and (MSISDN, IMSI, NAI or AccountId)</td>
<td>POST {BaseURI}/msr/pool/poolId/member/subKeyName/subKeyValue</td>
</tr>
<tr>
<td>Remove Member from Pool</td>
<td>Remove subscriber from a Pool</td>
<td></td>
<td>DELETE {BaseURI}/msr/pool/poolId/member/subKeyName/subKeyValue</td>
</tr>
</tbody>
</table>
### 6.6.1 Add Member to Pool

**Description**
This operation adds a Subscriber to 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 not already be a member of a pool.

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

**Request URL**

```
POST {BaseURI}/msr/pool/poolId/member/subKeyName/subKeyValue
```

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

- `subKeyName`: A key field within 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 68: Add Member to Pool**

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Subscriber successfully added to pool</td>
</tr>
<tr>
<td>400</td>
<td>MSR4060</td>
<td>Number of pool members exceeded</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4061</td>
<td>Specified pool does not exist</td>
</tr>
<tr>
<td>409</td>
<td>MSR4055</td>
<td>Subscriber is already a member of a pool</td>
</tr>
</tbody>
</table>
Examples

Request 1
A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is not already a member of a pool.

Request URL
POST {BaseURI}/msr/pool/100000/member/MSISDN/380561234567

Request Content
None

Response 1
The request is successful, and the subscriber is added to the pool.

HTTP Status Code
204

Response Content
None.

Request 2
A request is made to add a subscriber to a pool. The subscriber exists, but the pool does not.

Request URL
POST {BaseURI}/msr/pool/100009/member/IMSI/184569547984229

Request Content
None

Response 2
The request fails because the pool does not exist.

HTTP Status Code
404

Response Content
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4061">errorText</error>

Request 3
A request is made to add a subscriber to a pool. The pool exists, but the subscriber does not.

Request URL
POST {BaseURI}/msr/pool/900000/member/NAI/mum@foo.com

Request Content
None

Response 3
The request fails because the subscriber does not exist.
HTTP Status Code
404

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

Request 4
A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is already a member of a pool.

Request URL
POST {BaseURI}/msr/pool/100000/member/AccountId/10404723525

Request Content
None

Response 4
The request fails because the subscriber is already a member of a pool.

HTTP Status Code
409

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

Request 5
A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is not a member of a pool. The pool already has the maximum number of members allowed.

Request URL
POST {BaseURI}/msr/pool/100000/member/MSISDN/15141234567

Request Content
None

Response 5
The request fails because the pool has the maximum number of members allowed.

HTTP Status Code
400

Response Content
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4060">errorText</error>
6.6.2 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`: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 999999999999999999999
- `subKeyName`: A key field within 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 70: Remove Member from Pool Response Status/Error Codes**

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>—</td>
<td>Subscriber successfully removed from pool</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4061</td>
<td>Specified pool does not exist</td>
</tr>
<tr>
<td>404</td>
<td>MSR4062</td>
<td>Subscriber is not a member of the given pool</td>
</tr>
</tbody>
</table>

**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>

6.6.3 Get Pool Members

Description
This operation gets the list of Subscriber members of a Pool by poolId.

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

Request URL
GET {BaseURI}/msr/pool/poolId/member

- poolId: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999

Request Content
None.

Response Content
A <members> element that contains a <member> element for every subscriber that is a member of the pool. The <member> element is optional. There can be zero, one or many <member> elements. It is only present if the pool has member subscribers. One instance is present for every subscriber that is a member of the pool. A <member> element contains details about a single subscriber, containing all known user identities for that
subscriber, one user identity per `<id>` element. There can be one or many `<id>` elements per `<member>` element.

```
<members>
  [  
    <member>
      <id><name>keyName1</name><value>keyValue1</value></id>
      [  
        <id><name>keyName2</name><value>keyValue2</value></id>
        :  
        <id><name>keyNameN</name><value>keyValueN</value></id>
      ]
    </member>
  ]
</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>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Pool exists, and membership returned OK</td>
</tr>
<tr>
<td>404</td>
<td>MSR4061</td>
<td>Specified pool does not exist</td>
</tr>
</tbody>
</table>
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 has no member subscribers.

Request URL
GET {BaseURI}/msr/pool/200000/member

Request Content
None

Response 2
The request is successful, and no member subscribers are returned.

HTTP Status Code
200

Response Content
<?xml version="1.0" encoding="UTF-8"?>
<members>
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>

6.6.4 Get PoolID

Description
This operation gets the PoolID related to a subscriber, based on the given user identity of the subscriber.

Prerequisites
A subscriber with 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="PoolID">poolId</field>
</pool>

- poolId: PoolID value of the pool. Numeric value, 1 to 22 digits in length
  Values: 1 to 9999999999999999999999
Table 72: Get PoolID Response Status/Error Codes

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>—</td>
<td>Subscriber pool membership successfully returned</td>
</tr>
<tr>
<td>404</td>
<td>MSR4001</td>
<td>Subscriber is not found</td>
</tr>
<tr>
<td>404</td>
<td>MSR4062</td>
<td>Subscriber is not a member of a pool</td>
</tr>
</tbody>
</table>

Examples

Request 1
A request is made to get the PoolID for a subscriber. The subscriber is a member of a pool.

Request URL
GET {BaseURI}/msr/sub/MSISDN/380561234567/pool

Request Content
None

Response 1
The request is successful, and the PoolID value was returned.

HTTP Status Code
200

Response Content
<?xml version="1.0" encoding="UTF-8"?><pool><field name="PoolID">100000</field></pool>

Request 2
A request is made to get the PoolID for a subscriber. The subscriber is not a member of a pool.

Request URL
GET {BaseURI}/msr/sub/NAI/joe@foo.com/pool

Request Content
None

Response 2
The request fails, because the subscriber is not a member of a pool.

HTTP Status Code
404

Response Content
<?xml version="1.0" encoding="UTF-8"?><error code="MSR4062">errorText</error>
APPENDIX A  REST INTERFACE SYSTEM VARIABLES

The REST interface has a set of system variables that affect its operation as it runs. REST interface variables (Table 73) can be set using the UDR GUI and can be changed at runtime to effect dynamic server reconfiguration.

Table 73: Bulk Import/Export variables

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REST Interface Port</strong></td>
<td>REST Interface TCP Listening Port.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> 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.</td>
</tr>
<tr>
<td></td>
<td>Default is 8787; Range is 0 to 65535</td>
</tr>
<tr>
<td><strong>REST Interface Idle Timeout</strong></td>
<td>The maximum time (in seconds) that an open REST connection remains active without a request being sent, before the connection is dropped.</td>
</tr>
<tr>
<td></td>
<td>Default is 1200; Range is 1 to 86400</td>
</tr>
<tr>
<td><strong>Maximum REST Connections</strong></td>
<td>Maximum number of simultaneous REST Interface client connections.</td>
</tr>
<tr>
<td></td>
<td>Default is 100; Range is 1 to 100</td>
</tr>
<tr>
<td><strong>Allow REST Connections</strong></td>
<td>Whether or not to allow incoming provisioning connections on the REST Interface.</td>
</tr>
<tr>
<td></td>
<td>Default is checked</td>
</tr>
<tr>
<td><strong>REST Secure Mode</strong></td>
<td>Whether the REST Interface operates in secure mode (using TLS), or unsecure mode (plain text).</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> Changes to the Secure Mode do not take effect until the udrprov process is restarted.</td>
</tr>
<tr>
<td></td>
<td>Default is Unsecure</td>
</tr>
<tr>
<td><strong>Transaction Durability Timeout</strong></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Default is 5; Range is 2 is 3600</td>
</tr>
<tr>
<td><strong>Compatibility Mode</strong></td>
<td>Indicates whether backwards compatibility is enabled.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> Change to Compatibility Mode may cause the existing provisioning connections to be dropped. Default is R10+</td>
</tr>
</tbody>
</table>
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+” then UDR functions 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 request/responses being different to the default UDR behavior. This appendix lists the different request/responses that enabling the option applies to.

B.1  Get Row Response Format

UDR returns a data row in the format it is defined/stored (either “Field Based” or “Element Based”). The legacy SPR returns a (Quota) data row in “Element Based” format, even though the Quota entity is “Element Based”.

When configured in legacy SPR mode, UDR returns the (Quota) data row in “Field Based” format, within the CDATA. For example:

```xml
<?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>
```

If more than one matching row is found, then multiple <quota> rows are returned.

For example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
    <![CDATA[
      <?xml version="1.0" encoding="UTF-8"?>
      <usage>
        ... (same fields and values)
      </usage>
    ]]>]
  </data>
</subscriber>
```
<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, make 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 2 for Non-technical issue

You are connected to a live agent who can assist you with My Oracle Support registration and provide Support Identifiers. Mention you are a Tekelec Customer new to My Oracle Support.

My Oracle Support is available 24 hours a day, 7 days a week, 365 days a year.
APPENDIX D  CUSTOMER TRAINING

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

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

To reach Oracle University:

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

For the appropriate country or region contact phone number for the rest of the world, visit Oracle University at www.oracle.com/education/contacts.
APPENDIX E    LOCATE PRODUCT DOCUMENTATION ON THE ORACLE TECHNOLOGY NETWORK SITE

Oracle 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 www.adobe.com.

2. Click Applications.
3. Select Apps A-Z.
4. Click Communications.
5. In the Network Session Delivery and Control Infrastructure section, click User Data Repository.
6. Click release number.
7. To download a file to your location, click the download icon.