Oracle® Communications
User Data Repository

REST Provisioning Interface Specification

Release 12.1

E68357-02

February 2017

ORACLE

Oracle Communications User Data Repository REST Provisioning Interface Specification, Release 12.1

E68357-02

CAUTION: Use only the Installation procedure included in the Install Kit.
Before installing any system, please access My Oracle Support (MOS) (https://support.oracle.com) and review any Technical Service Bulletins (TSBs) that relate to this procedure.

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

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

See more information on MOS in the Appendix section.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose and Scope

This document presents the REST Provisioning interface to be used by provisioning client applications to administer the Provisioning Database of the Oracle Communications User Data Repository (UDR) system. Through REST interfaces, an external provisioning system supplied and maintained by the network operator may add, change, or delete subscriber/pool information in the Oracle Communications User Data Repository database.

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

1.2 References

The following external document references capture the source material used to create this document.

[1] *IMS Sh interface; Signalling flows and message contents*, 3GPP TS 29.328, Release 11

[2] *Sh interface based on the Diameter protocol; Protocol details*, 3GPP TS 29.329, Release 11

[3] *User Data Convergence (UDC); Technical realization and information flows; Stage 2*, 3GPP TS 23.335, Release 11

[4] *SDM v9.3 Subscriber Provisioning Reference Manual,* 910-6870-001,Revision A, January 2014

1.3 Glossary

This section lists terms and acronyms specific to this document.

Table 1: Glossary

Acronym/Term	Definition
ACID	Atomic, Consistent, Isolatable, Durable
BLOB	Binary Large Object
CFG	Configuration Data – data for components and system identification and configuration
CPS	Customer Provisioning System
DP	Database Processor
FRS	Feature Requirements Specification
FTP	File Transfer Protocol
GUI	Graphical User Interface
IMSI	International Mobile Subscriber Identity, or IMSI [im-zee]
IP	Internet Protocol
KPI	Key Performance Indicator
MEAL	Measurements, Events, Alarms, and Logs
MP	Message Processor
MSISDN	Mobile Subscriber ISDN Number
NA	Not Applicable

NE	Network Element
NPA	Numbering Plan Area (Area Code)
OAMP	Operations, Administration, Maintenance, and Provisioning
NOAM&P	Network OAM&Provisioning
PCRF	Policy Charging & Rules Function
PS	Provisioning System
REST	Representational State Transfer
SDO	Subscriber Data Object
SEC	Subscriber Entity Configuration
SNMP	Simple Network Management Protocol
SOAM	System Operation, Administration, and Maintenance
SPR	Subscriber Profile Repository
TCP	Transmission Control Protocol
UTC	Coordinated Universal Time
VIP	Virtual IP
XML	Extensible Markup Language

2 System Architecture

2.1 Overview

Oracle Communications User Data Repository (UDR) performs the function of an SPR, which is a database system that acts as a single logical repository that stores subscriber data. The subscriber data that traditionally has been stored into the HSS /HLR/AuC, Application Servers, etc., is now stored in UDR as specified in 3GPP UDC information model [12]. UDR facilitates the share and the provisioning of user related data throughout services of 3GPP system. Several Applications Front Ends, such as: one or more PCRF/HSS/HLR/AuCFEs can be served by UDR.

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

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

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

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

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

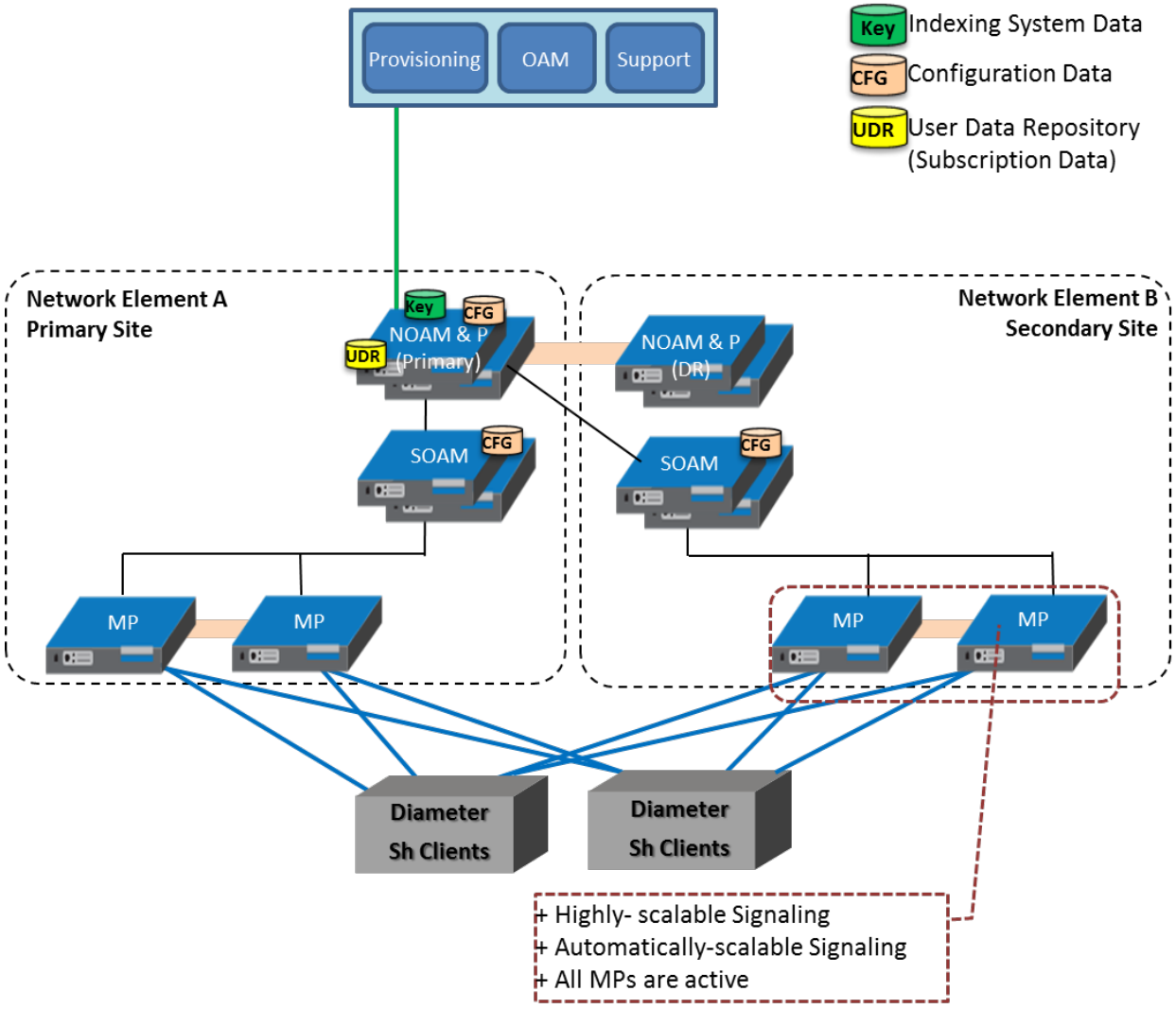
As the product evolves forward, the subscriber profiles in UDR can be expanded to support data associated with additional applications. Along with that, the MPs can be expanded to support additional Diameter interfaces associated with these applications. The IPFE can be integrated with the product to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

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

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

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

Figure 1: User Data Repository High Level Architecture



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
- Subscriber transparent data create/retrieve/modify/delete
  - Quota, State and Dynamic Quota
- Reset of a row within Subscriber Quota transparent 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 transparent data create/retrieve/modify/delete
  - Pool Quota, Pool State and Pool Dynamic Quota
- Reset of a row within Pool Quota transparent data
- 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 NOAM&P server. The RAS is responsible for:

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

2.4 Provisioning Clients

The RAS provides connections to 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 NOAM&P using the Primary NOAM&P's VIP.

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

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

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

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 currently 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 the server's certificate and public key are valid and has been signed by a trusted certificate authority (CA) that is known to the client application.
- **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 client's certificate and public key are valid and has been signed by a trusted certificate authority (CA) that is known to the server application.
- **An encrypted 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 upon which mode the RAS is configured to operate in (secure/unsecure), provisioning clients can connect using unsecure or secure connections to the RAS's well-known TCP/TLS listening port (configured using the `REST Secure Mode` configuration variable via UDR GUI).

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

2.5.2.1 TLS Certificates and Public/Private Key Pairs

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

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 should be copied to the server running the provisioning client.

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

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

Provisioning clients are required to send 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 TEKELEC Certification Authority (CA), it will not be able to establish a secure connection with the RAS.

2.5.2.2 Supported TLS Cipher Suites

A cipher suite is a set/combination of lower-level algorithms that a TLS-enabled connection uses to do authentication, key exchange, and stream encryption. 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 (currently those using 64 or 56 bit encryption algorithms but excluding export cipher suites), export-crippled ciphers (including 40 and 56 bits algorithms), or the MD5 hash algorithm are not supported due to their algorithms having known security vulnerabilities.

Table 3: TLS Supported Cipher Suites

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

2.6 Multiple Connections

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

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

The provisioning client establishes a new TCP/IP connection to RAS 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 new request comes before idle timeout.

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

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

2.7 Request Queue Management

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

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

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

2.8 Database Transactions

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

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

2.8.1.2 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

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

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

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

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

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 REST Provisioning Connections` (see Appendix A) controls whether REST interface connections are allowed to the configured port. If this variable is set to `NOT_ALLOWED`, then all existing connections are immediately dropped. Any attempts to connect are rejected.

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

2.9.2 Disable Provisioning

When the 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 that is sent will be rejected with a SERVICE_UNAVAILABLE error indicating the service is unavailable.

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

2.9.3 Idle Timeout

HTTP connection between Provisioning client and RAS is handled persistent fashion. The configuration variable `REST Interface Idle Timeout` (see Appendix A) indicates the time to wait before closing the connection due to inactivity (i.e. no requests are 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 will be raised indicating the impending behavior if the critical level is reached.

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

HTTP Status Code: `507`

Response Content:

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

2.11 Congestion Control

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

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

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

HTTP Status Code: `503`

Response Content:

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

3 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(S) interface that is designed based upon RESTful concepts. This section defines the operations that can be performed using the REST interface.

3.1 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.

3.1.1 HTTP(S) Request Headers

The following HTTP(S) requirements must be followed.

3.1.1.1 HTTP version

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

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

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

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

3.1.1.2 Accept Header

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

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

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

Operations in 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***.

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

3.1.1.3 Transfer-Encoding Header

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

```
Transfer-Encoding: chunked
```

3.1.1.4 Requests with body content

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

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

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

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

3.1.2 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 will include 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

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

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

Table 5: Error Codes

Error Code	Description
MSR4000	Invalid content request data supplied
MSR4001	Subscriber/pool not found
MSR4002	Subscriber/pool/data field is not defined
MSR4003	A key is detected to be already in the system for another subscriber/pool
MSR4004	Unique key not found for subscriber/pool
MSR4005	Field does not support multiple values and value for field already exists
MSR4049	Data type is not defined
MSR4050	Unknown key, the key provided in the request is invalid
MSR4051	The value provided for the field is invalid
MSR4053	Subscriber/pool exist, but the field value is incorrect
MSR4055	Subscriber is a member of a pool
MSR4056	Field is not updatable
MSR4057	Request only contains one field to update
MSR4058	Data type not found
MSR4059	Data row does not exist
MSR4060	Number of pool members exceeded
MSR4061	Specified pool does not exist
MSR4062	Subscriber is not a member of the pool
MSR4063	Entity cannot be reset
MSR4064	Occurrence constraint violation
MSR4065	Field is not set
MSR4066	Field value already exists
MSR4067	Multiple matching rows found
MSR4068	Free system memory is low
MSR4069	At least one key is required
MSR4070	Operation not allowed
MSR4097	Request rejected due to system congestion
MSR4098	Provisioning is disabled
MSR4099	Unexpected server error has occurred

Error Code	Description
MSR4100	Maximum number of Subscribers in a Basic Pool has been exceeded
MSR4101	Enterprise to Basic Pool Conversion failed threshold exceeded

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

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

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

4 REST Interface Message Definitions

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

4.1 Message Conventions

4.1.1 HTTP Method

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

4.1.2 Base URI

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

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

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

4.1.3 REST URL

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

4.1.3.1 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
```

4.1.3.2 Opaque Data Operations in URL

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

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

For example when deleting the Quota data for a subscriber :

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

4.1.3.3 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
```

4.1.3.4 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
```

4.1.3.5 Transparent Data Row Operations using an Instance Identifier in URL

For transparent data row based operations using an instance identifier the keywords ***data*** and ***row*** are used. The data type indicated in the URL can be any valid transparent data type which is row based. The data row name is also supplied. The instance identifier specified should be the unique identifier used to identify the transparent object.

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

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

4.1.3.6 Transparent Data Row Field Operations in URL

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

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

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

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

For transparent data row field based operations using an instance identifier the keyword ***data*** and ***row*** are used. The data type indicated in the URL can be any valid transparent data type which is row based. The data row name and field name are also supplied. The instance identifier specified should be the unique identifier used to identify the transparent object.

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

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

4.1.3.8 Transparent Data Field Operations in URL

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

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

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

4.1.4 URL Character Encoding

It is allowed to encode restricted characters in the URL using the % character, such as "%3B" for a ';' (semicolon) character, but it is not permitted to use double encoding such as "%253B" in order to first "quote" the '%' (percent) character.

4.2 Case Sensitivity

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

For example, the following is valid:

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

But the following is NOT:

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

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 will be returned in a response as "inputVolume"
- When a *fieldValue* is used to find a field (such as when using the "Delete Field Value" command), the field value ***is*** case-sensitive. If a multi-value field contained the values "DayPass,Weekend,Evening" and the Delete Field Value command was used to delete the value "WEEKEND", then this would fail
- When an attribute in the XML blob contains the row identifier name – aka *rowIdName* (for example for Quota, the element `<quota name="AggregateLimit">` contains the attribute called "name") the row identifier name is ***not*** case-sensitive
- When a *rowIdValue* is used to find a row (such as when using the "Get Row" command), the row identifier value ***is*** case-sensitive. If an entity contained a row called "DayPass", and the Get Row command was used to get the row "DAYPASS", then this would fail

- When a *keyValue* is specified in the URL (such as for an NAI), the value ***is*** case-sensitive. For example, for a subscriber with an NAI of "mum@foo.com", then "Mum@foo.com" or "MUM@FOO.COM" will ***not*** find the subscriber

4.3 XML Comments in a Request

A REST request may ***not*** contain XML comments within the request or the content body, such as :

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

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

HTTP Status Code: `400`

Response Content:

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

4.4 Request Content in a Request

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

Examples:

Request #1

Request URL: `GET {baseURI}/msr/sub/AccountId/10404723525`

Request Content:

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

Response #1

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

Request #2

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

Request Content:

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

Response #2

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

4.5 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 shown in colored column. Any blank/uncolored column represents unused parameter for corresponding command.

Table 6: Summary of Subscriber Commands

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

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

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 shown in colored column. Any blank/uncolored column represents unused parameter for corresponding command.

Table 7: Summary of Pool Commands

Operation Data	Command (Method)	URL	Main Object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	Instance Field Name	Instance Field Value	Field Name	Field Value	Additional Input
Pool Profile	Create Pool (POST)	{Base URL}/msr	pool	PoolID									Request Content

Operation Data	Command (Method)	URL	Main Object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	Instance Field Name	Instance Field Value	Field Name	Field Value	Additional Input
	Get Pool (GET)				{keyValue}								
	Update Pool (PUT)												Request Content
	Delete Pool (DELETE)												
Pool Profile Field	Add Field Value(POST)					field/ multipleFields					{fieldName}	{fieldValue}	
	Get Field (GET)												
	Get Field Value (GET)											{fieldValue}	
	Update Field (PUT)												
	Delete Field (DELETE)												
	Delete Field Value (DELETE)											{fieldValue}	
Pool Opaque Data	Set Opaque Data (PUT)					data	{opaqueDataType}						Request Content
	Get Opaque Data (GET)												
	Delete Opaque Data (DELETE)												
Pool Data Row	Set Row (PUT)							{rowIdValue}					Request Content
									{instance IdentifierField}	{instance IdentifierValue}			
	Get Row (GET)												
									{instance IdentifierField}	{instance IdentifierValue}			
	Delete Row (DELETE)												
									{instance IdentifierField}	{instance IdentifierValue}			
	Reset Pool Quota (POST)												
									{instance IdentifierField}	{instance IdentifierValue}			
Pool Data Row Field	Get Row Field (GET)										{fieldName}	{FieldValue}	
									{instance IdentifierField}	{instance IdentifierValue}			
	Get Row Field Value												

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

5 UDR Data Model

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

The subscriber-related data includes:

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

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

The pool-related data includes:

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

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

The Index contains information on the following:

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

The Subscription Data Object (SDO) :

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

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

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

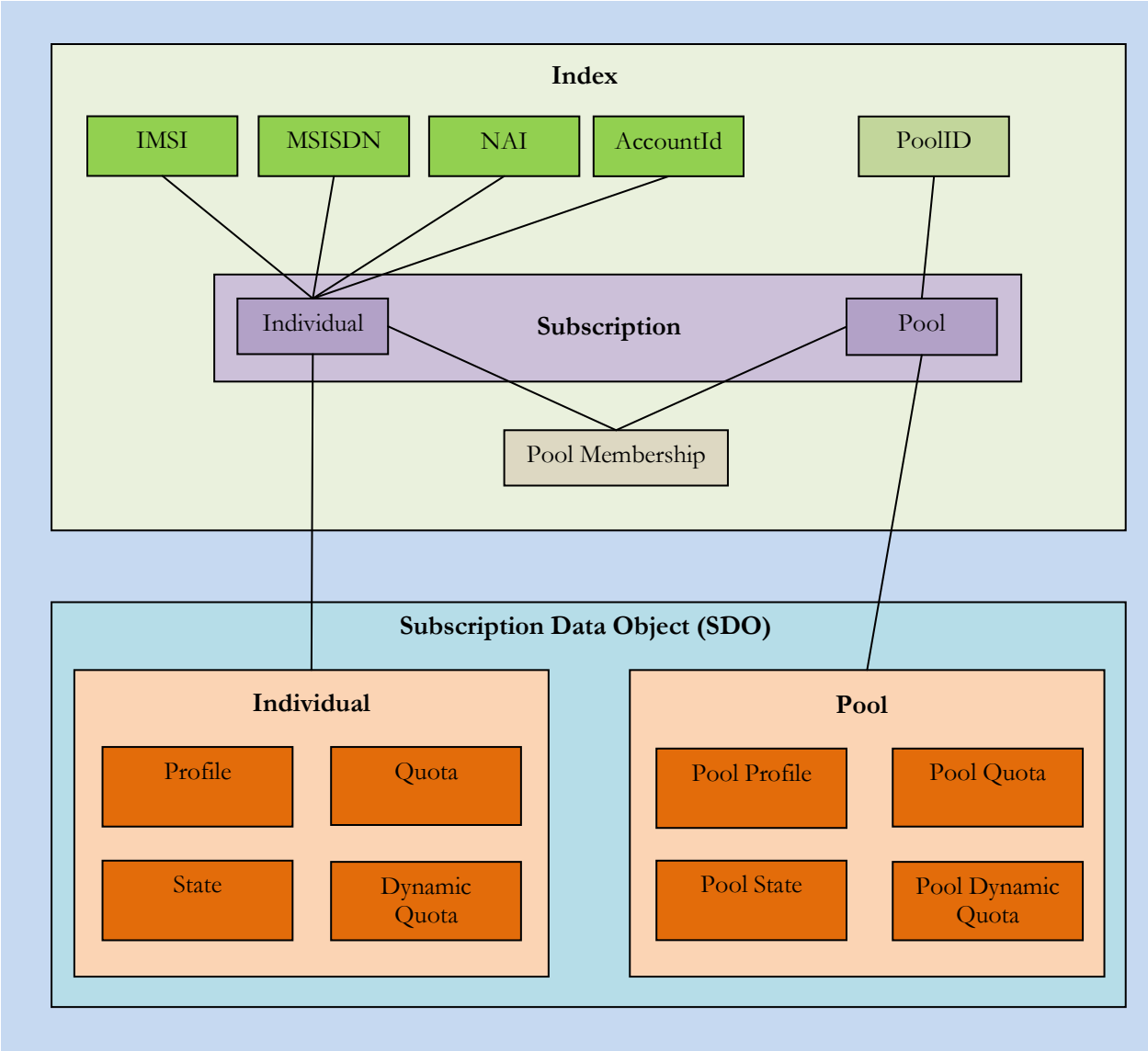
A field within an entity can have a default value. If an entity is created, and the field is not specified, it will be created with the default value.

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

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

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

Figure 2: Data Model



5.1 Subscriber Data

5.1.1 Subscriber Profile

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

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

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

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

Table 8: Subscriber Profile Entity Definition

Name (XML tag)	Type	Description
subscriber	**---**	**Sequence (multiplicity = 1)**
MSISDN	String	List of MSISDNs (8-15 numeric digits). A separate entry is included for each MSISDN associated with the subscriber's profile.
IMSI	String	List of IMSIs (10-15 numeric digits). A separate entry is included for each IMSI associated with the subscriber's profile.
NAI	String	List of NAIs (in format "user@domain","user", or "@domain"). A separate entry is included for each NAI associated with the subscriber's profile. The user or realm can be empty. Allowed characters for the user: '!', '%', '$', 'A'-'Z', 'a'-'z', '0'-'9', '.', '-', '_' Allowed characters for the host: 'A'-'Z', 'a'-'z', '0'-'9', '.', '-', '_' Example NAI Formats: bob, @privatecorp.example.net, fred$@example.com, eng.example.net!nancy@example.net , eng%nancy@example.net
AccountId	String	Any string that can be used to identify the account for the subscriber (1-255 characters). Allowed values are any ASCII printable character, values x20 to x7e.
BillingDay	String	Allowed values are [0-31]. The day of the month [1-31] on which the subscriber's associated quota should be reset. [0] indicates that the default value configured at the PCRF level should be used. This is automatically set in any record where BillingDay is not specified.
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the subscriber's profile.
Tier	String	Subscriber's tier.
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.
Custom3	String	Fields used to store customer-specific data.
Custom4	String	Fields used to store customer-specific data.
Custom5	String	Fields used to store customer-specific data.
Custom6	String	Fields used to store customer-specific data.
Custom7	String	Fields used to store customer-specific data.
Custom8	String	Fields used to store customer-specific data.
Custom9	String	Fields used to store customer-specific data.
Custom10	String	Fields used to store customer-specific data.
Custom11	String	Fields used to store customer-specific data.
Custom12	String	Fields used to store customer-specific data.
Custom13	String	Fields used to store customer-specific data.
Custom14	String	Fields used to store customer-specific data.
Custom15	String	Fields used to store customer-specific data.
Custom16	String	Fields used to store customer-specific data.
Custom17	String	Fields used to store customer-specific data.
Custom18	String	Fields used to store customer-specific data.
Custom19	String	Fields used to store customer-specific data.

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

5.1.2 Quota

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

The Quota entity shall be associated with a subscriber record and supports the following sequence of attributes.

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

Note: Quota Entity is generally created, updated and deleted by Oracle Communications Policy Management, hence it is advised that operations on Quota entity via Provisioning Interfaces such as SOAP or REST should be avoided.

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

- When a Quota instance is created, and no value is supplied for the field. In this case, the field is created with the value indicated
- When a Quota instance is reset using the "Reset Quota" command. If a field is defined as resettable, and the field currently exists, then it is set to the value indicated. If the field does not currently exist in the Quota, it is ***not*** created.
    - **Note**: If a resettable field does not exist, ***and*** the field is also defined as defaultable, then the field will get created with the value indicated

Table 9: Quota Entity Definition

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

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

5.1.3 State

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

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

Note: The default fields configured are not :

- Resettable
- Defaultable

The State entity shall support the following sequence of attributes:

Table 10: State Entity Definition

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

5.1.4 Dynamic Quota

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

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

Note: The default fields configured are not :

- Resettable
- Defaultable

The DynamicQuota entity shall support the following sequence of attributes:

Table 11: Dynamic Quota Entity Definition

Name (XML tag)	Type	Description
definition	**---**	**Sequence (multiplicity = 1)**
version	String	Version of the schema.
DynamicQuota	**---**	**Sequence (multiplicity = N)**
Type	String	Identifies the dynamic quota type.
name	String	The class identifier for a pass or top-up. This name will be used to match top-ups to quota definitions on the PCRF. This name will be used in policy conditions and actions on the PCRF.
InstanceId	String	A unique identifier to identify this instance of a dynamic quota object.
Priority	String	An integer represented as a string. This number allows service providers to specify when one pass or top-up should be used before another pass or top-up.
InitialTime	String	An integer represented as a string. The number of seconds initially granted for the pass/top-up.
InitialTotalVolume	String	An integer represented as a string. The number of bytes of total volume initially granted for the pass/top-up.
InitialInputVolume	String	An integer represented as a string. The number of bytes of input volume initially granted for the pass/top-up.
InitialOutputVolume	String	An integer represented as a string. The number of bytes of output volume initially granted for the pass/top-up.
InitialServiceSpecific	String	An integer represented as a string. The number of service specific units initially granted for the pass/top-up.
activationdatetime	String	The date/time after which the pass or top-up may be active. See section 5.3 for format details.
expirationdatetime	String	The date/time after which the pass or top-up is considered to be exhausted See section 5.3 for format details.
purchasedatetime	String	The date/time when a pass was purchased See section 5.3 for format details.
Duration	String	The number of seconds after first use in which the pass must be used or expired. If both Duration and expirationdatetime are present, the closest expiration time is used.
InterimReportingInterval	String	The number of seconds after which the GGSN/DPI/Gateway should revalidate quota grants with the PCRF.

5.2 Pool Data

5.2.1 Pool Profile

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

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

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

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

Table 12: Pool Profile Entity Definition

Name (XML tag)	Type	Description
pool	**---**	**Sequence (multiplicity = 1)**
PoolID	String	Pool identifier (1-22 numeric digits, minimum value 1).
BillingDay	Uint8	The day of the month [1-31] on which the pool's associated quota should be reset. [0] indicates that the default value configured at the PCRF level should be used.
BillingType	String	The billing frequency, monthly, weekly, daily.
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the pool's profile.
Tier	String	Pool's tier.
Type	String	Field used to identify an Enterprise Pool. Allowed value is "enterprise" and is ***not*** case-sensitive
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.
Custom3	String	Fields used to store customer-specific data.
Custom4	String	Fields used to store customer-specific data.
Custom5	String	Fields used to store customer-specific data.
Custom6	String	Fields used to store customer-specific data.
Custom7	String	Fields used to store customer-specific data.
Custom8	String	Fields used to store customer-specific data.
Custom9	String	Fields used to store customer-specific data.
Custom10	String	Fields used to store customer-specific data.
Custom11	String	Fields used to store customer-specific data.
Custom12	String	Fields used to store customer-specific data.
Custom13	String	Fields used to store customer-specific data.
Custom14	String	Fields used to store customer-specific data.
Custom15	String	Fields used to store customer-specific data.
Custom16	String	Fields used to store customer-specific data.
Custom17	String	Fields used to store customer-specific data.
Custom18	String	Fields used to store customer-specific data.
Custom19	String	Fields used to store customer-specific data.
Custom20	String	Fields used to store customer-specific data.

5.2.2 Pool Quota

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

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

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

Note: Pool Quota Entity is generally created, updated and deleted by Oracle Communications Policy Management, hence it is advised that operations on Pool Quota entity via Provisioning Interfaces such as SOAP or REST should be avoided.

5.2.3 Pool State

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

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

Note: The default fields configured are not :

- Resettable
- Defaultable

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

5.2.4 Pool Dynamic Quota

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

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

Note: The default fields configured are not :

- Resettable
- Defaultable

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

5.3 Date/Timestamp Format

The Date/Timestamp format used by many fields is :

CCYY-*MM*-*DD*T*hh*:*mm*:*ss*[<Z|<+|->*hh*:*mm*>]

This corresponds to either:

1. *CCYY*-*MM*-*DD*T*hh*:*mm*:*ss* (local time)
2. *CCYY*-*MM*-*DD*T*hh*:*mm*:*ss*Z (UTC time)
3. *CCYY*-*MM*-*DD*T*hh*:*mm*:*ss*+*hh*:*mm* (positive offset from UTC)
4. *CCYY*-*MM*-*DD*T*hh*:*mm*:*ss*-*hh*:*mm* (negative offset from UTC)

where:

- CC = century

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

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

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

6 Subscriber Provisioning

6.1 Subscriber Profile Commands

Table 13: Summary of Subscriber Profile Commands

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

6.1.1 Create Subscriber

Description

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

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

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

Prerequisites

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

Request URL

`POST {baseURI}/msr/sub`

Request Content

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

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

- ***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*

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

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

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content:

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

Response #1

The request is successful, and the subscriber was created.

HTTP Status Code: `201`

Response Content: None

Request #2

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

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

Request Content:

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

```
<subscriber>
  <field name="MSISDN">5141234567</field>
  <field name="IMSI">184126781623863</field>
  <field name="BillingDay">2</field>
  <field name="Location">Montreal</field>
</subscriber>
```

Response #2

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

HTTP Status Code: `404`

Response Content:

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

Request #3

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

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

Request Content:

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

Response #3

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

HTTP Status Code: `400`

Response Content:

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

Request #4

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

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

Request Content:

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

Response #4

The request fails because no key values were supplied.

HTTP Status Code: `400`

Response Content:

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

Request #5

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

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <field name="MSISDN">33123654862</field>
  <field name="IMSI">184569547984229</field>
  <field name="BillingDay">1</field>
  <field name="Custom15">allocate</field>
</subscriber>
```

Response #5

The request fails, because provisioning has been disabled.

HTTP Status Code: `503`

Response Content:

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

6.1.2 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*

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

Response Status/Error Codes

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

Examples

Request #1

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

Request URL: `GET {baseURI}/msr/sub/AccountId/10404723525`

Request Content: None

Response #1

The request is successful, and the subscriber was retrieved.

HTTP Status Code: `200`

Response Content:

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

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `404`

Response Content:

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

6.1.3 Update Profile

Description

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

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

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

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

Prerequisites

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

Request URL

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="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*

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

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

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	The subscriber data was replaced successfully

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

Examples

Request #1

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

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

Request Content:

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

Response #1

The request is successful, and the subscriber was updated.

HTTP Status Code: `204`

Response Content: None

6.1.4 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 will fail.

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.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

The request is successful.

HTTP Status Code: `204`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `409`

Response Content:

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

6.2 Subscriber Profile Field Commands

Table 14: Summary of Subscriber Profile Field Commands

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

6.2.1 Add Field Value

Description

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

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

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

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

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

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

Prerequisites

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

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

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

Request URL

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

- ***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 will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Successfully added field values
400	MSR4005	Field does not support multiple values
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4066	Field value already exists
404	MSR4001	Subscriber is not found

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `200`

Response Content: None

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: `400`

Response Content:

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

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/14161112222;14505556666`

Request Content: None

Response #4

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

HTTP Status Code: `200`

Response Content: None

6.2.2 Get Field

Description

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

Prerequisites

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

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

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 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*

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

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

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

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `200`

Response Content:

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

Request #3

A request is made to get the *Custom11* field for a subscriber. The field is valid, but is not set for the subscriber.

Request URL: `GET {baseURI}/msr/sub/MSISDN/33123654862/field/Custom11`

Request Content: None

Response #3

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

HTTP Status Code: `404`

Response Content:

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

6.2.3 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.

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

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

Prerequisites

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

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

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

Request URL

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

- ***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 will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

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

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

- ***fieldNameX***: The requested user defined field within the subscriber Profile
- ***fieldValueX***: Corresponding field value assigned to *fieldNameX*

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

Response Status/Error Codes

HTTP Status Code	Error Code	Description

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

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

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `200`

Response Content:

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

6.2.4 Update Field

Description

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

This operation replaces ("sets") the value of the field, which means that any existing values for the field are deleted first. For multi-value fields, all previous values are erased and the new set specified here 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
  - **Note**: a field name cannot be for a key value – i.e. IMSI, MSISDN, NAI, or AccountId
- ***fieldValue***: Corresponding field value assigned to *fieldName*
  - **Note**: for multi-value fields, the value will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Field(s) were successfully updated
400	MSR4051	The value provided for the field is invalid
400	MSR4056	Field is not updatable

HTTP Status Code	Error Code	Description
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `201`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `201`

Response Content: None

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: `201`

Response Content: None

Request #4

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

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

Request Content: None

Response #4

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

HTTP Status Code: `201`

Response Content: None

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

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

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

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

Prerequisites

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

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

Request URL

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

- ***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 – i.e. IMSI, MSISDN, NAI, or AccountId
- ***fieldValueX***: Corresponding field value assigned to *fieldNameX*
  - **Note**: for multi-value fields, the value will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

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

6.2.6 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 removal of the entire field from the subscriber Profile. I.e. the field is not present, not just the value is empty.

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

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

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

Prerequisites

A subscriber 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.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

The request is successful, and the field was deleted.

HTTP Status Code: `204`

Response Content: None

Request #2

A request is made to delete the *IMSI* key field. The subscriber currently 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 currently 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 currently 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 now does not have any MSISDN values, and just has an IMSI

HTTP Status Code: `204`

Response Content: None

6.2.7 Delete Field Value

Description

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

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

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

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

Note: The *fieldValue* is case-sensitive. An attempt to 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 – i.e. IMSI, MSISDN, NAI, or AccountId
- ***fieldValue***: Corresponding field value assigned to *fieldName*
  - **Note**: for multi-value fields, the value will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested field(s) were successfully deleted
400	MSR4005	Field does not support multiple values
400	MSR4056	Field is not updatable
400	MSR4069	At least one key is required
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `400`

Response Content:

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

Request #3

A request is made to delete the *MSISDN* fields with values of *14161112222* and *15141234567*. The subscriber currently 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 now has a single MSISDN of 15145556666.

HTTP Status Code: `204`

Response Content: None

6.3 Subscriber Opaque Data Commands

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

Table 15: Summary of Subscriber Opaque Data Commands

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

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 subscriber identified by the *keyName* and *keyValue* in the request.

The opaque data is provided in the request content.

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

Prerequisites

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

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

Request URL

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

- ***keyName***: A key field 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 version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="opaqueDataType">
<![CDATA[
cdataFieldValue
]]>
  </data>
</subscriber>
```

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

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

Response Content

None.

Response Status/Error Codes

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

HTTP Status Code	Error Code	Description
404	MSR4049	Data type is not defined

Examples

Request #1

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

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>92233720368547758O7</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</subscriber>
```

Response #1

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

HTTP Status Code: `201`

Response Content: None

Request #2

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

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="state">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
```

```
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>expire</name>
    <value>2010-02-09T11:20:32</value>
  </property>
  <property>
    <name>approved</name>
    <value>yes</value>
  </property>
</state>
]]>
  </data>
</subscriber>
```

Response #2

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

HTTP Status Code: `201`

Response Content: None

6.3.2 Get Opaque Data

Description

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

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

Prerequisites

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

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

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

Request URL

`GET {baseURI}/msr/sub/`*`keyName/keyValue`*`/data/`***`opaqueDataType`**

- ***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

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"

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

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

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

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `200`

Response Content:

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

6.3.3 Delete Opaque Data

Description

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

Only one opaque data type can be deleted per request.

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

Prerequisites

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

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

Request URL

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

- ***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.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `204`

Response Content: None

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: `204`

Response Content: None

6.4 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*.

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

Table 16: Summary of Subscriber Data Row Commands

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

6.4.1 Set Row

Description

This operation creates a new or updates an existing data row for the subscriber identified by the *keyName* and *keyValue*.
The data row identifier field value is specified in *rowIdValue*. All fieldNameX *fields* specified are set within the row.
If more than one existing row matches the requested *rowIdValue*, then the update request will fail.
If the specified row does not exist, it is created. If the row does exist, it is updated/replaced.

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

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

Prerequisites

A subscriber with the key of the *keyName*/*keyValue* supplied must exist.
The *transparentDataType* must reference a valid transparent Entity in the Interface Entity Map table in the SEC.

Request URL

Without Instance Identifier

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

With Instance Identifier

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

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

Request Content

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

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

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

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
```

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

Response #1

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

HTTP Status Code: `201`

Response Content: None

Request #2

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

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Q1">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
```

Response #2

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

HTTP Status Code: `201`

Response Content: None

Request #3

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

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Weekday">
    <cid>9223372036854775807</cid>
    <time>3422</time>
```

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

Response #3

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

HTTP Status Code: `400`

Response Content:

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

Request #4

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

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

Request Content:

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

Response #4

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

HTTP Status Code: `201`

Response Content: None

Request #5

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

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="AggregateLimit">
    <Type>pass</Type>
    <InstanceId>15678</InstanceId>
    <Priority>4</Priority>
    <InitialTime>135</InitialTime>
    <InitialTotalVolume>2000</InitialTotalVolume>
    <InitialInputVolume>1500</InitialInputVolume>
    <InitialOutputVolume>500</InitialOutputVolume>
    <InitialServiceSpecific>4</InitialServiceSpecific>
    <activationdatetime>2015-05-22T00:00:00-05:00</activationdatetime>
    <expirationdatetime>2015-05-29T00:00:00-05:00</expirationdatetime>
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
</definition>
```

Response #5

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

HTTP Status Code: `201`

Response Content: None

6.4.2 Get Row

Description

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

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

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

Note: The *rowIdValue* is case-sensitive. If a row 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

Without Instance Identifier

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

With Instance Identifier

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

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

Request Content

None.

Response Content

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

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

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

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested data row exists for subscriber

HTTP Status Code	Error Code	Description
404	MSR4001	Subscriber is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Q1">
    <cid>9223372036854775807</cid>
    <time>1</time>
    <totalVolume>0</totalVolume>
    <inputVolume>0</inputVolume>
    <outputVolume>0</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-12T16:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</subscriber>
```

Request #2

A request is made to get the *Weekend* data row from the *quota* transparent data for a subscriber. The subscriber has the Quota entity, but 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>
  <quota name="Weekend">
    <cid>7682364872564782343</cid>
    <time>32</time>
    <totalVolume>250</totalVolume>
    <inputVolume>4570</inputVolume>
    <outputVolume>11230</outputVolume>
    <serviceSpecific>29</serviceSpecific>
    <nextResetTime>2010-06-01T16:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</subscriber>
```

Request #4

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

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

Request Content: None

Response #4

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

HTTP Status Code: `200`

Response Content:

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

6.4.3 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.

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

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

Prerequisites

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

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

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

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `204`

Response Content: None

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: `404`

Response Content:

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

Request #4

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

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

Request Content: None

Response #4

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

HTTP Status Code: `204`

Response Content: None

Request #5

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

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

Request Content: None

Response #5

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

HTTP Status Code: `204`

Response Content: None

6.5 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*.

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

Table 17: Summary of Subscriber Data Row Field Commands

Command	Description	Key(s)	Command Syntax
Get Row Field	Retrieve value(s) for the specified field	(MSISDN, IMSI, NAI or	**`GET {baseURI}/msr/sub/`***`keyName/keyValue`*`/data/`*`transparentDataType/rowIdValue/fieldName`*

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

6.5.1 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 will be returned.

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

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

Prerequisites

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

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

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

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- ***keyName***: A key field 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
  - Value is `dynamicquota` for the DynamicQuota transparent data
- ***rowIdValue***: The row name value that identifies the row within the transparent data blob
- ***instanceFieldName***: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` or `Type` for the Quota transparent data
  - Value is `InstanceId` or `Type` for the DynamicQuota transparent data
- ***instanceFieldValue***: Corresponding field value assigned to *instanceFieldName*
- ***fieldName***: A user defined field 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 version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="transparentDataType">
<![CDATA[
cdataRowFieldValue
]]>
  </data>
</subscriber>
```

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

Response Status/Error Codes

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

Examples

Request #1

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

Request URL: `GET {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/inputVolume`

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Q1">
    <inputVolume>980</inputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Request #2

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

Request URL: `GET {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday/outputVolume`

Request Content: None

Response #2

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

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Weekday">
    <inputVolume>980</outputVolume>
  </quota>
  <quota name="Weekday">
    <inputVolume>2140</outputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: 200

Response Content:

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

6.5.2 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 will be returned.

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

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

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

Prerequisites

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

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

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

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

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- ***keyName***: A key field 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
  - Value is `dynamicquota` for the DynamicQuota transparent data
- ***rowIdValue***: The row name value that identifies the row within the transparent data blob
- ***instanceFieldName***: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` or `Type` for the Quota transparent data
  - Value is `InstanceId` or `Type` for the DynamicQuota transparent data
- ***instanceFieldValue***: Corresponding field value assigned to *instanceFieldName*
- ***fieldName***: A user defined field 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
  - Value is `dynamicquota` for the DynamicQuota transparent data
- *cdataRowFieldValue*: Contents of the XML data "blob", with the field from the row data

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

Response Status/Error Codes

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

Examples

Request #1

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

Request URL: `GET {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/inputVolume/980`

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Q1">
    <inputVolume>980</inputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Request #2

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

Request URL: `GET {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/outputVolume/2000`

Request Content: None

Response #2

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

HTTP Status Code: `400`

Response Content:

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

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Weekday">
    <inputVolume>3220</inputVolume>
  </quota>
  <quota name="Weekday">
    <inputVolume>3220</inputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Request #4

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

Request URL: GET
{BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Weekday/inputVolume/980

Request Content: None

Response #4

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

HTTP Status Code: 200

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="quota">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Weekday">
    <inputVolume>980</inputVolume>
  </quota>
</usage>
]]>
</data>
</subscriber>
```

Request #5

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

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

Request Content: None

Response #5

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

HTTP Status Code: `200`

Response Content:

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

6.5.3 Update Row Field

Description

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

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

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

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

Note: The *rowIdValue* is case-sensitive. If a row 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

Without Instance Identifier

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

With Instance Identifier

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

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

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

HTTP Status Code	Error Code	Description
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples

Request #1

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

Request URL: `PUT {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/inputVolume/0`

Request Content: None

Response #1

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

HTTP Status Code: `201`

Response Content: None

Request #2

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

Request URL: `PUT {BaseURI}/msr/sub/MSISDN/33123654862/data/quota/Q1/cid/45678`

Request Content: None

Response #2

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

HTTP Status Code: `400`

Response Content:

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

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: `400`

Response Content:

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

Request #4

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

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

Request Content: None

Response #4

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

HTTP Status Code: `201`

Response Content: None

6.5.4 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 will fail.

Note: If the specified row does not exist, the request will fail. If the specified row exists, but the field does not exist, this is not treated as an error, and no row/field data is deleted.

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

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

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

Prerequisites

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

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

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

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

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

Request Content

None.

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested transparent data row field was successfully deleted
400	MSR4056	Field is not updatable
400	MSR4067	Multiple matching rows found
400	MSR4064	Occurrence constraint violation
404	MSR4001	Subscriber is not found

HTTP Status Code	Error Code	Description
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `400`

Response Content:

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

Request #3

A request is made to delete the *InitialTotalVolume* field in the D*Q1* data row of the *dynamicquota* transparent data for a subscriber.

Request URL: DELETE
{BaseURI}/msr/sub/MSISDN/33123654862/data/dynamicquota/DQ1/InitialTotalVolume

Request Content: None

Response #3

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

HTTP Status Code: 204

Response Content: None

Request #4

A request is made to delete the *InitialTotalVolume* field in the D*Q1* data row of the *dynamicquota* transparent data for a subscriber with a *Type* value of *pass*.

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

Request Content: None

Response #4

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

HTTP Status Code: 204

Response Content: None

6.6 Subscriber Data Field Commands

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

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

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

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

Table 18: Summary of Subscriber Data Field Commands

Command	Description	Key(s)	Command Syntax

Command	Description	Key(s)	Command Syntax
Set Data Field	Create/update data field in transparent data of the specified type.	(MSISDN, IMSI, NAI or AccountId) and Field Name	**POST {baseURI}/msr/sub/***keyName/keyValue/* **data/***transparentDataType/* fieldName/fieldValue
			PUT {baseURI}/msr/sub/*keyName/keyValue/* **data/***transparentDataType/* fieldName/fieldValue
Get Data Field	Retrieve data field from transparent data of the specified type.		**GET {baseURI}/msr/sub/***keyName/keyValue/* **data/***transparentDataType/*fieldName
Delete Data Field	Delete data field within transparent data of the specified type		**DELETE {baseURI}/msr/sub/***keyName/keyValue/* **data/***transparentDataType/*fieldName

6.6.1 Set Data Field

Description

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

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

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

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

Note: The *fieldName* is NOT case-sensitive. If a field already existed called "mcc", then an attempt to update an existing field called "MCC" would be successful.

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

Prerequisites

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

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

Request URL

Format 1

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

Format 2

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

- *keyName*: A key field 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 `state` for the State transparent data
- *fieldName*: A user defined field within the transparent data
  - For the State entity, this corresponds to a 'property' within the entity
    - The `fieldName` is stored exactly as it is sent in the request. The case of `fieldName` would change if an update is done using a different case.
- *fieldValue*: Corresponding field value assigned to *fieldName*.

Request Content

None**.**

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Data field was successfully created/updated
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4067	Multiple matching fields found
404	MSR4001	Subscriber is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `201`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `201`

Response Content: None

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: `201`

Response Content: None

Request #4

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

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

Request Content: None

Response #4

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

HTTP Status Code: `400`

Response Content:

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

6.6.2 Get Data Field

Description

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

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

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

The transparent data field is specified in *fieldName*.

Note: The *fieldName* is NOT case-sensitive. If a field existed called "mcc", then an attempt to get a field called "MCC" would be successful.

Prerequisites

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

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

A field within the transparent data should exist for the subscriber.

Request URL

`GET {baseURI}/msr/sub/`*`keyName/keyValue`*`/data/`***`transparentDataType/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 `state` for the State transparent data
- ***fieldName***: A user defined field within the transparent data
  - For the State entity this corresponds to a 'property' within the entity

Request Content

None.

Response Content

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

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

- ***transparentDataType***: A user defined type/name for the transparent data

  - Value is `state` for the State transparent data
- *cdataFieldValue*: Contents of the XML data "blob", with the field from the transparent data

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

The request is successful, and the requested property is returned

HTTP Status Code: `200`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="state">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
</state>
]]>
</data>
</subscriber>
```

Request #2

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

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

Request Content: None

Response #2

The request fails, because the requested property does not exist.

HTTP Status Code: `404`

Response Content:

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

Request #3

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

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

Request Content: None

Response #3

The request is successful, and the both the properties are returned

HTTP Status Code: `200`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<data name="state">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>mcc</name>
    <value>315</value>
  </property>
  <property>
    <name>mcc</name>
    <value>400</value>
  </property>
</state>
]]>
</data>
</subscriber>
```

6.6.3 Delete Data Field

Description

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

The field identifier is specified in *fieldName*.

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

Note: If the specified field does not exist, this is not considered an error and a successful result will be returned.

Note: The *fieldName* is NOT case-sensitive. If a field existed called "mcc", then an attempt to delete a field called "MCC" would be successful.

Prerequisites

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

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

Request URL

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

- ***keyName***: A key field 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 `state` for the State transparent data
- ***fieldName***: A user defined field within the transparent data
  - For the State entity this corresponds to a 'property' within the entity

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

A request is made to delete the *mcc property* in the *state* transparent data. The *mcc* property exists in the State data.

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

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `204`

Response Content: None

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: `404`

Response Content:

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

6.7 Subscriber Special Operation Commands

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

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

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

Table 19: Summary of Subscriber Special Operation Commands

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

6.7.1 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 will fail.

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

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

Note: 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 currently exist, it is *not* created. But, if a resettable field does not exist, and the field has a default value, then the field will then get created with the default value.

Prerequisites

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

The Quota transparent data must exist for the subscriber.

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- ***keyName***: A key field 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
- ***instanceFieldName***: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` for the Quota transparent data
- ***instanceFieldValue***: Corresponding field value assigned to *instanceFieldName*

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `404`

Response Content:

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

Request #3

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

Request URL: `POST {baseURI}/msr/sub/MSISDN/33123654862/data/quota/Q6`

Request Content: None

Response #3

The request fails, because the *Q6* row does not exist.

HTTP Status Code: `404`

Response Content:

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

Request #4

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

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

Request Content: None

Response #4

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

HTTP Status Code: `400`

Response Content:

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

Request #5

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

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

Request Content: None

Response #5

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

HTTP Status Code: `204`

Request Content: None

7 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 20: Summary of Pool Profile Commands

Command	Description	Key(s)	Command Syntax
Create Pool	Create a new pool/pool Profile	-	**POST {baseURI}/msr/pool**
Get Pool	Get pool Profile data	PoolID	**GET {baseURI}/msr/pool/*poolId***
Update Pool	Replace an existing pool Profile		**PUT {baseURI}/msr/pool/*poolId***
Delete Pool	Delete all pool Profile data and all opaque data associated with the pool		**DELETE {baseURI}/msr/pool/*poolId***

7.1.1 Create Pool

Description

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

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

Note: Multi-value fields can be specified by a single *fieldNameX* value with a delimited list of values, or 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, 1-22 digits in length

  - Values: `1-99999999999999999999`
- ***fieldNameX***: A user defined field within the pool Profile
- ***fieldValueX***: Corresponding field value assigned to *fieldNameX*

Note: PoolID/field order in the request is not important

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content:

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

Response #1

The request is successful, and the pool was created.

HTTP Status Code: `201`

Response Content: None

Request #2

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

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

Request Content:

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

Response #2

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

HTTP Status Code: `400`

Response Content:

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

7.1.2 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, 1-22 digits in length
  - Values: `1-99999999999999999999`

Request Content

None.

Response Content

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

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

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

Note: PoolID/field order in the request is not important

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

The request is successful, and the pool was retrieved.

HTTP Status Code: `200`

Response Content:

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

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

Request #2

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

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

Request Content: None

Response #2

The request fails, as the pool does not exist.

HTTP Status Code: `404`

Response Content:

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

7.1.3 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).

Note: If the PoolID is included in the content, and it's different from the value specified in the URL, the request will fail.

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, 1-22 digits in length
  - Values: `1-99999999999999999999`

Request Content

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

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

```
  <field name="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, 1-22 digits in length
  - Values: `1-9999999999999999999`
- *fieldNameX*: A user defined field within the pool Profile
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

Note: PoolID/field order in the request is not important

Response Content

None.

Response Status/Error Codes

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

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

7.1.4 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 will fail.

Request URL

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

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-9999999999999999999`

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

The request is successful.

HTTP Status Code: `204`

Response Content: None

Request #2

The pool with the given PoolID is deleted. The pool exists, 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>
```

7.2 Pool Profile Field Commands

Table 21: Summary of Pool Profile Field Commands

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

Command	Description	Key(s)	Command Syntax
Delete Field Value	Delete a value for the specified field		**DELETE {baseURI}/msr/pool/***poolId***/field/***fieldName***/***fieldValue***

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

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

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

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

Prerequisites

A 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, 1-22 digits in length
  - Values: 1-9999999999999999999
- ***fieldName***: A user defined field within the pool Profile
- ***fieldValue***: Corresponding field value assigned to *fieldName*
  - **Note**: for multi-value fields, the value will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

7.2.2 Get Field

Description

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

Note: Depending upon 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, 1-22 digits in length
  - Values: `1-99999999999999999999`
- ***fieldName***: A user defined field within the pool Profile

Request Content

None.

Response Content

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

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

- ***fieldName***: A user defined field within the pool Profile
- ***fieldValueX***: Corresponding field value assigned to *fieldName*

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested field exists for pool

HTTP Status Code	Error Code	Description
404	MSR4001	Pool is not found
404	MSR4002	Pool field is not defined
404	MSR4065	Field is not set

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

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

7.2.3 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.

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

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

Prerequisites

A pool with the PoolID 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, 1-22 digits in length
  - Values: 1-9999999999999999999999
- *fieldName*: A user defined field within the pool Profile
- *fieldValue*: Corresponding field value assigned to *fieldName*
  - **Note**: for multi-value fields, the value will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

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

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

- *fieldName*: A user defined field within the pool Profile
- *fieldValueX*: Corresponding field value assigned to *fieldName*

Response Status/Error Codes

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

Examples

Request #1

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

Request URL: `GET {BaseURI}/msr/pool/200000/field/Tier/Gold`

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

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

Request #2

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>
```

7.2.4 Update Field

Description

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

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

Note: This command cannot be used to update 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, 1-22 digits in length
  - Values: `1-99999999999999999999`
- ***fieldName***: A user defined field within the pool Profile
- ***fieldValue***: Corresponding field value assigned to *fieldName*
  - **Note**: for multi-value fields, the value will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: 201

Response Content: None

7.2.5 Update Multiple Fields

Description

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

This operation replaces ("sets") the value of the field, which means that any existing values for the field are deleted first. For multi-value fields, all previous values are erased and the new set specified here 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 will result in an error.

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

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

Prerequisites

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

The field(s) *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, 1-22 digits in length
  - Values: `1-99999999999999999999`
- ***fieldNameX***: A user defined field within the pool Profile
- ***fieldValueX***: Corresponding field value assigned to *fieldNameX*
  - **Note**: for multi-value fields, the value will contain a semicolon separated list of values on a single line. E.g. "a;b;c"
    - **Note**: The semicolon between the field values may need to be encoded as %3B for certain clients

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `201`

Response Content: None

7.2.6 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 entire field from the pool Profile. I.e. the field is not present, not just the value is empty.

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

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

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

Prerequisites

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

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

Request URL

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

- ***poolId***: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-9999999999999999999`
- ***fieldName***: A user defined field within the subscriber Profile

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

The request is successful, and the field was deleted.

HTTP Status Code: `204`

Response Content: None

Request #2

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

Request URL: `DELETE {BaseURI}/msr/pool/100000/field/Type`

Request Content: None

Response #2

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

HTTP Status Code: `400`

Response Content:

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

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

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

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

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

Prerequisites

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

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

Request URL

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

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

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `204`

Request Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `204`

Response Content: None

7.3 Pool Opaque Data Commands

Table 22: Summary of Pool Opaque Data Commands

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

7.3.1 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.

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

Prerequisites

A pool with the 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, 1-22 digits in length
  - Values: `1-99999999999999999999`
- *opaqueDataType*: A user defined type/name for the opaque data
  - Value is either `poolquota`, `poolstate`, or `pooldynamicquota`

Request Content

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

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

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

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

Response Content

None.

Response Status/Error Codes

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

Example

Request #1

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

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="poolquota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>92233720368547758O7</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</pool>
```

Response #1

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

HTTP Status Code: `201`

Response Content: None

Request #2

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

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

Request Content:

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

7.3.2 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 entire 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, 1-22 digits in length
  - Values: `1-9999999999999999999999`
- *opaqueDataType*: A user defined type/name for the opaque data
  - Value is either `poolquota`, `poolstate`, or `pooldynamicquota`

Request Content

None.

Response Content

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

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

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

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested opaque data exists for pool
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4053	Data type is not set for this pool

Example

Request #1

A request is made to get the *poolquota* opaque data for a pool.

Request URL: `GET {baseURI}/msr/pool/100001/data/poolquota`

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

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

Request #2

A request is made to get the *poolstate* opaque data for a pool.

Request URL: `GET {baseURI}/msr/pool/100004/data/poolstate`

Request Content: None

Response #2

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

HTTP Status Code: `200`

Response Content:

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

7.3.3 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.

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

Prerequisites

A pool with the 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, 1-22 digits in length
  - Values: `1-99999999999999999999`
- *opaqueDataType*: A user defined type/name for the opaque data
  - Value is either `poolquota`, `poolstate`, or `pooldynamicquota`

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Example

Request #1

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

Request URL: `DELETE {baseURI}/msr/pool/500005/data/pooldynamicquota`

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

A request is made to delete the *poolstate* opaque data. The PoolState opaque data is a valid entity, but the requested pool does not contain any PoolState opaque data.

Request URL: `DELETE {baseURI}/msr/pool/600006/data/poolstate`

Request Content: None

Response #2

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

HTTP Status Code: 204

Response Content: None

7.4 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*.

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

Table 23: Summary of Pool Data Row Commands

Command	Description	Key(s)	Command Syntax
Set Row	Create/update data row in data of the specified type.	PoolID and Row Identifier Or PoolID and Row Identifier and Instance Identifier	**PUT {baseURI}/msr/pool/***poolId***/data/** ***transparentDataType/rowIdValue***
	Create/update data row in data of the specified type and instance identifier.		**PUT {baseURI}/msr/pool/***poolId***/data/** ***transparentDataType/rowIdValue*****/row/** ***instanceFieldName/instanceFieldValue***
Get Row	Retrieve data row from data of the specified type.		**GET {baseURI}/msr/pool/***poolId***/data/** ***transparentDataType/rowIdValue***
	Retrieve data row from data of the specified type and instance identifier.		**GET {baseURI}/msr/pool/***poolId***/data/** ***transparentDataType/rowIdValue*****/row/** ***instanceFieldName/instanceFieldValue***
Delete Row	Delete data row within data of the specified type		**DELETE {baseURI}/msr/pool/***poolId***/data/** ***transparentDataType/rowIdValue***
	Delete data row within data of the specified type and instance identifier		**DELETE {baseURI}/msr/pool/***poolId***/data/** ***transparentDataType/rowIdValue*****/row/** ***instanceFieldName/instanceFieldValue***

7.4.1 Set Row

Description

This operation creates a new 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 will fail.

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

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

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

Without Instance Identifier

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

With Instance Identifier

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

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-99999999999999999999`
- *transparentDataType*: A user defined type/name for the transparent data
  - Value is `poolquota` for the PoolQuota transparent data
  - Value is `pooldynamicquotafor` the PoolDynamicQuota transparent data
- *rowIdValue*: The row name value that identifies the row within the data blob
- There is no rowIdValue for `State` transparent data
- *instanceFieldName*: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` or `Type` for the PoolQuota transparent data
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

Request Content

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

- *rowValue*: Contents of the XML data "blob", with the row data
  - **Note**: the *rowValue* is of the same format as an entire PoolQuota entity, just containing a single row, the row being added

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

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Data row was successfully created/updated
400	MSR4000	Request content is not valid
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4067	Multiple matching rows found
404	MSR4001	Pool is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined

Examples

Request #1

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

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

Request Content:

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

Response #1

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

HTTP Status Code: `201`

Response Content: None

Request #2

A request is made to update a data row in the *poolquota* transparent data for a pool. The data row identifier field value is *PQ1*. The pool has an existing PoolQuota row called *PQ1*.

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

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="PQ1">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
```

Response #2

The request is successful, and the data row *PQ1* was updated.

HTTP Status Code: `201`

Response Content: None

Request #3

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

Request URL: `PUT {baseURI}/msr/pool/10000/data/pooldynamicquota/AggregateLimit/row/InstanceId/15678`

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <version>1</version>
  <DynamicQuota name="AggregateLimit">
    <Type>pass</Type>
    <InstanceId>15678</InstanceId>
    <Priority>4</Priority>
    <InitialTime>135</InitialTime>
    <InitialTotalVolume>2000</InitialTotalVolume>
    <InitialInputVolume>1500</InitialInputVolume>
    <InitialOutputVolume>500</InitialOutputVolume>
    <InitialServiceSpecific>4</InitialServiceSpecific>
    <activationdatetime>2015-05-22T00:00:00-05:00</activationdatetime>
    <expirationdatetime>2015-05-29T00:00:00-05:00</expirationdatetime>
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
</definition>
```

Response #3

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

HTTP Status Code: `201`

Response Content: None

7.4.2 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*.

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

Prerequisites

A 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

Without Instance Identifier

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

With Instance Identifier

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

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

Request Content

None.

Response Content

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

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

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

Response Status/Error Codes

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

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>92233720368547758O7</cid>
    <time>1</time>
    <totalVolume>0</totalVolume>
    <inputVolume>0</inputVolume>
    <outputVolume>0</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-12T16:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</pool>
```

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `404`

Response Content:

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

Request #3

A request is made to get the *Weekday* data row from the *poolquota* transparent data for a pool. The pool has the PoolQuota entity. Two instances of the *Weekday* data row exist.

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

Request Content: None

Response #3

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

HTTP Status Code: `200`

Response Content:

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

```
<pool>
  <data name="poolquota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="Weekend">
    <cid>9223372036854775807</cid>
    <time>1</time>
    <totalVolume>0</totalVolume>
    <inputVolume>0</inputVolume>
    <outputVolume>0</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-12T16:00:00-05:00</nextResetTime>
  </quota>
  <quota name="Weekend">
    <cid>7682364872564782343</cid>
    <time>32</time>
    <totalVolume>250</totalVolume>
    <inputVolume>4570</inputVolume>
    <outputVolume>11230</outputVolume>
    <serviceSpecific>29</serviceSpecific>
    <nextResetTime>2010-06-01T16:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</pool>
```

Request #4

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

Request URL: `GET`

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

Request Content: None

Response #4

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

HTTP Status Code: `200`

Response Content:

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

```
    <Duration>10</Duration>
  </DynamicQuota>
  <DynamicQuota name="PDQ1">
    <Type>topup</Type>
    <InstanceId>11223344</InstanceId>
    <Priority>4</Priority>
    <InterimReportingInterval>200</InterimReportingInterval>
    <Duration>20</Duration>
  </DynamicQuota>
</definition>]]>
  </data>
</pool>
```

7.4.3 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.

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

Note: 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

Without Instance Identifier

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

With Instance Identifier

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

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-9999999999999999999`
- *transparentDataType*: A user defined type/name for the transparent data
  - Value is `poolquota` for the PoolQuota transparent data
  - Value is `pooldynamicquota` for the PoolDynamicQuota transparent data
- *rowIdValue*: The row name value that identifies the row within the transparent data blob
- *instanceFieldName*: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` or `Type` for the PoolQuota transparent data
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data

- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

Examples

Request #1

A request is made to delete the *PQ1* data row in the *poolquota* transparent data. The *PQ1* data row exists in the PoolQuota data.

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

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

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

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

Request Content: None

Response #2

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

HTTP Status Code: `204`

Response Content: None

Request #3

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

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

Request Content: None

Response #3

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

HTTP Status Code: `204`

Response Content: None

7.5 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*.

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

Table 24: Summary of Pool Data Row Field Commands

Command	Description	Key(s)	Command Syntax
Get Row Field	Retrieve value(s) for the specified field	PoolID and Row Identifier and Field name Or PoolID and Row Identifier, Instance Identifier and Field name	**GET {baseURI}/msr/pool/***poolId***/data/***transparentDataType***/***rowIdValue***/***fieldName***
	Retrieve value(s) for the specified field and instance identifier		**GET {baseURI}/msr/pool/***poolId***/data/***transparentDataType***/***rowIdValue***/row/**instanceFieldName/instanceFieldValue/***fieldName***
Get Row Field Value	Retrieve a single value for the specified field		**GET {baseURI}/msr/pool/***poolId***/data/***transparentDataType***/***rowIdValue***/***fieldName***/***fieldValue***
	Retrieve a single value for the specified field and instance identifier		**GET {baseURI}/msr/pool/***poolId***/data/***transparentDataType***/***rowIdValue***/row/**instanceFieldName/instanceFieldValue/***fieldName***/***fieldValue***

Command	Description	Key(s)	Command Syntax
Update Row Field	Update field to the specified value		**PUT {baseURI}/msr/pool/***poolId***/data/** *transparentDataType/rowIdValue/ fieldName/fieldValue*
	Update field to the specified value and instance identifier		**PUT {baseURI}/msr/pool/***poolId***/data/** *transparentDataType/rowIdValue/***row/** instanceFieldName/instanceFieldValue/ *fieldName/fieldValue*
Delete Row Field	Delete all values for the specified field		**DELETE {baseURI}/msr/pool/***poolId***/data/** *transparentDataType/rowIdValue/fieldName*
	Delete all values for the specified field and instance identifier		**DELETE {baseURI}/msr/pool/***poolId***/data/** *transparentDataType/rowIdValue/***row/** instanceFieldName/instanceFieldValue/ *fieldName*

7.5.1 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 will be returned.

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

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

Prerequisites

A 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

Without Instance Identifier

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

With Instance Identifier

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

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: 1-9999999999999999999

- *transparentDataType*: A user defined type/name for the transparent data
  - Value is `poolquota` for the PoolQuota transparent data
  - Value is `pooldynamicquota` for the PoolDynamicQuota transparent data
- *rowIdValue*: The row name value that identifies the row within the transparent data blob
- *instanceFieldName*: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` or `Type` for the PoolQuota transparent data
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*
- *fieldName*: A user defined field within the transparent data row

Request Content

None.

Response Content

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

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

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

Response Status/Error Codes

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

Examples

Request #1

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

Request URL: `GET {BaseURI}/msr/pool/100000/data/poolquota/PQ1/inputVolume`

Request Content: None

Response #1

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

HTTP Status Code: `200`

Response Content:

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

Request #2

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

Request URL: `GET {BaseURI}/msr/pool/100000/data/poolquota/Weekday/outputVolume`

Request Content: None

Response #2

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

HTTP Status Code: `200`

Response Content:

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

```
  </quota>
</usage>
]]>
</data>
</pool>
```

Request #3

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

Request URL: `GET {BaseURI}/msr/pool/10000/data/pooldynamicquota/PDQ1/row/InstanceId/11223344/InitialInputVolume`

Request Content: None

Response #3

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

HTTP Status Code: `200`

Response Content:

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

7.5.2 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 will be returned.

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

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

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

Prerequisites

A 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

Without Instance Identifier

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

With Instance Identifier

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

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

Request Content

None.

Response Content

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

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

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

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

Response Status/Error Codes

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

Examples

Request #1

A request is made to get the *inputVolume* field with the value of *980* in the *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>
```

Request #4

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

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

Request Content: None

Response #4

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

HTTP Status Code: `200`

Response Content:

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

7.5.3 Update Row Field

Description

This operation updates a field(s) 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 currently exist, it will be created.

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

Note: The *rowIdValue* is case-sensitive. If a row 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

Without Instance Identifier

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

With Instance Identifier

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

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-9999999999999999999`
- *transparentDataType*: A user defined type/name for the transparent data
  - Value is `poolquota` for the PoolQuota transparent data
  - Value is `pooldynamicquota` for the PoolDynamicQuota transparent data
- *rowIdValue*: The row name value that identifies the row within the transparent data blob
- *instanceFieldName*: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` or `Type` for the PoolQuota transparent data
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*
- *fieldName*: A user defined field within the transparent data row
- *fieldValue*: Corresponding field value assigned to *fieldName*

Request Content

None.

Response Content

None.

Response Status/Error Codes

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

HTTP Status Code	Error Code	Description
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples

Request #1

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

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

Request Content: None

Response #1

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

HTTP Status Code: `201`

Response Content: None

Request #2

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

Request URL: `PUT {BaseURI}/msr/pool/100000/data/poolquota/PQ1/cid/45678`

Request Content: None

Response #2

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

HTTP Status Code: `400`

Response Content:

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

Request #3

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

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

Request Content: None

Request #4

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

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

Request Content: None

Response #4

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

HTTP Status Code: `201`

Response Content: None

7.5.4 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 will fail.

Note: If the specified row does not exist, the request will fail. If the specified row exists, but the field does not exist, this is not treated as an error, and no row/field data is deleted.

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

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

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

Prerequisites

A 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

Without Instance Idenitifier

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

With Instance Identifier

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

- ***poolId***: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-999999999999999999999`
- ***transparentDataType***: A user defined type/name for the transparent data
  - Value is `poolquota` for the PoolQuota transparent data
  - Value is `pooldynamicquota` for the PoolDynamicQuota transparent data
- ***rowIdValue***: The row name value that identifies the row within the transparent data blob
- ***instanceFieldName***: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` or `Type` for the PoolQuota transparent data
  - Value is `InstanceId` or `Type` for the PoolDynamicQuota transparent data
- ***instanceFieldValue***: Corresponding field value assigned to *instanceFieldName*
- ***fieldName***: A user defined field within the transparent data row

Request Content

None.

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested transparent data row field was successfully deleted
400	MSR4056	Field is not updatable
400	MSR4067	Multiple matching rows found
400	MSR4064	Occurrence constraint violation
404	MSR4001	Pool is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

Examples

Request #1

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

Request URL: `DELETE {BaseURI}/msr/pool/100000/data/poolquota/PQ1/inputVolume`

Request Content: None

Response #1

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

HTTP Status Code: `204`

Response Content: None

Request #2

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

Request URL: `DELETE {BaseURI}/msr/pool/100000/data/poolquota/Weekday/inputVolume`

Request Content: None

Response #2

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

HTTP Status Code: `400`

Response Content:

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

Request #3

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

Request URL: `DELETE {BaseURI}/msr/pool/33123654862/data/pooldynamicquota/PDQ1/InitialTotalVolume`

Request Content: None

Response #3

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

HTTP Status Code: `204`

Response Content: None

7.6 Pool Data Field Commands

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

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

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

Note: Pool data field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque will result in an HTTP Status Code `400`, with an `MSR4070` error being returned.

Table 25: Summary of Pool Data Field Commands

Command	Description	Key(s)	Command Syntax
Set Data Field	Create/update data field in transparent data of the specified type.	PoolID and Field Name	**POST {baseURI}/msr/pool/***poolId***/data/***transparentDataType***/**fieldName/fieldValue
			PUT {baseURI}/msr/pool/*poolId***/data/***transparentDataType***/**fieldName/fieldValue
Get Data Field	Retrieve data field from transparent data of the specified type.		**GET {baseURI}/msr/pool/***poolId***/data/***transparentDataType***/**fieldName
Delete Data Field	Delete data field within transparent data of the specified type		**DELETE {baseURI}/msr/pool/***poolId***/data/***transparentDataType***/**fieldName

7.6.1 Set Data Field

Description

This operation creates a new or updates an existing field in a transparent data for the pool identified by the *poolId*.

The field name is specified in *fieldName*, and the field value is specified in *fieldValue*.

If more than one existing fields matches the requested *fieldName*, then the update request will fail.

If the specified field does not exist, it is created. If the field does exist, it is updated/replaced.

Note: The *fieldName* is NOT case-sensitive. If a field already existed called "mcc", then an attempt to update an existing field called "MCC" would be successful.

Note: If the transparent entity specified in *entityName* does not exist for the pool, it will be 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

Format 1

PUT {baseURI}/msr/pool/*poolId***/data/***transparentDataType/fieldName/fieldValue***

Format 2

POST {baseURI}/msr/pool/*poolId***/data/***transparentDataType/fieldName/fieldValue***

- ***poolId***: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-99999999999999999999`
- ***transparentDataType***: A user defined type/name for the transparent data
  - Value is `poolstate` for the PoolState transparent data
- ***fieldName***: A user defined field within the transparent data
  - For the PoolState entity this corresponds to a 'property' within the entity
    - The ***`fieldName`*** is stored exactly as it is sent in the request. The case of ***`fieldName`*** would change if an update is done using a different case.
- ***fieldValue***: Corresponding field value assigned to *fieldName*.

Request Content

None.

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
201	-	Data field was successfully created/updated
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4067	Multiple matching fields found
404	MSR4001	Pool is not found
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined

Examples

Request #1

A request is made to create a property in the *poolstate* transparent data for a pool. The property name is *mcc* and the property value is *315*. The pool does not have an existing PoolState property called *mcc*.

Request URL: `POST {baseURI}/msr/pool/10000/data/poolstate/mcc/315`

Request Content: None

Response #1

The request is successful, and the property *mcc* with value *315* was created.

HTTP Status Code: `201`

Response Content: None

Request #2

A request is made to create a property in the *poolstate* transparent data for a pool. The property name is *mcc* and the property value is *315*. The pool does not have an existing PoolState property called *mcc*. The pool does not have the PoolState transparent data.

Request URL: `PUT {baseURI}/msr/pool/10000/data/poolstate/mcc/315`

Request Content: None

Response #2

The request is successful, and the property *mcc* as well as the PoolState entity is created.

HTTP Status Code: `201`

Response Content: None

Request #3

A request is made to update a property in the *poolstate* transparent data for a pool. The property name is *mcc*. The pool has an existing PoolState property called *mcc*.

Request URL: `POST {baseURI}/msr/pool/10000/data/poolstate/mcc/400`

Request Content: None

Response #3

The request is successful, and the property *mcc* was updated.

HTTP Status Code: `201`

Response Content: None

Request #4

A request is made to update a property in the *poolstate* transparent data for a pool. The property name is *mcc*. Two properties with the name *mcc* exist.

Request URL: Request URL: `PUT {baseURI}/msr/pool/10000/data/poolstate/mcc`

Request Content: None

Response #4

The request fails, as more than one property called *mcc* exists.

HTTP Status Code: `400`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4067">errorText</error>
```

7.6.2 Get Data Field

Description

This operation retrieves a data field within a transparent data for the pool identified by the *poolId*.

All fields that match the requested *fieldName* are returned.

If more than one field matches the requested *fieldName*, then all matching fields will be returned.

The transparent data field is specified in *fieldName*.

Note: The *fieldName* is NOT case-sensitive. If a field existed called "mcc", then an attempt to get a field called "MCC" would be successful.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *transparentDataType* must reference a valid Pooled transparent Entity in the Interface Entity Map table in the SEC.

A field within the transparent data should exist for the pool.

Request URL

`GET {baseURI}/msr/pool/`*`poolId`***`/data/`***`transparentDataType/fieldName`***

- ***poolId***: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-99999999999999999999`
- ***transparentDataType***: A user defined type/name for the transparent data
  - Value is `poolstate` for the PoolState transparent data
- ***fieldName***: A user defined field within the transparent data
  - **Note**: For the PoolState entity this corresponds to a 'property' within the entity

Request Content

None.

Response Content

A `<pool>` element that contains a `<data>` element, which contains the specified transparent data field (if it exists) for the identified pool.

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="transparentDataType">
<![CDATA[
cdataFieldValue
]]>
  </data>
</pool>
```

- *transparentDataType*: A user defined type/name for the transparent data
  - Value is `poolstate` for the PoolState transparent data
- *cdataFieldValue*: Contents of the XML data "blob", with the field from the transparent data

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Requested data field exists for pool
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data field does not exist

Examples

Request #1

A request is made to get the property *mcc* in the *poolstate* transparent data for a pool. The property *mcc* exists.

Request URL: `GET {BaseURI}/msr/pool/10000/data/poolstate/mcc`

Request Content: None

Response #1

The request is successful, and the requested property is returned.

HTTP Status Code: `200`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
<data name="poolstate">
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<state>
  <version>1</version>
  <property>
    <name>P3</name>
    <value>200</value>
  </property>
</state>
```

```
]]>
</data>
</pool>
```

Request #2

A request is made to get property with name *mcc* in the *poolstate* transparent data for a pool. *The* property with name *mcc* does not exist.

Request URL: `GET {BaseURI}/msr/pool/10000/data/poolstate/mcc`

Request Content: None

Response #2

The request fails, because the requested property does not exist.

HTTP Status Code: `400`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4059">errorText</error>
```

7.6.3 Delete Data Field

Description

This operation deletes a data field within a transparent data for the pool identified by the *poolId*.

The field identifier is specified in *fieldName*.

If more than one data field matches the requested *fieldName*, then all matching fields are deleted.

Note: If the specified field does not exist, this is not considered an error and a successful result will be returned.

Note: The *fieldName* is NOT case-sensitive. If a field existed called "mcc", then an attempt to delete a field called "MCC" would be successful.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

The *transparentDataType* must reference a valid Pooled transparent Entity in the Interface Entity Map table in the SEC.

Request URL

`DELETE {baseURI}/msr/pool/`*`poolId`***`/data/`***`transparentDataType/fieldName`***

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-99999999999999999999`
- *transparentDataType*: A user defined type/name for the transparent data
  - Value is `poolstate` for the PoolState transparent data
- *fieldName*: A user defined field within the transparent data

- **Note**: For the PoolState entity this corresponds to a 'property' within the entity

Request Content

None.

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested transparent data field was successfully deleted
400	MSR4067	Multiple matching fields found
400	MSR4064	Occurrence constraint violation
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found

Examples

Request #1

A request is made to delete the *mcc property* in the *poolstate* transparent data. The *mcc* property exists in the PoolState data.

Request URL: `DELETE {baseURI}/msr/pool/10000/data/poolstate/mcc`

Request Content: None

Response #1

The request is successful, and the property in the PoolState transparent data was deleted.

HTTP Status Code: `204`

Response Content: None

Request #2

A request is made to delete the *mcc* property in the *poolstate* transparent data. The *mcc* property does NOT exist in the PoolState transparent data.

Request URL: `DELETE {baseURI}/msr/pool/10000/data/poolstate/mcc`

Request Content: None

Response #2

The request is successful, even though the *mcc* property does not exist.

HTTP Status Code: `204`

Response Content: None

Request #3

A request is made to delete the *mcc* property in the *poolstate* transparent data The PoolState opaque data is a valid entity, but the requested pool does not contain any PoolState opaque data.

Request URL: `DELETE {baseURI}/msr/pool/10000/data/poolstate/mcc`

Request Content: None

Response #3

The request fails, because the specified pool does not contain PoolState data.

HTTP Status Code: `404`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4058">errorText</error>
```

7.7 Additional Pool Commands

Table 26: Summary of Additional Pool Commands

Command	Description	Key(s)	Command Syntax
Add Member to Pool	Add subscriber to a Pool	PoolID and (MSISDN, IMSI, NAI or AccountId)	**POST {BaseURI}/msr/pool/***poolId***/member/***subKeyName/subKeyValue*
Remove Member from Pool	Remove subscriber from a Pool		**DELETE {BaseURI}/msr/pool/***poolId***/member/***subKeyName/subKeyValue*
Get Pool Members	Retrieve pool member subscribers by PoolID	PoolID	**GET {BaseURI}/msr/pool/***poolId***/member**
Get PoolID	Retrieve PoolID for specified member subscriber	(MSISDN, IMSI, NAI or AccountId	**GET {BaseURI}/msr/sub/***subKeyName/subKeyValue***/pool**

7.7.1 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, 1-22 digits in length
  - Values: `1-99999999999999999999`
- 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.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Subscriber successfully added to pool
400	MSR4100	Maximum number of Subscribers in a Basic Pool has been exceeded
404	MSR4001	Subscriber is not found
404	MSR4061	Specified pool does not exist
409	MSR4055	Subscriber is already a member of a pool

Examples

Request #1

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is not 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 basic pool. Both the pool and the subscriber exist. The subscriber is not a member of a pool. The basic 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="MSR4100">errorText</error>
```

7.7.2 Remove Member from Pool

Description

This operation removes a Subscriber from a Pool.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

A subscriber with the key of the *keyName*/*keyValue* supplied must exist.

The subscriber must be a member of the specified pool.

Request URL

DELETE {BaseURI}/msr/pool/*poolId***/member/***subKeyName***/***subKeyValue*

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: 1-9999999999999999999999
- *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.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Subscriber successfully removed from pool
404	MSR4001	Subscriber is not found
404	MSR4061	Specified pool does not exist
404	MSR4062	Subscriber is not a member of the given pool

Examples

Request #1

A request is made to remove a subscriber from a pool. Both the pool and the subscriber exist. The subscriber is a member of the pool.

Request URL: `DELETE {BaseURI}/msr/pool/100000/member/MSISDN/380561234567`

Request Content: None

Response #1

The request is successful, and the subscriber 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>
```

7.7.3 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, 1-22 digits in length
  - Values: `1-99999999999999999999`

Request Content

None.

Response Content

A `<members>` element that contains a `<member>` element for every subscriber that is a member of the pool. The `<member>` element is optional. There can be zero, one or many `<member>` elements. It is only present if the pool has member subscribers. One instance is present for every subscriber that is a member of the pool. A `<member>` element contains details about a single subscriber, containing all known user identities for that subscriber, one user identity per `<id>` element. There can be one or many `<id>` elements per `<member>` element.

```
<members>
[
  <member>
    <id><name>keyName1</name><value>keyValue1</value></id>
[
    <id><name>keyName2</name><value>keyValue2</value></id>
    :
    <id><name>keyNameN</name><value>keyValueN</value></id>
]
  </member>
]
[
  <member>
    <id><name>keyName1</name><value>keyValue1</value></id>
[
    <id><name>keyName2</name><value>keyValue2</value></id>
    :
    <id><name>keyNameN</name><value>keyValueN</value></id>
]
  </member>
  :
  <member>
    <id><name>keyName1</name><value>keyValue1</value></id>
```

```
[
    <id><name>keyName2</name><value>keyValue2</value></id>
            :
    <id><name>keyNameN</name><value>keyValueN</value></id>
]
  </member>
]
</members>
```

- ***keyNameX***: A key field for the member subscriber
  - Value is either `IMSI`, `MSISDN`, `NAI`, or `AccountId`
- ***keyValueX***: Corresponding key field value assigned to *keyNameX*

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Pool exists, and membership returned OK
404	MSR4061	Specified pool does not exist

Examples

Request #1

A request is made to get the list of subscribers for a pool.

Request URL: `GET {BaseURI}/msr/pool/100000/member`

Request Content: None

Response #1

The request is successful, and the 3 member subscribers are returned.

HTTP Status Code: `200`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  </member>
  <member>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
    <id><name>NAI</name><value>joe@wireless.com</value></id>
    <id><name>NAI</name><value>p12321@mynet.com</value></id>
  </member>
</members>
```

Request #2

A request is made to get the list of subscribers for a pool. The pool exists, but has no member subscribers.

Request URL: `GET {BaseURI}/msr/pool/200000/member`

Request Content: None

Response #2

The request is successful, and no member subscribers are returned.

HTTP Status Code: `200`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
</members>
```

Request #3

A request is made to get the list of subscribers for a pool. The pool does not exist.

Request URL: `GET {BaseURI}/msr/pool/300000/member`

Request Content: None

Response #3

The request fails, because the pool was not found.

HTTP Status Code: `404`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4061">errorText</error>
```

7.7.4 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, 1-22 digits in length
  - Values: `1-99999999999999999999`

Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	-	Subscriber pool membership successfully returned
404	MSR4001	Subscriber is not found
404	MSR4062	Subscriber is not a member of a pool

Examples

Request #1

A request is made to get the 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>
```

7.8 Pool Special Operation Commands

A transparent data entity may contain data that is organized in "rows". An example of a row is a specific quota within the PoolQuota entity.

The required row is identified in the request by the *rowIdValue*.

A specific instance of a quota (i.e. a specified row) within the PoolQuota transparent data entity can have its fields reset to pre-defined values using a provisioning command.

Table 27: Summary of Pool Special Operation Commands

Command	Description	Key(s)	Command Syntax
Reset Pool Quota	Reset the fields within the specified Pool Quota	PoolID and Row Identifier or PoolID, Row Identifier and Instance Identifier	**POST {BaseURI}/msr/pool/***keyName*/**data/***transparentDataType*/*rowIdValue*
	Reset the fields within the specified Pool Quota and instance identifier		**POST {BaseURI}/msr/pool/***keyName*/**data/***transparentDataType*/*rowIdValue*/**row/**instanceFieldName/instanceFieldValue

7.8.1 Reset Pool Quota

Description

This operation resets a particular quota row within the PoolQuota transparent data associated with a pool.

If more than one row matches the requested *rowIdValue*, then the reset request will fail.

If the pool has PoolQuota transparent data, then the configured values within the specified quota row are reset to the configured reset values.

Note: The *rowIdValue* is case-sensitive. If a row existed called "DayPass", then an attempt to reset a quota row called "DayPass" would be successful, but an attempt to reset a quota row called "DAYPASS" would fail.

Note: When a PoolQuota instance is reset using the "Pool Reset Quota" command, each resettable field is set to its defined reset value. If the field does *not* currently exist, it is *not* created. But, if a resettable field does not exist, and the field has a default value, then the field will then get created with the default value.

Prerequisites

A pool with the key of the *keyName* supplied must exist.

The PoolQuota transparent data must exist for the pool.

The specified quota row must exist in the PoolQuota transparent data.

Request URL

Without Instance Identifier

POST {BaseURI}/msr/pool/*poolId***/data/***transparentDataType/rowIdValue*

With Instance Identifier

POST {BaseURI}/msr/pool/*poolId***/data/***transparentDataType/rowIdValue***/row/***instanceFieldName/instanceFieldValue*

- *poolId*: PoolID value of the pool. Numeric value, 1-22 digits in length
  - Values: `1-99999999999999999999`
- *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
- *instanceFieldName*: A user defined field within the data row that is used to define a unique row instance
  - Value is `cid` for the PoolQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

Request Content

None.

Response Content

None.

Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	-	Requested transparent data row was successfully reset
400	MSR4067	Multiple matching rows found
404	MSR4001	PoolID is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist

HTTP Status Code	Error Code	Description
409	MSR4063	Entity cannot be reset

Examples

Request #1

A request is made to reset the *PQ1* PoolQuota row for a pool. The pool has PoolQuota transparent data, and the PoolQuota transparent data contains a PoolQuota row called *PQ1*.

Request URL: `POST {baseURI}/msr/pool/10000/data/poolquota/PQ1`

Request Content: None

Response #1

The request is successful, and the specified PoolQuota row was reset.

HTTP Status Code: `204`

Response Content: None

Request #2

A request is made to reset the *PQ1* PoolQuota row for a pool. The pool does not have PoolQuota transparent data.

Request URL: `POST {baseURI}/msr/pool/10000/data/poolquota/PQ1`

Request Content: None

Response #2

The request fails because the pool does not have PoolQuota transparent data.

HTTP Status Code: `404`

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4058">errorText</error>
```

Request #3

A request is made to reset the *PQ6* PoolQuota row for a pool. The pool has PoolQuota transparent data, but the PoolQuota transparent data does NOT contain a PoolQuota row called *PQ6*.

Request URL: `POST {baseURI}/msr/pool/10000/data/poolquota/PQ6`

Request Content: None

Response #3

The request fails, because the *PQ6* row does not exist.

HTTP Status Code: 404

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4059">errorText</error>
```

Request #4

A request is made to reset the *Weekday* PoolQuota row for a pool. The pool has PoolQuota transparent data, and the PoolQuota transparent data contains two rows called *Weekday*.

Request URL: `POST {baseURI}/msr/pool/10000/data/poolquota/Weekday`

Request Content: None

Response #4

The request fails, as more than one row called *Weekday* exists.

HTTP Status Code: 400

Response Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<error code="MSR4067">errorText</error>
```

Request #5

A request is made to reset the *PQ1* PoolQuota row for a pool having *cid* of value *45678*. The pool has PoolQuota transparent data, and the PoolQuota transparent data contains a PoolQuota row called *PQ1* having *cid* of value *45678*.

Request URL: `POST {baseURI}/msr/pool/10000/data/poolquota/PQ1/row/cid/45678`

Request Content: None

Response #5

The request is successful, and the specified PoolQuota row was reset.

HTTP Status Code: 204

Response Content: None

Appendix A. REST Interface System Variables

The REST interface has a set of system variables that affect its operation as it runs. REST interface variables (shown below in Table 25) can be set via the UDR GUI and can be changed at runtime to effect dynamic server reconfiguration.

Table 28: Bulk Import/Export variables

Parameter	Description
REST Interface Port	REST Interface TCP Listening Port. NOTE: Changes to the TCP listening port do not take effect until the 'udrprov' process is restarted. Also, you must specify a different port than the SOAP interface. DEFAULT = 8787; RANGE = 0-65535
REST Interface Idle Timeout	The maximum time (in seconds) that an open REST connection will remain active without a request being sent, before the connection is dropped. DEFAULT = 1200; RANGE = 1-86400
Maximum REST Connections	Maximum number of simultaneous REST Interface client connections. DEFAULT = 100; RANGE = 1-100
Allow REST Connections	Whether or not to allow incoming provisioning connections on the REST Interface. DEFAULT = UNCHECKED
REST Secure Mode	Whether the REST Interface operates in secure mode (using TLS), or unsecure mode (plain text). NOTE: Changes to the Secure Mode do not take effect until the 'udrprov' process is restarted. DEFAULT = Unsecure
Transaction Durability Timeout*	The amount of time (in seconds) allowed between a transaction being committed and it becoming durable. If Transaction Durability Timeout lapse, DURABILITY_TIMEOUT response is sent to the originating client. The associated request should be resent to ensure that the request was committed. DEFAULT = 5; RANGE = 2-3600
Compatibility Mode*	Indicates whether backwards compatibility is enabled. NOTE: Change to Compatibility Mode may cause the existing provisioning connections to be dropped. DEFAULT = R10.0+

Appendix B. Legacy SPR Compatibility Mode

UDR can be configured to run in a compatibility mode with the legacy SPR.

When the `Compatibility Mode` system option (see Appendix A), which is configurable by the UDR GUI , is set to "R10.0+" then UDR will behave as described in the main body of this document. When `Compatibility Mode` is set to "R9.x", then the differences contained in this appendix will apply.

Enabling this configuration option results in the format of some request/responses being different to the default UDR behavior. This appendix lists the different request/responses 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 version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <field name="cid"/>
  <field name="time"/>
  <field name="totalVolume">0</field>
  <field name="inputVolume">0</field>
  <field name="outputVolume">0</field>
  <field name="serviceSpecific"/>
  <field name="nextResetTime"/>
  <field name="Type">quota</field>
  <field name="grantedTotalVolume">0</field>
  <field name="grantedInputVolume">0</field>
  <field name="grantedOutputVolume">0</field>
  <field name="grantedTime"/>
  <field name="grantedServiceSpecific"/>
  <field name="QuotaState">Valid/Inactive</field>
  <field name="RefInstanceId"/>
  <field name="name">test</field>
</usage>
]]>
  </data>
</subscriber>
```

Note: If more than one matching row is found, then multiple <quota> rows are returned. E.g. :

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  :
</usage>
]]>
  </data>
```

```
</subscriber>

<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  :
</usage>
]]>
  </data>
</subscriber>
```

Appendix C. My Oracle Support (MOS)

MOS (https://support.oracle.com) is your initial point of contact for all product support and training needs. A representative at Customer Access Support (CAS) can assist you with MOS registration.

Call the CAS main number at **1-800-223-1711** (toll-free in the US), or call the Oracle Support hotline for your local country from the list at http://www.oracle.com/us/support/contact/index.html. When calling, make the selections in the sequence shown below on the Support telephone menu:

1. Select **2** for New Service Request
2. Select **3** for Hardware, Networking and Solaris Operating System Support
3. Select **2** for Non-technical issue

You will be connected to a live agent who can assist you with MOS registration and provide Support Identifiers. Simply mention you are a Tekelec Customer new to MOS.

MOS is available 24 hours a day, 7 days a week, 365 days a year.

Appendix D. Customer Training

Oracle University offers expert training on Oracle Communications solutions for service providers and enterprises. Make sure your staff has the skills to configure, customize, administer and operate your communications solutions, so that your business can realize all of the benefits that these rich solutions offer.

Visit the Oracle University web site to view and register for Oracle Communications training: education.oracle.com/communication.

To reach Oracle University:

- In the US please dial 800-529-0165.
- In Canada, please dial 866-825-9790.
- In Germany please dial 0180 2000 526 (toll free) or +49 8914301200 (International).
- In Spain please dial +34 91 6267 792.
- In the United Kingdom please dial 0845 777 7 711 (toll free) or +44 11 89 726 500 (International).

For the appropriate country or region contact phone number for the rest of the world, please visit Oracle University's web site at www.oracle.com/education/contacts.

Appendix E. Locate Product Documentation on Oracle Help Center

Oracle customer documentation is available on the web at the Oracle Help Center site, http://docs.oracle.com. You do not have to register to access these documents. Viewing these files requires Adobe Acrobat Reader, which can be downloaded at www.adobe.com.

1. Go to the Oracle Communications page on Oracle Help Center at: http://docs.oracle.com/en/industries/communications/index.html
2. Navigate to your product and release number to see the list of books for that release. You can view or download any document in PDF format.