Oracle® Communications

Specification

User Data Repository
REST Provisioning Interface Specification
for Release 12.11.0

F56667-01

May 2022

ORACLE®

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

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

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

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

See more information on My Oracle Support in Appendix C.

Oracle Communications User Data Repository REST Provisioning Interface Specification, Release 12.11.0

F56667-01

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Purpose and Scope

This document describes 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 or 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 need to use the REST interface.

1.2 References

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
[5] OpenSSL based on SSLeay library developed by Eric A. Young and Tim J. Hudson.

1.3 Acronyms

This section lists terms and acronyms specific to this document.

Table 1: Acronyms

Acronym or 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
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity, or IMSI [im-zee]
IP	Internet Protocol
KPI	Key Performance Indicator

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

Chapter 2. System Architecture

2.1 Overview

Oracle Communications User Data Repository (UDR) performs the function of a Subscriber Profile Repository (SPR), a database system that acts as a single logical repository that stores subscriber data. The subscriber data stored in the HSS, HLR, AuC, or application servers is stored in UDR as specified in the 3GPP UDC information model [3]. UDR facilitates the sharing and provisioning of user data throughout the services of the 3GPP system. UDR serves several Applications Front End, such as PCRF or HSS or HLR or AuCFEs.

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

Oracle Communications User Data Repository is a database system providing the storage and management of subscriber policy control data for PCRF nodes. Subscriber and pool data is created, retrieved, modified, or deleted through the provisioning or by the Sh interface peers (PCRF). Subscriber and pool data is stored in Oracle Communications User Data Repository:

- Subscriber
    - Profile
    - Quota
    - State
    - Dynamic quota
- Pool
    - Pool profile
    - Pool quota
    - Pool state
    - Pool dynamic quota

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

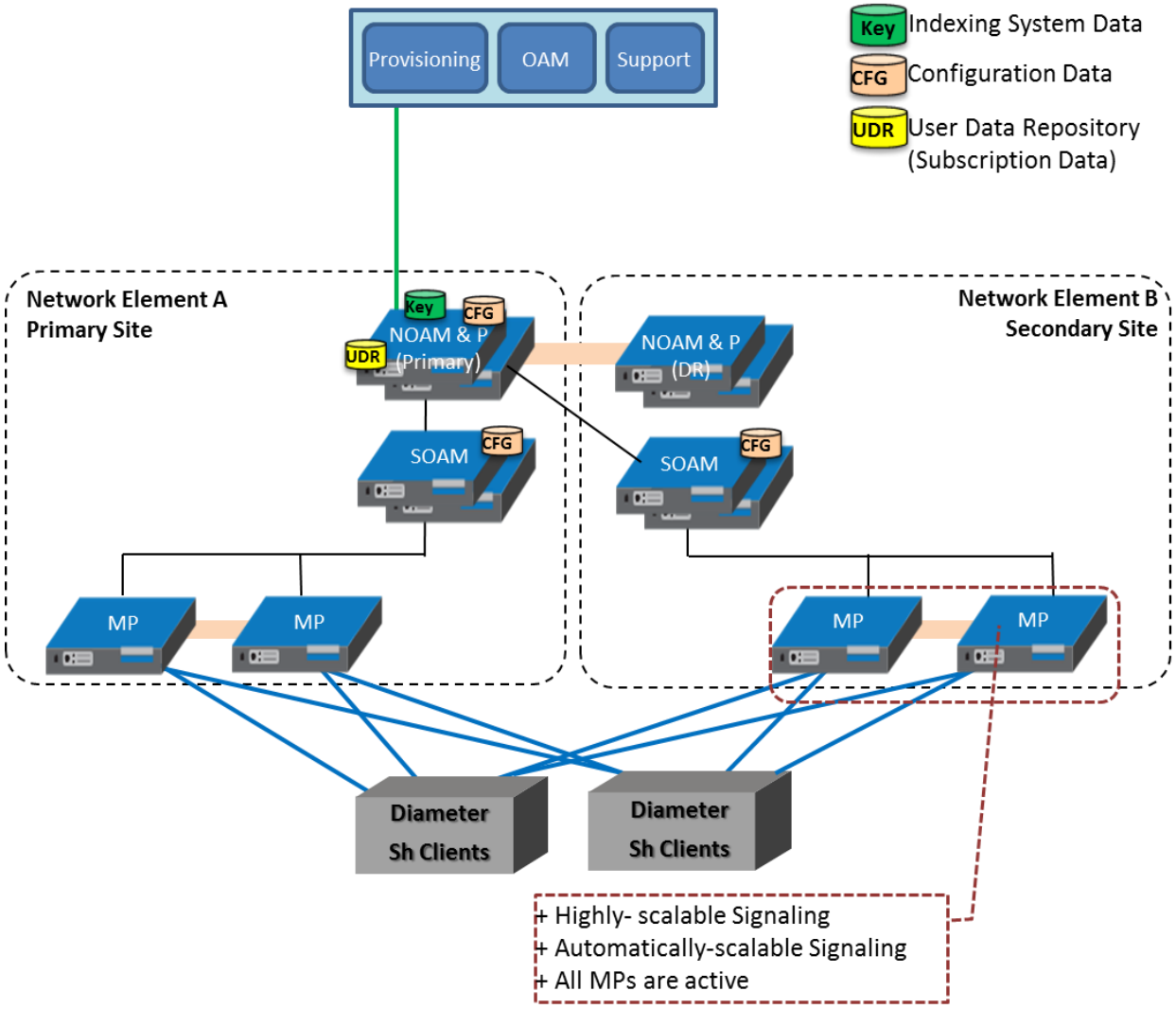
Figure 1 also shows that Oracle Communications User Data Repository consists of several functional blocks. The Message Processors (MP) support a variety of protocols that entail the front-end signaling to peer network nodes. The back-end UDR database resides on the NOAMP servers. This release focuses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

The Network level OAMP server (NOAMP) in Figure 1 provisions, configures, and maintains functions for all the network elements.

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

The MP functions as the client-side of the network application and is used for 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 uses these data manipulation commands for subscribers and pools.

2.2.1 Subscriber

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

  Quota, state and dynamic quota
- Subscriber transparent data create, retrieve, modify, and delete

  Quota, state and dynamic quota
- Reset of a row in subscriber quota transparent data

2.2.2 Pool

- Pool profile create, retrieve, modify, and delete
- Pool profile field add, retrieve, modify, and delete
- Pool opaque data create, retrieve, modify, and delete

  Pool quota, pool state and pool dynamic quota

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

  Pool quota, pool state and pool dynamic quota

- Reset of a row in pool quota transparent data
- Pool subscriber membership operations
  - Add and remove from pool
  - Get pool subscriber membership
  - Get pool for subscriber

2.3 REST Application Server (RAS)

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

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

2.4 Provisioning Clients

The RAS is used to connect 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 or IP connections to the RAS running on the active NOAMP using the VIP for the primary NOAMP.

Provisioning clients must re-establish connections with the RAS using the VIP for the primary UDR after switchover from the active to the standby UDR server for the primary. Provisioning clients also must redirect connections to the VIP for the secondary after switchover from the primary UDR site to the disaster recovery UDR site.

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

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

2.5 Security

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

- Client server IP address white list
- Secure connections using TLS

2.5.1 Client Server IP Address White List

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

The system configuration process maintains a whitelist of server IP addresses and/or IP address ranges from which clients are authorized to establish a TCP or IP connection from.

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

2.5.2 Secure Connection using TLS

The RAS supports secure (encrypted) connections between provisioning clients and the RAS using Transport Layer Security version 1.0 (TLSv1.0) protocol implemented using OpenSSL.

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

The capabilities of TLS address several fundamental concerns about communication over TCP or IP networks:

- TLS server authentication

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

- TLS client authentication

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

- An encrypted TLS connection

  Requires all information 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 is used for message integrity. Message integrity is used to determine if the data has been tampered with since it was sent by the partner application.

Depending which mode, the RAS is configured to operate in (secure or unsecure), provisioning clients can connect using unsecure or secure connections to the well-known TCP or TLS listening port for the RAS (configured using the REST Secure Mode configuration variable through UDR GUI).

A TLS-enabled connection is slower than an unsecure TCP or 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.

TLS Certificates and Public or 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 the public key for the owner. Any data encrypted with this public key can be decrypted only using the corresponding, matching private key, which is held by the owner of the certificate.

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

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

Certificate and Key PEM-encoded Files	Description
tklcCaCert.pem	Oracle self-signed un-trusted root Certification Authority (CA) X.509v3 certificate.
serverCert.pem	The RAS's X.509v3 certificate and 2,048-bit RSA public key digitally signed by Oracle 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 Oracle 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 Oracle 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 Oracle Certification Authority (CA) using SHA-1 message digest algorithm.

Provisioning clients are required to send a TLS authenticating X.509v3 certificate when requested by the RAS during the secure connection handshake protocol for mutual (two-way) authentication. If the provisioning client does not submit a certificate that is issued or signed by Oracle Certification Authority (CA), it is not able to establish a secure connection with the RAS.

Supported TLS Cipher Suites

A cipher suite is a set or combination of lower-level algorithms that a TLS-enabled connection uses to do authentication, key exchange, and stream encryption. Table 3 lists the set of TLS cipher suites from the relevant specification and their OpenSSL equivalents that are supported by the RAS to secure a TLS-enabled connection with provisioning clients. The cipher suites are listed and selected for use in the order of key strength, from highest to lowest. This ensures that during the handshake protocol of a TLS-enabled connection, cipher suite negotiation selects the most secure suite possible from the list of cipher suites the client wishes to support, and if necessary, back off to the next most secure, and so on down the list.

NOTE: Cipher suites containing anonymous DH ciphers, low bit-size ciphers (those using 64 or 56 bit encryption algorithms but excluding export cipher suites), export-crippled ciphers (including 40 and 56 bits algorithms), or the MD5 hash algorithm are not supported due to their algorithms having 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 or IP connection. However, connections are closed after being idle for a time limit configured in idle timeout (see section 2.9.3).

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

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

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

For example, if the certified maximum provisioning TPS rate is 200 TPS, and the Maximum REST Connections (see 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 issue requests, each request is queued and processed in the order in which it was received on a per connection basis. The client must wait for a response from one request before issuing another.

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

All requests from a client sent on a single connection are processed by UDR serially. Multiple requests can be sent without receiving a response, but each request is queued and not processed until the previous request has

completed. A client can send multiple requests across multiple connections, and these may run in parallel (but requests on each connection are still processed serially).

2.8 Database Transactions

Each create, update, and delete request coming from the REST interface triggers a unique database transaction, that is, 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.

Atomicity

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

Consistency

Data across all requests performed inside a transaction is consistent.

Isolation

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

Durability

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

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

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 additive configurable levels of durability are supported:

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

2.9 Connection Management

It is possible to enable or disable or 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 is rejected with a SERVICE_UNAVAILABLE error indicating the service is unavailable.

For an example of a provisioning request or response when provisioning is disabled, see the last example in section 5.1.1.

2.9.3 Idle Timeout

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

2.9.4 Maximum Simultaneous Connections

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

2.9.5 TCP Port Number

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

2.10 Behavior during Low Free System Memory

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

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*error>
```

2.11 Congestion Control

If UDR starts to encounter congestion (based on high CPU usage), then based on the congestion level, UDR rejects some requests (based on the HTTP method, see section 3.1.1).

If the minor CPU usage threshold is crossed (CL1), then UDR rejects GET requests

If the major CPU usage threshold is crossed (CL2), then UDR rejects GET and PUT requests

If the critical CPU usage threshold is crossed (CL3), then UDR rejects all requests

The 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*error>
```

2.12 Pools Spanning UDRs

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

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

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

2.13 Enterprise Pools

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

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

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

2.13.1 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 on RESTful concepts. This section defines the operations that can be performed using the REST interface.

2.14 Rest Conventions

The REST interface uses the RESTful concepts:

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

2.14.1 HTTP(S) Request Headers

The HTTP(S) requirements must be followed.

HTTP version

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

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

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

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

Accept Header

Set the `Accept` header property to the MIME version using one of theses formats:

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

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

Operations in Oracle Communications User Data Repository support both versions 1 and 2.0.

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

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

Transfer-Encoding Header

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

```
Transfer-Encoding: chunked
```

Requests with body content

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

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

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

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

2.14.2 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 includes response content that has an error message entity.

Table 4 provides a list of 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 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 more data.

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

Error Code	Description
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
MSR4100	Maximum number of Subscribers in a Basic Pool has been exceeded

Error Code	Description
MSR4101	Enterprise to Basic Pool Conversion failed threshold exceeded
MSR4102	Provisioning Request Timeout, as no response was received from Remote UDR
MSR4103	A key is detected to be already in the system for an AE subscriber

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

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

In the examples, the error text associated with the MSRxxxx is not shown as this varies depending on the entity, key, or field values used.

Chapter 3. REST Interface Message Definitions

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

3.1 Message Conventions

3.1.1 HTTP Method

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

3.1.2 Base URI

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

- http://udr.oracle.com:8787/rs
- https://1.2.3.4:8787/rs

3.1.3 REST URL

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

Subscriber or Pool in URL

Keyword sub indicates subscriber operations and pool indicates pool operations

For example, for a subscriber:

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

And for a pool:

```
DELETE {baseURI}/msr/pool/100000/field/Custom12
```

Opaque Data Operations in URL

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

Opaque data operations can be performed on entities defined as opaque or transparent. An opaque data operation works on the XML blob creating, getting, or deleting.

For example when deleting the Quota data for a subscriber:

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

Field in URL

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

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

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

Transparent Data Row Operations in URL

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

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

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

Transparent Data Row Operations using an Instance Identifier in URL

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

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

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

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
```

Transparent Data Row Field Operations using an Instance Identifier in URL

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

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

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

Transparent Data Field Operations in URL

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

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

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

3.1.4 URL Character Encoding

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.

3.2 Case Sensitivity

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

For example, this is valid:

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

Tis example is not valid:

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

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

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

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

3.2.1 Case Sensitivity 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 field is created using the name INPUTVOLUME, then it is returned in a response as inputVolume
- When a *fieldValue* is used to find a field (such as when using the Delete Field Value command), the field value is case-sensitive. If a multi-value field contained the values DayPass,Weekend,Evening and the Delete Field Value command was used to delete the value WEEKEND, then this fails
- When an attribute in the XML blob contains the row identifier name (*rowIdName)* the row identifier name is not case-sensitive. For example, in Quota the element `<quota name="AggregateLimit">`, contains the attribute called name.
- When a *rowIdValue* is used to find a row (such as when using the `Get Row` command), the row identifier value is case-sensitive. If an entity contained a row called DayPass, and the `Get Row` command was used to get the row DAYPASS, then this fails
- When a *keyValue* is specified in the URL (such as for an NAI), the value is case-sensitive. For example, for a subscriber with an NAI of mum@foo.com, then Mum@foo.com or MUM@FOO.COM does not find the subscriber

3.3 XML Comments in a Request

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

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

If a request contains a comment, the request is rejected with the HTTP status code.

HTTP Status Code

400

Response Content

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

3.4 Request Content in a Request

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

Request Content in a Request 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.

This is an example of request content for UDR for DSR based EIR solution with IMEI as key.

Request 3

Request URL

```
GET {baseURI}/msr/sub/IMEI/12345770382981
```

Request Content

```
<?xml version="1.0"?>s
<subscriber>
    <field name="IMEI">12345770382981</field>
    <field name="IMSI">123457703829810</field>
    <field name="WL">1</field>
    <field name="BL">0</field>
```

```
    <field name="GL">0</field>
    <field name="SV">00</field>
</subscriber>
```

Response 3

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

Request 4

Request URL

```
PUT {BaseURI}/msr/sub/IMEI/98765439876543
```

Request Content

```
<?xml version="1.0"?>
<subscriber>
    <field name="IMEI">98765439876543</field>
    <field name="IMSI">987654300000000</field>
    <field name="WL">1</field>
    <field name="BL">0</field>
    <field name="GL">0</field>
    <field name="SV">00</field>
</subscriber>
```

Response 4

The request is successful, and IMSI is updated to the new value. The rest of the IMSI values in the request are ignored.

3.5 List of Messages

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

Table 6: Summary of Subscriber Commands

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										Request Content
	Get Profile (GET)			*{keyName}* MSISDN, NAI, IMSI, AccountId	*{keyValue}*								
	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
									{instanceIdentifierField}	{instanceIdentifierValue}			
	Get Row (GET)												
									{instanceIdentifierField}	{instanceIdentifierValue}			
	Delete Row (DELETE)												
									{instanceIdentifierField}	{instanceIdentifierValue}			

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
	Reset Quota (POST)								{instanceIdentifierField}	{instanceIdentifierValue}			
Subscriber Data Row Field	Get Row Field (GET)								{instanceIdentifierField}	{instanceIdentifierValue}	{fieldName}	{FieldValue}	
	Get Row Field Value (GET)								{instanceIdentifierField}	{instanceIdentifierValue}			
	Update Row Field (PUT)								{instanceIdentifierField}	{instanceIdentifierValue}			
	Delete Row Field (DELETE)								{instanceIdentifierField}	{instanceIdentifierValue}			
	Delete Row Field Value(DELETE)								{instanceIdentifierField}	{instanceIdentifierValue}			
Subscriber Data Field	Set Data Field (PUT or POST)												
	Get Data Field (GET)												
	Delete Data Field (DELETE)												

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

Table 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
	Get Pool (GET)				{*keyValue*}								
	Update Pool (PUT)												Request Content
	Delete Pool (DELETE)												

Operation Data	Command (Method)	URL	Main Object	Key Name	Key Value	subObject Type	subObject Name	subObject Value	Instance Field Name	Instance Field Value	Field Name	Field Value	Additional Input
Pool 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
	Get Row (GET)								{instanceIdentifierField}	{instanceIdentifierValue}			
	Delete Row (DELETE)								{instanceIdentifierField}	{instanceIdentifierValue}			
	Reset Pool								{instanceIdentifierField}	{instanceIdentifierValue}			

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
	Quota (POST)								{instanceIdentifierField}	{instanceIdentifierValue}			
Pool Data Row Field	Get Row Field (GET)										{*fieldName*}	{FieldValue}	
									{instanceIdentifierField}	{instanceIdentifierValue}			
	Get Row Field Value (GET)												
									{instanceIdentifierField}	{instanceIdentifierValue}			
	Update Row Field (PUT)												
									{instanceIdentifierField}	{instanceIdentifierValue}			
	Delete Row Field (DELETE)												
									{instanceIdentifierField}	{instanceIdentifierValue}			
	Delete Row Field Value (DELETE)												
									{instanceIdentifierField}	{instanceIdentifierValue}			
Pool Data Field	Set Data Field (PUT or POST)												
	Get Data Field (GET)												
	Delete Data Field (DELETE)												

Chapter 4. UDR Data Model

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

The subscriber-related data includes:

- Profile or Subscriber Data

  Pre-provisioned information that describes the capabilities of each subscriber. This data is typically written by the OSS system (through a provisioning interface) and referenced by the PCRF (through the Sh interface).

- Quota

  Information that represents the use of managed resources by the subscriber (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.

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, IMEI, or NAI to share one quota.

The pool-related data includes:

- Pool profile: Pre-provisioned information that describes a pool
- Pool quota: Information that represents the use of managed resources for the pool (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:

- 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, IMEI, 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:

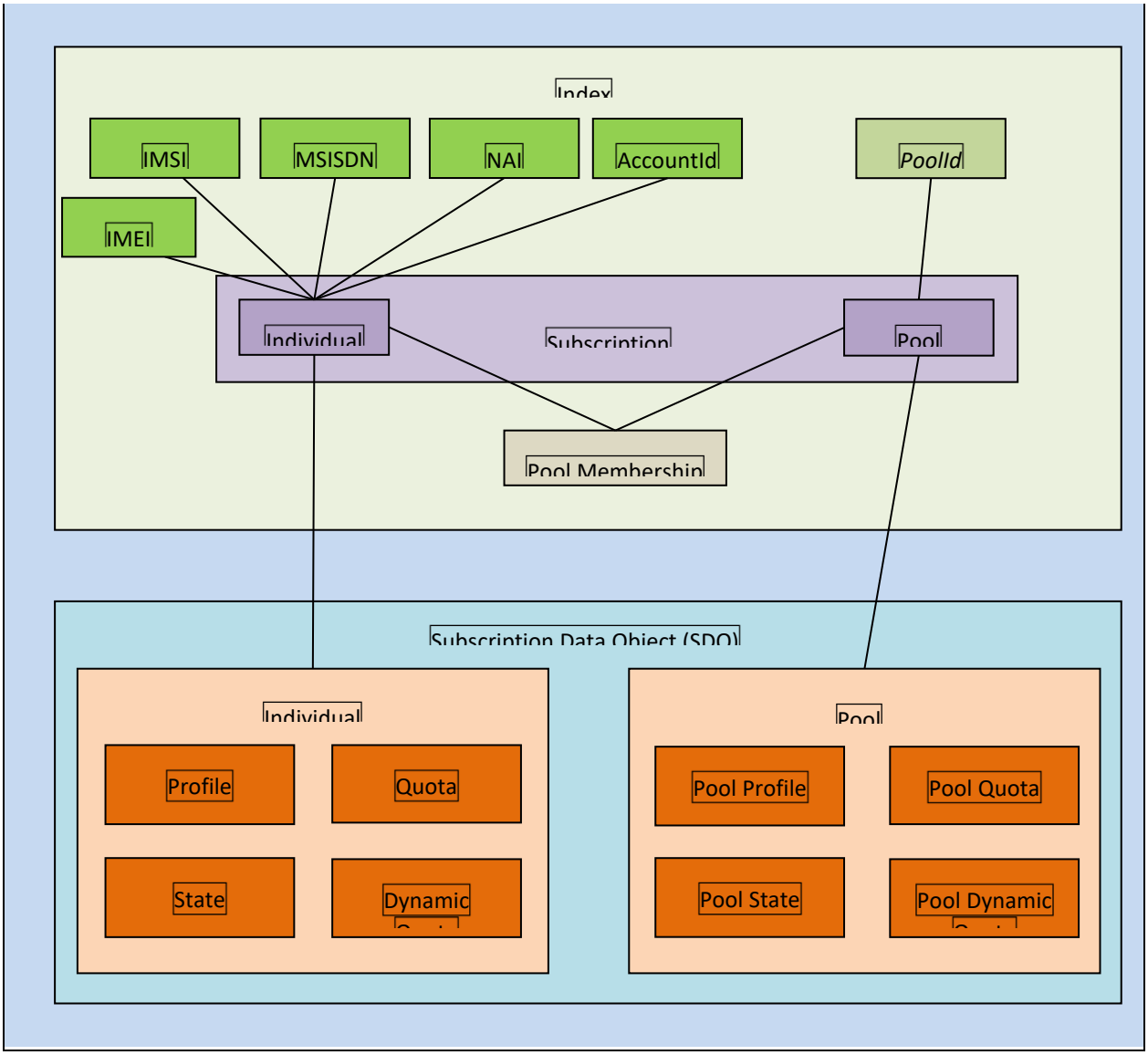
  Each individual subscriber

  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 through IMSI, MSISDN, IMEI, NAI or AccountId. The pool SDO can be accessed through *PoolId.*
- A field in an entity can be defined as mandatory, or optional. A mandatory field must exist and cannot be deleted.
- A field in an entity can have a default value. If an entity is created, and the field is not specified, it is created with the default value.
- A field in an entity can be defined so that after it is created, it cannot be modified. Any attempt to update the field after it is created, fails.
- A field in an entity can have a reset value. If a reset command is used on the entity, those fields with a defined reset value are set to the defined value. This is only applicable to field values in a row for the Quota entity.

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

Figure 2: Data Model



4.1 Subscriber Data

4.1.1 Subscriber Profile

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

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

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

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

Table 8: Subscriber profile Entity Definition

Name (XML tag)	Type	Description
subscriber		**Sequence (multiplicity is 1)**
MSISDN	String	List of MSISDNs (8 to 15 numeric digits). A separate entry is included for each MSISDN associated with the profile for the subscriber.
IMSI	String	List of IMSIs (10 to 15 numeric digits). A separate entry is included for each IMSI associated with the profile for the subscriber.
IMEI	String	List of IMEIs (8 to 14 numeric digits). A separate entry is included for each IMEI associated with the profile for the subscriber.
NAI	String	User or Domain length is between 0 to 63 characters **NOTE:** The limitation for 0 to 63 characters is because if an NAI has more than 63 characters, it may is not be possible to transfer through all devices. The user must ensure that the combination of User and Domain does not exceed 63 characters (not including the @ (at) character). List of NAIs (in format user@domain, user, or @domain). A separate entry is included for each NAI associated with the profile for the subscriber. The user or domain can be empty. Allowed characters for the user: !, %, $, A through Z, a through z, 0 through 9, ., -, _, /, *, =, ^, `, \|, #, ‘, +, ?, {, }, ~ Allowed characters for the domain: A though Z, a though z, 0 through 9, ., -, _ Example NAI Formats: `bob, @privatecorp.example.net` `fred$@example.com` `eng.example.net!nancy@example.net` `eng%nancy@example.net` `bob#+ ?@example.net`
AccountId	String	Any string that can be used to identify the account for the subscriber (1 to 255 characters). Allowed values are any ASCII printable character, values x20 to x7e.
BillingDay	String	Allowed values are 0 to 31. The day of the month (1 to 31) when the associated quota for the subscriber is reset. 0 indicates that the default value configured at the PCRF level is used. This is automatically set in any record where BillingDay is not specified.
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the profile for the subscriber.
Tier	String	Tier for the Subscriber.
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.
Custom3	String	Fields used to store customer-specific data.

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

4.1.2 Quota

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

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

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

The default value in the table is used either:

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

  NOTE: If a field is not re-configurable and is set to a default value, the field is created with the indicated value.

Table 9: Quota Entity Definition

Name (XML tag)	Type	Default Value	Description	Quota Versions
usage			**Sequence (multiplicity is 1)**	1/2/3
version	String		Version of the schema.	1/2/3
quota			**Sequence (multiplicity is N)**	1/2/3
name	String		Quota name (identifier).	1/2/3
cid	String		Internal identifier used to identity a quota in a subscriber profile.	1/2/3
time	String	Empty string ""	This element tracks the time-based resource consumption for a Quota.	1/2/3
totalVolume	String	"0"	This element tracks the bandwidth volume-based resource consumption for a Quota.	1/2/3
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 4.3 for format details.	1/2/3
Type	String	Empty string ""	Type of the resource in use.	2/3
grantedTotalVolume	String	"0"	Granted Total Volume represents the granted total volume for all the subscribers in the pool for pool quota. For individual quota, it represents the granted volume to all the PDN connections for that subscriber.	2/3
grantedInputVolume	String	"0"	Granted Input Volume.	2/3
grantedOutputVolume	String	"0"	Granted Output Volume.	2/3
grantedTime	String	Empty string ""	Granted Total Time.	2/3
grantedServiceSpecific	String	Empty string ""	Granted Service Specific Units.	2/3
QuotaState	String	Empty string ""	State of the resource in use.	3
RefInstanceId	String	Empty string ""	Instance-id of the associated provisioned pass, top-up or roll-over.	3

4.1.3 State

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

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

The default fields configured cannot be reset or set as default value.

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

Table 10: State Entity Definition

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

4.1.4 Dynamic Quota

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

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

The default fields configured are not:

- Resettable
- Defaultable

The *Dynamicquota* entity supports the sequence of attributes in Table 11.

Table 11: Dynamic Quota Entity Definition

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

Name (XML tag)	Type	Description
		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 4.3 for format details.
expirationdatetime	String	The date/time after which the pass or top-up is considered to be exhausted See section 4.3 for format details.
purchasedatetime	String	The date/time when a pass was purchased See section 4.3 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 revalidates quota grants with the PCRF.

4.2 Pool Data

This section provides information on pool and its entities.

4.2.1 Pool Profile

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

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

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

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

Table 12: Pool Profile Entity Definition

Name (XML tag)	Type	Description
pool		**Sequence (multiplicity is 1)**
PoolId	String	Pool identifier (1 to 22 numeric digits, minimum value 1).
BillingDay	Uint8	The day of the month (1 to 31) when the associated quota for the pool is reset.

Name (XML tag)	Type	Description
		0 indicates that the default value configured at the PCRF level is used.
BillingType	String	The billing frequency, monthly, weekly, daily.
Entitlement	String	List of entitlements. A separate entry is included for each entitlement associated with the profile for the pool.
Tier	String	Tier for the pool.
Type	String	Field used to identify an Enterprise Pool. Allowed value is enterprise and is not case-sensitive
Custom1	String	Fields used to store customer-specific data.
Custom2	String	Fields used to store customer-specific data.
Custom3	String	Fields used to store customer-specific data.
Custom4	String	Fields used to store customer-specific data.
Custom5	String	Fields used to store customer-specific data.
Custom6	String	Fields used to store customer-specific data.
Custom7	String	Fields used to store customer-specific data.
Custom8	String	Fields used to store customer-specific data.
Custom9	String	Fields used to store customer-specific data.
Custom10	String	Fields used to store customer-specific data.
Custom11	String	Fields used to store customer-specific data.
Custom12	String	Fields used to store customer-specific data.
Custom13	String	Fields used to store customer-specific data.
Custom14	String	Fields used to store customer-specific data.
Custom15	String	Fields used to store customer-specific data.
Custom16	String	Fields used to store customer-specific data.
Custom17	String	Fields used to store customer-specific data.
Custom18	String	Fields used to store customer-specific data.
Custom19	String	Fields used to store customer-specific data.
Custom20	String	Fields used to store customer-specific data.

4.2.2 Pool Quota

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

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

The *Poolquota* entity attributes are the same as defined for the Quota entity in section 4.1.2.

4.2.3 Pool State

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

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

The default fields configured cannot be reset or set as default value.

The PoolState entity attributes are the same as defined for the State entity in section 4.1.3.

4.2.4 Pool Dynamic Quota

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

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

The default fields configured are not:

- Resettable
- Defaultable

The Pool*Dynamicquota* entity attributes are the same as defined for the *Dynamicquota* entity in section 4.1.4.

4.3 Date/Timestamp Format

The Date/Timestamp format used by many fields is:

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

This corresponds to either:

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

Where:

- CC is century
- YY is year
- MM is month
- DD is day
- T is Date/Time separator

- hh is hour
- mm is minutes
- ss is seconds
- Z is UTC (Coordinated Universal Time)
- +|- is time offset from UTC

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

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

Chapter 5. Subscriber Provisioning

5.1 Subscriber profile Commands

Table 13: Summary of Subscriber profile Commands

Command	Description	Keys	Command Syntax
Create Profile	Create a subscriber or subscriber profile	MSISDN, NAI, IMSI, IMEI, 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`

5.1.1 Create Subscriber

Description

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

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

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

Prerequisites

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

Request URL

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

Request Content

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

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

- *keyNameX*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId

- *keyValueX*: Corresponding key field value assigned to *keyNameX*
- *fieldNameX*: A user defined field in the subscriber profile
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

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

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

Response Content

None.

Table 14 Create Subscriber Response Status/Error Codes

HTTP Status Code	Error Code	Description
201		Successfully created
400	MSR4000	Invalid content request data supplied
400	MSR4003	A key is detected 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
400	MSR4103	A key is detected to be already in the system for an AE subscriber
404	MSR4002	Subscriber field is not defined

Create Subscriber 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

400

Response Content

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

Request 3

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

Request URL

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

Request Content:

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

Response 3

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

HTTP Status Code

400

Response Content

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

Request 4

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

Request URL

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

Request Content:

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

Response 4

The request fails because key values were not supplied.

HTTP Status Code

400

Response Content

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

Request 5

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

Request URL

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

Request Content:

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

Response 5

The request fails, because provisioning has been disabled.

HTTP Status Code

503

Response Content

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

Request 6

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

Request URL

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

Request Content:

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

Response 6

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

HTTP Status Code

400

Response Content

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

Request 7

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

Request URL

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

Request Content:

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

Response 7

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

HTTP Status Code

400

Response Content

```
<?xml version=“1.0” encoding=“UTF-8”?>
<error code=*error>
```

Below is an example of subscriber creation for UDR for DSR based EIR solution.

Request 8

A subscriber is created, with one IMEI, one IMSI field key and WL, BL, GL and SV with default values set.

Request URL

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

Request Content:

```
<?xml version="1.0"?>
<subscriber>
    <field name="IMEI">98765439876543</field>
    <field name="IMSI">987654398765430</field>
    <field name="WL">1</field>
    <field name="BL">0</field>
    <field name="GL">0</field>
    <field name="SV">00</field>
</subscriber>
```

Response 8

The request is successful, and the subscriber was created.

HTTP Status Code

201

Response Content

None

Request 9

A subscriber is created, with two IMEI field as range key and WL, BL and GL with default values set.

This is an example of subscriber creation with IMEI Range for UDR for DSR based EIR solution.

Request URL

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

Request Content:

```
<?xml version="1.0"?>
<subscriber>
    <field name="IMEI">44111111111111</field>
    <field name="IMEI">44441111111111</field>
    <field name="WL">1</field>
    <field name="BL">0</field>
    <field name="GL">0</field>
</subscriber>
```

Response 9

The request is successful, and the subscriber with this range was created.

HTTP Status Code

201

Response Content

None

5.1.2 Get Profile

Description

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

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

Prerequisites

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

Request URL

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId

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

Request Content

None.

Response Content

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

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

- *keyNameX*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId

- *keyValueX*: Corresponding key field value assigned to *keyNameX*

- *fieldNameX*: A user defined field in the subscriber profile
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

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

Table 15 Get Profile Response Status/Error Codes

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

Get Profile 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

400

Response Content

```
<?xml version=“1.0” encoding=“UTF-8”?>
<error code=*error>
```

Request 3

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

This is an example of UDR for DSR based EIR solution

Request URL

```
GET {baseURI}/msr/sub/IMEI/98765439876543
```

Request Content

None

Response 3

The request is successful, and the subscriber was retrieved.

HTTP Status Code

200

Response Content

```
<?xml version="1.0"?>
<subscriber>
    <field name="IMEI">98765439876543</field>
    <field name="IMSI">987654398765430</field>
    <field name="WL">1</field>
    <field name="BL">0</field>
    <field name="GL">0</field>
    <field name="SV">00</field>
</subscriber>
```

Request 4

The subscriber with the given IMEI range is retrieved. The subscriber exists.

This is an example of IMEI Range for UDR for DSR based EIR solution

Request URL

```
GET {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111
```

Request Content

None

Response 4

The request is successful, and the subscriber was retrieved.

HTTP Status Code

200

Response Content

```
<?xml version="1.0"?>
<subscriber>
    <field name="IMEI">44111111111111</field>
    <field name="IMEI">44441111111111</field>
    <field name="WL">1</field>
    <field name="BL">0</field>
    <field name="GL">0</field>
</subscriber>
```

5.1.3 Update Profile

Description

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

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

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

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

Prerequisites

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

Request URL

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId

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

Request Content

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

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

- *keyNameX*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId

- *keyValueX*: Corresponding key field value assigned to *keyNameX*

- *fieldNameX*: A user defined field in the subscriber profile
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

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

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

Response Content

None.

Table 16 Update Profile Response Status/Error Codes

HTTP Status Code	Error Code	Description
204		The subscriber data was replaced successfully
400	MSR4000	Invalid content request data supplied
400	MSR4003	A key is detected 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

Update Profile 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

Request 2

A subscriber is updated having one IMEI, one IMSI is updated where WL is set to zero and BL is set to one. The subscriber exists.

This is the request content from UDR for DSR based EIR solution.

Request URL

```
PUT {baseURI}/msr/sub/IMEI/98765439876543
```

Request Content:

```
<?xml version="1.0"?>
<subscriber>
    <field name="IMEI">98765439876543</field>
    <field name="IMSI">987654300000010</field>
    <field name="WL">0</field>
    <field name="BL">1</field>
    <field name="GL">0</field>
    <field name="SV">00</field>
</subscriber>
```

Response 2

The request is successful, and the subscriber was updated.

HTTP Status Code

204

Response Content

None

Request 3

A subscriber having IMEI range as key, is updated where WL is set to zero and BL is set to one. The subscriber exists.

This is the request content for IMEI range from UDR for DSR based EIR solution.

Request URL

```
PUT {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111
```

Request Content:

```
<?xml version="1.0"?>
<subscriber>
    <field name="IMEI">44111111111111</field>
    <field name="IMEI">44441111111111</field>
    <field name="WL">0</field>
    <field name="BL">1</field>
    <field name="GL">0</field>
</subscriber>
```

Response 3

The request is successful, and the subscriber was updated.

HTTP Status Code

204

Response Content

None

5.1.4 Delete Profile

Description

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

Prerequisites

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

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

Request URL

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId
- *keyValue*: Corresponding key field value assigned to *keyName*

Request Content

None.

Response Content

None.

Table 17 Delete Profile Response Status/Error Codes

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

Delete Profile 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=*error>
```

Request 3

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

Request URL

```
DELETE {baseURI}/msr/sub/IMEI/98765439876543
```

Request Content

None

Response 3

The request is successful.

HTTP Status Code

204

Response Content

None

Request 4

The subscriber with the given IMEI Range is deleted. The subscriber exists.

Request URL

```
DELETE {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111
```

Request Content

None

Response 4

The request is successful.

HTTP Status Code

204

Response Content

None

5.2 Subscriber Profile Field Commands

Table 18: Summary of Subscriber Profile Field Commands

Command	Description	Keys	Command Syntax
Add Field Value	Adds a value to the specified field. This operation does not affect any pre-existing values for the field	MSISDN, IMSI, IMEI,NAI or AccountId	POST {baseURI}/msr/sub/*keyName*/*keyValue*/field/*fieldName*/*fieldValue*
Get Field	Retrieve the values for the specified field		GET {baseURI}/msr/sub/*keyName*/*keyValue*/field/*fieldName*
Get Field Value	Retrieve the single value for the specified field (if set as specified)		GET {baseURI}/msr/sub/*keyName*/*keyValue*/field/*fieldName*/*fieldValue*
Update Field Value	Updates field to the specified value		PUT {baseURI}/msr/sub/*keyName*/*keyValue*/field/*fieldName*/*fieldValue*
Update Multiple Fields	Update multiple fields to the specified values		PUT {baseURI}/msr/sub/*keyName*/*keyValue*/multipleFields/*fieldName*1fieldValue1/*fieldName2*/*fieldValue2*/…
Delete Field	Delete all the values for the specified field		DELETE {baseURI}/msr/sub/*keyName*/*keyValue*/field/*fieldName*
Delete Field Value	Delete a value for the specified field		DELETE {baseURI}/msr/sub/*keyName*/*keyValue*/field/*fieldName*/*fieldValue*

5.2.1 Add Field Value

Description

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

This operation can only be run for the fields defined as multi-value field in the subscriber entity configuration. Any pre-existing values for the field are not affected.

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

If an added value exists, the request fails.

If the field to which the value is added does not exist, it is created.

The *fieldValue* is case-sensitive. An attempt to add the value a to current field value of a;b;c fails, but an attempt to add the value A is successful and result in the field value is a;b;c;A

Prerequisites

A subscriber with the key of the *keyName* or *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* added must not be present in the field.

Request URL

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId
- *keyValue*: Corresponding key field value assigned to *keyName*
- *fieldName*: A user defined field in the subscriber profile
- *fieldValue*: Corresponding field value assigned to *fieldName*

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

Request Content

None.

Response Content

None.

Table 19 Add Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
200		Successfully added field values
400	MSR4005	Field does not support multiple values
400	MSR4051	Invalid value for a field

HTTP Status Code	Error Code	Description
400	MSR4056	Field is not updatable
400	MSR4064	Occurrence constraint violation
400	MSR4066	Field value already exists
404	MSR4001	Subscriber is not found
404	MSR4002	Subscriber field is not defined

Add Field Value Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

200

Response Content

None

Request 2

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

200

Response Content

None

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*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 has three MSISDNs, 15141234567, 14161112222, and 14505556666

HTTP Status Code

200

Response Content

None

Request 5

A request is made to add the value EveningPass to the Entitlement field. The Entitlement field is a valid multi-value field. The Entitlement field contains a value NightPass. With this request EveningPass value added to the Entitlement field.

Request URL

```
POST {baseURI}/msr/sub/IMEI/74747474747474/field/Entitlement/EveningPass
```

Request Content

None

Response 5

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

HTTP Status Code

200

Response Content

None

Request 6

A request is made to add the value EveningPass to the Entitlement field. The Entitlement field is a valid multi-value field. The Entitlement field contains a value NightPass. With this request EveningPass value added to the Entitlement field.

Request URL

```
POST {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111/field/Entitlement/EveningPass
```

Request Content

None

Response 6

The request is successful, and the value was added to the Custom1 field.

HTTP Status Code

200

Response Content

None

5.2.2 Get Field

Description

This operation retrieves the values for the specified fields for the subscriber identified by the specified *keyName*and *keyValue*.

Prerequisites

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

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

Request URL

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

- *keyName*: A key field in the subscriber profile

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

Request Content

None.

Response Content

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

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

- *fieldName*: The requested user defined field in the subscriber profile
- *fieldValueX*: Corresponding field value assigned to *fieldName*

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

Table 20 Get Field 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

Get Field Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

200

Response Content

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

Request 2

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

200

Response Content

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

Request 3

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

Request URL

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

Request Content

None

Response 3

The request fails and an error is returned.

HTTP Status Code

400

Response Content

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

Request 4

This is an example of request content for UDR for DSR based EIR solution with IMEI as key.

Request URL

```
GET {baseURI}/msr/sub/IMEI/98765439876543/field/BL
```

Request Content:

None

Response 4

The request is successful, and the subscriber field was retrieved.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<field name="BL">0</field>
</subscriber>
```

Request 5

This is an example of request content with IMEI range for UDR for DSR based EIR solution with IMEI as key:

Request URL

```
GET {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111/field/BL
```

Request Content:

None

Response 5

The request is successful, and the subscriber field was retrieved.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
<field name="BL">0</field>
</subscriber>
```

5.2.3 Get Field Value

Description

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

For a request where the presence of multiple values for a multi-value field is requested, a match is only considered to have been made if the requested values form a subset of the values stored in the profile. That is, if

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

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

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

Prerequisites

- A subscriber with the key of the *keyName*/*keyValue* supplied must exist.
- The requested field *fieldName* must be a valid field in the subscriber profile.
- The requested field must contain the values supplied in the *fieldValue*.

Request URL

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId

- *keyValue*: Corresponding key field value assigned to *keyName*
- *fieldName*: A user defined field in the subscriber profile
- *fieldValue*: Corresponding field value assigned to *fieldName*

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

Request Content

None.

Response Content

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

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

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

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

Table 21 Get Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
200		Requested field exists for subscriber with given value

HTTP Status Code	Error Code	Description
404	MSR4053	Subscriber and field exist, but values do not match
404	MSR4001	Subscriber does not exist
404	MSR4002	Subscriber field is not defined

Get Field Value 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>
```

This is an example of UDR for DSR based EIR solution.

Request 3

A request is made to get the GL field with the value 0. The field exists and has the specified value.

Request URL

```
GET {baseURI}/msr/sub/IMEI/98765439876543/field/GL/0
```

Request Content

None

Response 3

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="GL">0</field>
</subscriber>
```

This is an example of request with IMEI range for UDR for DSR based EIR solution.

Request 4

A request is made to get the GL field with the value 0. The field exists and has the specified value.

Request URL

```
GET {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111/field/GL/0
```

Request Content

None

Response 4

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="GL">0</field>
</subscriber>
```

5.2.4 Update Field

Description

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

This operation replaces (sets) the value of the field, which means that any existing values for the field are deleted first. For multi-value fields, all previous values are erased and the 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 in the subscriber profile
  - Value is either IMSI, MSISDN, IMEI, NAI, or AccountId
- *keyValue*: Corresponding key field value assigned to *keyName*
- *fieldName*: A user defined field in the subscriber profile

  NOTE: A field name cannot be for a key value—that is, IMSI, MSISDN, IMEI, NAI, or AccountId
- *fieldValue*: Corresponding field value assigned to *fieldName*

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

Request Content

None.

Response Content

None.

Table 22 Update Field Response Status/Error Codes

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

Update Field Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

201

Response Content

None

Request 2

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

201

Response Content

None

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

201

Response Content

None

Request 4

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

Request URL

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

Request Content

None

Response 4

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

HTTP Status Code

201

Response Content

None

Request 5

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

Request URL

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

Request Content

None

Request 6

A request is made to update the value of the GLfield to 1 from 0.

This is an example of UDR for DSR based EIR solution.

Request URL

```
PUT {baseURI}/msr/sub/IMEI/98765439876543/field/GL/1
```

Request Content

None

Response 6

The request is successful, and the GL field was updated.

HTTP Status Code

201

Response Content

None

Request 7

A request is made to update the value of the GLfield to 1 from 0.

This is an example of request with IMEI range for UDR for DSR based EIR solution.

Request URL

```
PUT {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111/field/GL/1
```

Request Content

None

Response 7

The request is successful, and the GL field was updated.

HTTP Status Code

201

Response Content

None

Response 5

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

HTTP Status Code

201

Response Content

None

5.2.5 Update Multiple Fields

Description

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

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

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

Prerequisites

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

The fields *fieldNameX* must all be valid fields in the subscriber profile.

Request URL

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, IMEI, NAI, or AccountId
- *keyValue*: Corresponding key field value assigned to *keyName*
- *fieldNameX*: A user defined field in the subscriber profile

  NOTE: A field name cannot be for a key value—that is, IMSI, MSISDN, IMEI, NAI, or AccountId
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

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

Request Content

None.

Response Content

None.

Table 23 Update Multiple Fields Response Status/Error Codes

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

Update Multiple Fields 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

Request 3

A request is made to update the WL and GL fields both to zero from value 1.

This is is an example of UDR for DSR based EIR solution.

Request URL

```
PUT {baseURI}/msr/sub/IMEI/98765439876543/multipleFields/GL/0/WL/0.
```

Request Content

None

Response 3

The request is successful, and the WL and GL fields were all updated.

HTTP Status Code

201

Response Content

None

Request 4

A request is made to update the WL and GL fields both to zero from value 1.

This is an example of request with IMEI range for UDR for DSR based EIR solution.

Request URL

```
PUT {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111/multipleFields/GL/0/WL/0.
```

Request Content

None

Response 4

The request is successful, and the WL and GL fields were all updated.

HTTP Status Code

201

Response Content

None

5.2.6 Delete Field

Description

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

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

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

If the deleted field is mandatory, and is defined as having a default value, then the field is not removed, but has the default value assigned.

If a key (IMSI, MSISDN, NAI, or AccountId) field is deleted for a subscriber, the subscriber must still have at least one key type and value remaining or the request fails.

Prerequisites

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

Request URL

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

- *keyName*: A key field in the subscriber profile

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

Request Content

None.

Response Content

None.

Table 24 Delete Field Response Status/Error Codes

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

Delete Field Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

The request is successful, and the field was deleted.

HTTP Status Code

204

Response Content

None

Request 2

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

204

Response Content

None

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

400

Response Content

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

Request 4

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

Request URL

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

Request Content

None

Response 4

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

HTTP Status Code

204

Response Content

None

Request 5

A request is made to delete the Custom1 field. The field is a valid subscriber profile field and contains multiple values.

Request URL

```
DELETE {baseURI}/msr/sub/IMEI/74747474747474/field/Custom1
```

Request Content

None

Response 5

The request is successful, and the field was deleted.

HTTP Status Code

204

Response Content

None

Request 6

A request is made to delete the IMSI field. The field is a valid subscriber profile field and contains multiple values.

This is an example of UDR for DSR based EIR Soultion.

Request URL

```
DELETE {baseURI}/msr/sub/IMEI/98765439876543/field/IMSI
```

Request Content

None

Response 6

The request is successful, and all the IMSI fields were deleted.

HTTP Status Code

204

Response Content

None

Request 7

A request is made to delete the IMEI field.

This is an example of request with IMEI range for UDR for DSR based EIR Soultion.

Request URL

```
DELETE {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111/field/IMEI
```

Request Content

None

Response 7

HTTP Status Code

400

Response Content

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

5.2.7 Delete Field Value

Description

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

This operation can only be run for the fields defined as multi-value field in the subscriber entity configuration.

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

If all values are removed, the field is removed from the subscriber profile (that is, the XML element is not present).

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

Prerequisites

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

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

Request URL

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

- *keyName*: A key field in the subscriber profile

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

  NOTE: A field name cannot be for a key value—IMSI, MSISDN, IMEI, NAI, or AccountId
- *fieldValue*: Corresponding field value assigned to *fieldName*

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

Request Content

None.

Response Content

None.

Table 25 Delete Field Value Response Status/Error Codes

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

Delete Field Value 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=*error>
```

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

204

Response Content

None

Request 4

A request is made to delete one IMSI. The IMSI field is a multi-value field. The field exists and contains the specified values.

This is an example for UDR for DSR based EIR solution.

Request URL

```
DELETE {baseURI}/msr/sub/IMEI/98765439876543/field/IMSI/987654398765430
```

Request Content

None

Response 4

HTTP Status Code

204

Response Content

None

Request 5

A request is made to delete IMEI range field which is the only key.

This is an example using IMEI range for UDR for DSR based EIR solution.

Request URL

```
DELETE {baseURI}/msr/sub/IMEI/44111111111111/IMEI/44441111111111/field/IMEI/44111111111111
```

Request Content

None

Response 5

HTTP Status Code

400

Response Content

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

5.3 Subscriber Opaque Data Commands

Table 26 commands perform opaque data operations. They can be used on entities defined as either opaque or transparent. The opaque data operation operates on the entity at the XML blob level. The content of the entity is set, returned, or deleted.

Table 26: Summary of Subscriber Opaque Data Commands

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

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

5.3.1 Set Opaque Data

Description

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

The opaque data is provided in the request content.

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

Prerequisites

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

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

Request URL

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

- *keyName*: A key field in the subscriber profile

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

  Value is either quota, state, or dynamicquota

Request Content

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

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

- *opaqueDataType*: A user defined type/name for the opaque data

  Value is either quota, state, or dynamicquota
- *cdataFieldValue*: Contents of the XML data blob

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

Response Content

None.

Table 27 Set Opaque Data Response Status/Error Codes

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

Set Opaque Data 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 has an existing State entity.

Request URL

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

Request Content:

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

Response 2

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

HTTP Status Code

201

Response Content

None

5.3.2 Get Opaque Data

Description

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

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

Prerequisites

- A subscriber with the key of the *keyName*/*keyValue* supplied must exist.
- The *opaqueDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.
- The opaque data of the *opaqueDataType* must exist for the subscriber.

Request URL

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

- *keyName*: A key field in the subscriber profile

Value is either IMSI, MSISDN, NAI, or AccountId

- *keyValue*: Corresponding key field value assigned to *keyName*
- *opaqueDataType*: A user defined type/name for the opaque data
  Value is either quota, state, or dynamicquota

Request Content

None.

Response Content

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

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

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

Table 28 Get Opaque Data Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	NA	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

Get Opaque Data 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>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>
```

Request 2

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

200

Response Content

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

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

5.3.3 Delete Opaque Data

Description

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

Only one opaque data type can be deleted per request.

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

Prerequisites

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

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

Request URL

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

- *keyName*: A key field in the subscriber profile

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

Request Content

None.

Response Content

None.

Table 29 Delete Opaque Data Response Status/Error Codes

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

Delete Opaque Data Examples

Request 1

A request is made to delete the quota opaque data.

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

204

Response Content

None

Request 2

A request is made to delete the state opaque data.

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

204

Response Content

None

Request 3

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

Request URL

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

Request Content

None

Response 3

The request is successful, although state opaque data was not deleted.

HTTP Status Code

204

Response Content

None

5.4 Subscriber Data Row Commands

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

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

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

Table 30: Summary of Subscriber Data Row Commands

Command	Description	Keys	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 in data of the specified type		`DELETE {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue`
	Delete data row in data of the specified type and instance identifier		`DELETE {baseURI}/msr/sub/keyName/keyValue/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue`

5.4.1 Set Row

Description

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

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

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

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

The *rowIdValue* is case-sensitive. If a row exists called DayPass then an attempt to update an existing row called DAYPASS is successful, and two rows called DayPass and DAYPASS are present

If the transparent entity specified in *entityName* does not exist for the subscriber, it is created

Prerequisites

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

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

Request Content

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

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

  NOTE: The *rowValue* is the same format as a Quota entity, just containing a single row, the added row.

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

Response Content

None.

Table 31 Set Row Response Status/Error Codes

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

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

Set Row 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>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>
```

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=*error>
```

Request 4

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

Request URL

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

Request Content:

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

Response 4

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

HTTP Status Code

201

Response Content

None

Request 5

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

Request URL

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

Request Content:

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

Response 5

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

HTTP Status Code

201

Response Content

None

5.4.2 Get Row

Description

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

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

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

The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to get a row called DayPass is successful, but an attempt to get a row called DAYPASS fails

Prerequisites

- A subscriber with the key of the *keyName*/*keyValue* supplied must exist.
- The *transparentDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier in the transparent data exists for the subscriber.

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

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

Request Content

None.

Response Content

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

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

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

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

Table 32 Get Row Response Status/Error Codes

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

Get Row 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

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*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>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>
  <quota name="Weekend">
    <cid>7682364872564782343</cid>
    <time>32</time>
    <totalVolume>250</totalVolume>
    <inputVolume>4570</inputVolume>
    <outputVolume>11230</outputVolume>
    <serviceSpecific>29</serviceSpecific>
    <nextResetTime>2010-06-01T16:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</subscriber>
```

Request 4

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

Request URL

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

Request Content

None

Response 4

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

HTTP Status Code

200

Response Content

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

```
    <InterimReportingInterval>200</InterimReportingInterval>
    <Duration>20</Duration>
   </DynamicQuota>
  </definition>]]>
  </data>
</subscriber>
```

5.4.3 Delete Row

Description

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

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

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

The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to delete a row called DayPass is successful, but an attempt to delete a row called DAYPASS fails

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

Prerequisites

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

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

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

Request Content

None.

Response Content

None.

Table 33 Delete Row Response Status/Error Codes

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

Delete Row 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

400

Response Content

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

Request 4

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

Request URL

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

Request Content

None

Response 4

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

HTTP Status Code

204

Response Content

None

Request 5

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

Request URL

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

Request Content

None

Response 5

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

HTTP Status Code

204

Response Content

None

5.5 Subscriber Data Row Field Commands

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

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

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

Table 34: Summary of Subscriber Data Row Field Commands

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

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

5.5.1 Get Row Field

Description

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

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

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

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

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

Prerequisites

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

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

A data row with the given identifier in the transparent data exists for the subscriber.

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the subscriber profile

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

Request Content

None.

Response Content

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

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

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

Table 35 Get Row Field Response Status/Error Codes

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

Get Row Field Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

200

Response Content

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

Request 2

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

200

Response Content

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

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

200

Response Content

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

5.5.2 Get Row Field Value

Description

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

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

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

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

The *fieldValue* is case-sensitive. An attempt to get the value Data from a current field value of Data is successful, but an attempt to get the value DATA fails

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 in the transparent data exists for the subscriber.
- The field name specified must be a valid field for the Entity as defined in the SEC.
- The field value in *fieldValue* must match the specified value in the request.

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

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

Request Content

None.

Response Content

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

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

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

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

Table 36 Get Row Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
200		Requested data row field/value exists for subscriber
404	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

Get Row Field Value 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=*error>
```

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

200

Response Content

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

Request 4

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

Request URL

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

Request Content

None

Response 4

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

HTTP Status Code

200

Response Content

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

Request 5

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

Request URL

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

Request Content

None

Response 5

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

HTTP Status Code

200

Response Content

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

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

5.5.3 Update Row Field

Description

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

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

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

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

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

Prerequisites

- A subscriber with the key of the *keyName*/*keyValue* supplied must exist.
- The *transparentDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier in the transparent data exists for the subscriber.
- The field name specified must be a valid field for the Entity as defined in the SEC. The field must be updatable.

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

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

- *fieldName*: A user defined field in the transparent data row
- *fieldValue*: Corresponding field value assigned to *fieldName*

Request Content

None.

Response Content

None.

Table 37 Update Row Field Response Status/Error Codes

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

Update Row Field 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=*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=*error>
```

Request 4

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

Request URL

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

Request Content

None

Response 4

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

HTTP Status Code

201

Response Content

None

5.5.4 Delete Row Field

Description

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

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

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

If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and the row or field data is not deleted.

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

If the deleted field is mandatory, and is defined as having a default value, then the field is not removed, but has the default value assigned.

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

Prerequisites

- A subscriber with the key of the *keyName*/*keyValue* supplied must exist.
- The *transparentDataType* must reference a valid Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier in the transparent data exists for the subscriber.
- The field name specified must be a valid field for the Entity as defined in the SEC. The field must be updatable.

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

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

Request Content

None.

Response Content

None.

Table 38 Delete Row Field Response Status/Error Codes

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

Delete Row Field 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=*error>
```

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

204

Response Content

None

Request 4

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

Request URL

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

Request Content

None

Response 4

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

HTTP Status Code

204

Response Content

None

5.6 Subscriber Data Field Commands

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

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

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

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

Table 39: Summary of Subscriber Data Field Commands

Command	Description	Keys	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 in transparent data of the specified type		DELETE {baseURI}/msr/sub/*keyName*/*keyValue*/data/*transparentDataType*/*fieldName*

5.6.1 Set Data Field

Description

This operation creates or updates a 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 field matches the requested *fieldName*, then the update request fails.

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

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

If the transparent entity specified in *entityName* does not exist for the subscriber, it is created

Prerequisites

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

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

Request URL

Format 1

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

Format 2

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

- *keyValue*: Corresponding key field value assigned to *keyName*
- *transparentDataType*: A user defined type/name for the transparent data

  Value is state for the State transparent data

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

Request Content

None.

Response Content

None.

Table 40 Set Data Field Response Status/Error Codes

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

Set Data Field 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=*error>
```

5.6.2 Get Data Field

Description

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

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

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

The transparent data field is specified in *fieldName*.

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

Prerequisites

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

Request URL

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

- *keyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

- *keyValue*: Corresponding key field value assigned to *keyName*
- *transparentDataType*: A user defined type/name for the transparent data

  Value is state for the State transparent data

- *fieldName*: A user defined field in the transparent data

  For the State entity this corresponds to a property in the entity

Request Content

None.

Response Content

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

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

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

  Value is state for the State transparent data

- *cdataFieldValue*: Contents of the XML data blob, with the field from the transparent data

Table 41 Get Data Field Response Status/Error Codes

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

Get Data Field 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

400

Response Content

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

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

200

Response Content

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

5.6.3 Delete Data Field

Description

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

The field identifier is specified in *fieldName*.

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

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

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

Prerequisites

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

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

Request URL

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

- *keyName*: A key field in the subscriber profile

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

  Value is state for the State transparent data
- *fieldName*: A user defined field in the transparent data

  For the State entity this corresponds to a property in the entity

Request Content

None.

Response Content

None.

Table 42 Delete Data Field Response Status/Error Codes

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

Delete Data Field 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

400

Response Content

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

5.7 Subscriber Special Operation Commands

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

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

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

Table 43: Summary of Subscriber Special Operation Commands

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

5.7.1 Reset Quota

Description

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

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

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

The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to reset a quota row called "DayPass" is successful, but an attempt to reset a quota row called DAYPASS fails.

When a Quota instance is reset using the `Reset Quota` command, each resettable field is set to its defined reset value. If the field does not exist, it is not created. But, if a resettable field does not exist, and the field has a default value, then the field is created with the default value.

Prerequisites

- A subscriber with the key of the *keyName*/*keyValue* supplied must exist.
- The Quota transparent data must exist for the subscriber.
- The specified Quota row must exist in the Quota transparent data.

Request URL

Without Instance Identifier

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

With Instance Identifier

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

- *keyName*: A key field in the subscriber profile

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

  Value is quota for the Quota transparent data
- *rowIdValue*: The row name value that identifies the row in the transparent data blob
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance

  Value is cid for the Quota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

Request Content

None.

Response Content

None.

Table 44 Reset Quota Response Status/Error Codes

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

Reset Quota 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

400

Response Content

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

400

Response Content

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

Chapter 6. Pool Provisioning

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

Provisioning clients can create, retrieve, modify, and delete pool data. Pool data is accessed through the *PoolId* value associated with the pool.

Table 45: Summary of Pool Profile Commands

Command	Description	Keys	Command Syntax
Create Pool	Creates a pool or pool profile	NA	`POST {baseURI}/msr/pool`
Get Pool	Gets pool profile data	*PoolId*	`GET {baseURI}/msr/pool/poolId`
Update Pool	Replaces an existing pool profile		`PUT {baseURI}/msr/pool/poolId`
Delete Pool	Deletes all pool profile data and all opaque data associated with the pool		`DELETE {baseURI}/msr/pool/poolId`

6.1.1 Create Pool

Description

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

Unlike other pool commands, the key value (*PoolId*) is not specified in the URL. Request content includes *poolId*, and field-value pairs, all as specified in the subscriber entity configuration.

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

If the PSO feature is enabled and the *PoolId* falls in a range that is maintained by a different UDR instance, then the pool is created as a Non Pool Host UDR pool (remote pool); otherwise the pool is created as a Pool Host UDR pool.

If the PSO feature is enabled, a pool cannot be provisioned with the Type field on a Non Pool Host UDR system.

Prerequisites

A pool with the supplied *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 pool.

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

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

  Values: 1 to 999999999999999999999

- *fieldNameX*: A user defined field in the pool profile
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

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

Response Content

None.

Table 46 Create Pool Response Status/Error Codes

HTTP Status Code	Error Code	Description
201		Successfully created
400	MSR4000	Invalid content request data supplied
400	MSR4003	A key is detected 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
400	MSR4070	Operation not allowed

Create Pool 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 exists with the given *PoolId*.

Request URL

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

Request Content:

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

Response 2

The request fails. The error code indicates the *PoolId* exists.

HTTP Status Code

400

Response Content

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

Request 3

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

Request URL

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

Request Content:

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

Response 3

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

HTTP Status Code

400

Response Content

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

6.1.2 Get Pool

Description

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

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

Prerequisites

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

Request URL

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

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

  Values: 1 to 999999999999999999999

Request Content

None.

Response Content

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

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

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

  Values: 1 to 999999999999999999999

- *fieldNameX*: A user defined field in the pool profile
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

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

Table 47 Get Pool Response Status/Error Codes

HTTP Status Code	Error Code	Description
200		Successfully located the pool

HTTP Status Code	Error Code	Description
404	MSR4001	Could not find the pool by *PoolId*

Get Pool 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

400

Response Content

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

6.1.3 Update Pool

Description

This operation replaces an existing subscriber profile, for the pool identified by *poolId*.

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

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

If the PSO feature is enabled, a pool cannot be updated with the Type field on a Non Pool Host UDR system.

Prerequisites

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

Request URL

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

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

  Values: 1 to 999999999999999999999

Request Content

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

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

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

  Values: 1 to 999999999999999999999
- *fieldNameX*: A user defined field in the pool profile
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

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

Response Content

None.

Table 48 Update Pool Response Status/Error Codes

HTTP Status Code	Error Code	Description
204		The pool data was replaced successfully
400	MSR4000	Invalid content/payload
400	MSR4000	The *PoolId* supplied in URL and request content do not match

HTTP Status Code	Error Code	Description
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
400	MSR4070	Operation not allowed

Update Pool 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="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>
```

Response 1

The request is successful, and the pool was updated.

HTTP Status Code

204

Response Content

None

Request 2

A pool is updated with Type field set. The pool exists.

Request URL

```
PUT {BaseURI}/msr/pool/100000
```

Request Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <field name="PoolId">100000</field>
```

```
  <field name="BillingDay">5</field>
  <field name="Type">Enterprise</field>
  <field name="Entitlement">Weekpass</field>
  <field name="Entitlement">Daypass</field>
  <field name="Custom15">allo</field>
</pool>
```

Response 1

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

HTTP Status Code

400

Response Content

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

6.1.4 Delete Pool

Description

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

Prerequisites

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

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

Request URL

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

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

  Values: 1 to 9999999999999999999

Request Content

None.

Response Content

None.

Table 49 Delete Pool 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

Delete Pool Examples

Request 1

The pool with the given *PoolId* is deleted. The pool exists and does not have 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=*error>
```

6.2 Pool Profile Field Commands

Table 50: Summary of Pool Profile Field Commands

Command	Description	Keys	Command Syntax
Add Field Value	Adds a value to the specified field. This operation does not affect any pre-existing values for the field	*PoolId*	POST {baseURI}/msr/pool/*poolId*/field/*fieldName*/*fieldValue*
Get Field	Retrieve values 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*

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

6.2.1 Add Field Value

Description

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

This operation can only be run for the fields defined as multi-value field in the subscriber entity configuration. Any pre-existing values for the field are not affected.

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

If an added value exists, the request fails.

If the field to which the value is added does not exist, it is created.

The *fieldValue* is case-sensitive. An attempt to add the value a to current field value of a;b;c fails, but an attempt to add the value A is successful and result in the field value is 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* must not be present in the field.

Request URL

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

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

  Values: 1 to 999999999999999999999

- *fieldName*: A user defined field in the pool profile
- *fieldValue*: Corresponding field value assigned to *fieldName*

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

Request Content

None.

Response Content

None.

Table 51 Add Field Value Response Status/Error Codes

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

Add Field Value Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

200

Response Content

None

Request 2:

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

200

Response Content

None

6.2.2 Get Field

Description

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

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

Prerequisites

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

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

Request URL

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

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

  Values: 1 to 999999999999999999999
- *fieldName*: A user defined field in the pool profile

Request Content

None.

Response Content

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

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

- *fieldName*: A user defined field in the pool profile
- *fieldValueX*: Corresponding field value assigned to *fieldName*

Table 52 Get Field Response Status/Error Codes

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

Get Field Examples

Request 1

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

Request URL

```
GET {BaseURI}/msr/pool/100000/field/Entitlement
```

Request Content

None

Response 1

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

HTTP Status Code

200

Response Content

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

6.2.3 Get Field Value

Description

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

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

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

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

Prerequisites

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

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

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

Request URL

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

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

  Values: 1 to 999999999999999999999

- *fieldName*: A user defined field in the pool profile
- *fieldValue*: Corresponding field value assigned to *fieldName*

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

Request Content

None.

Response Content

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

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

- *fieldName*: A user defined field in the pool profile
- *fieldValueX*: Corresponding field value assigned to *fieldName*

Table 53 Get Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
200		Requested field exists for pool
404	MSR4053	Pool and field exist, but values do not match
404	MSR4001	Pool is not found
404	MSR4002	Pool field is not defined

Get Field Value Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

200

Response Content

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

Request 2

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

Request URL

```
GET {baseURI}/msr/pool/300000/field/Entitlement/DayPass;HighSpeedData
```

Request Content

None

Response 2

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

HTTP Status Code

200

Response Content

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

6.2.4 Update Field

Description

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

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

This command cannot be used to update the *PoolId*.

Prerequisites

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

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

Request URL

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

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

  Values: 1 to 999999999999999999999

- *fieldName*: A user defined field in the pool profile
- *fieldValue*: Corresponding field value assigned to *fieldName*

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

Request Content

None.

Response Content

None.

Table 54 Update Field Response Status/Error Codes

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

Update Field Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

201

Response Content

None

6.2.5 Update Multiple Fields

Description

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

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

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

All fields are updated at once in the database. 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 returns an error). For example, if the third field fails validation, then none of the fields are updated.

This command cannot be used to update the *PoolId*.

Prerequisites

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

The fields *fieldNameX* must all be valid fields in the pool profile.

Request URL

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

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

  Values: 1 to 9999999999999999999

- *fieldNameX*: A user defined field in the pool profile
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

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

Request Content

None.

Response Content

None.

Table 55 Update Multiple Fields Response Status/Error Codes

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

Update Multiple Fields Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

201

Response Content

None

6.2.6 Delete Field

Description

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

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

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

This command cannot be used to delete the *PoolId*.

If the deleted field is mandatory, and is defined as having a default value, then the field is not removed, but has the default value assigned.

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- The requested field *fieldName* must be a valid field in the pool profile.

Request URL

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

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

  Values: 1 to 999999999999999999999

- *fieldName*: A user defined field in the subscriber profile

Request Content

None.

Response Content

None.

Table 56 Delete Field Response Status/Error Codes

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

Delete Field 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=*error>
```

6.2.7 Delete Field Value

Description

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

This operation can only be run for the fields defined as multi-value field in the subscriber entity configuration.

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

If all values are removed, the field is removed from the pool profile (the XML element is not present).

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

Prerequisites

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

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

Request URL

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

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

  Values: 1 to 999999999999999999999

- *fieldName*: A user defined field in the pool profile
- *fieldValue*: Corresponding field value assigned to *fieldName*

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

Request Content

None.

Response Content

None.

Table 57 Delete Field Value Response Status/Error Codes

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

Delete Field Value Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

204

Request Content

None

Request 2

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

204

Response Content

None

6.3 Pool Opaque Data Commands

Table 58: Summary of Pool Opaque Data Commands

Command	Description	Keys	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`*

6.3.1 Set Opaque Data

Description

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

The opaque data is provided in the request content.

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

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

Prerequisites

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

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

Request URL

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

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

  Values: 1 to 999999999999999999999

- *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

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

Response Content

None.

Table 59 Set Opaque Data Response Status/Error Codes

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

Set Opaque Data 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 has an existing PoolState entity.

Request URL

```
PUT {baseURI}/msr/pool/100002/data/poolstate
```

Request Content:

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

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

Response 2

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

HTTP Status Code

201

Response Content

None

6.3.2 Get Opaque Data

Description

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

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

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

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- The *opaqueDataType* must reference a valid pooled Entity in the Interface Entity Map table in the SEC.
- The opaque data of the *opaqueDataType* must exist for the pool.

Request URL

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

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

  Values: 1 to 999999999999999999999

- *opaqueDataType*: A user defined type/name for the opaque data

  Value is either poolquota, poolstate, or pooldynamicquota

Request Content

None.

Response Content

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

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

```
<![CDATA[
cdataFieldValue
]]>
  </data>
</pool>
```

- *opaqueDataType*: A user defined type/name for the opaque data

  Value is either poolquota, poolstate, or pooldynamicquota

- *cdataFieldValue*: Contents of the XML data blob

Table 60 Get Opaque Data Response Status/Error Codes

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

Get Opaque Data 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>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>
```

Request 2

A request is made to get the poolstate opaque data for a pool.

Request URL

```
GET {baseURI}/msr/pool/100004/data/poolstate
```

Request Content

None

Response 2

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

HTTP Status Code

200

Response Content

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

Request 3

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

Request URL

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

Request Content

None

Response 3

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

HTTP Status Code

204

Response Content

None

6.3.3 Delete Opaque Data

Description

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

Only one opaque data type can be deleted per request.

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

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

Prerequisites

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

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

Request URL

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

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

  Values: 1 to 999999999999999999999

- *opaqueDataType*: A user defined type/name for the opaque data

  Value is either poolquota, poolstate, or pooldynamicquota

Request Content

None.

Response Content

None.

Table 61 Delete Opaque Data Response Status/Error Codes

HTTP Status Code	Error Code	Description
204		Opaque data was successfully deleted
404	MSR4001	Pool is not found

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

Delete Opaque Data Example

Request 1

A request is made to delete the pool*dynamicquota* opaque data.

Request URL

```
DELETE {baseURI}/msr/pool/500005/data/pool dynamicquota
```

Request Content

None

Response 1

The request is successful, and the Pool*Dynamicquota* 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 the PoolState opaque data was not deleted.

HTTP Status Code

204

Response Content

None

6.4 Pool Data Row Commands

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

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

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

Table 62: Summary of Pool Data Row Commands

Command	Description	Keys	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 in data of the specified type		DELETE {baseURI}/msr/pool/*poolId*/data/*transparentDataType*/*rowIdValue*
	Delete data row in data of the specified type and instance identifier		DELETE {baseURI}/msr/pool/*poolId*/data/*transparentDataType*/*rowIdValue*/row/*instanceFieldName*/*instanceFieldValue*

6.4.1 Set Row

Description

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

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

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

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

The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to update an existing row called DAYPASS is successful, and two rows called DayPass and DAYPASS are present

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

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

Prerequisites

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

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

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

  Values: 1 to 999999999999999999999

- *transparentDataType*: A user defined type/name for the transparent data
  - Value is *poolquota* for the *Poolquota* transparent data
  - Value is pooldynamicquotafor the Pool*Dynamicquota* transparent data
- *rowIdValue*: The row name value that identifies the row in the data blob
- There is not a *rowIdValue* for State transparent data
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance
  - Value is cid or Type for the *Poolquota* transparent data
  - Value is InstanceId or Type for the Pool*Dynamicquota* transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

Request Content

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

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

  NOTE: The *rowValue* is the same format as a *Poolquota* entity, just containing a single row, the row added.

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

Response Content

None.

Table 63 Set Row Response Status/Error Codes

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

HTTP Status Code	Error Code	Description
404	MSR4002	Field is not defined for this data type
404	MSR4049	Data type is not defined

Set Row 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>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>
```

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>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>
```

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 pool*dynamicquota* transparent data for a pool. The data row identifier field value is AggregateLimit. The AggregateLimit data row exists in the Pool*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/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

6.4.2 Get Row

Description

This operation retrieves a transparent data row for the pool identified by the *poolId*. The data row identifier is specified in *rowIdValue* .

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

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

The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to get a row called DayPass is successful, but an attempt to get a row called DAYPASS fails

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

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- The *transparentDataType* must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier in the transparent data exists 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 to 22 digits in length

  Values: 1 to 999999999999999999999

- *transparentDataType*: A user defined type/name for the transparent data
  - Value is poolquota for the Poolquota transparent data
  - Value is pooldynamicquota for the Pool*Dynamicquota* transparent data
- *rowIdValue*: The row name value that identifies the row in the transparent data blob
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance
  - Value is cid or Type for the *Poolquota* transparent data
  - Value is InstanceId or Type for the Pool*Dynamicquota* 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 Pool*Dynamicquota* transparent data
- *cdataRowValue*: Contents of the XML data blob, with the row data

Table 64 Get Row Response Status/Error Codes

HTTP Status Code	Error Code	Description
200		Requested data row exists for pool
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found
404	MSR4059	Data row does not exist
204		No Content returned

Get Row Examples

Request 1

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

Request URL

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

Request Content

None

Response 1

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

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<pool>
  <data name="poolquota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
```

```
  <quota name="PQ1">
    <cid>9223372036854775807</cid>
    <time>1</time>
    <totalVolume>0</totalVolume>
    <inputVolume>0</inputVolume>
    <outputVolume>0</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-12T16:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]>
  </data>
</pool>
```

Request 2

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

Request URL

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

Request Content

None

Response 2

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

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*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>76823648725647823343</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 pool*dynamicquota* transparent data for a pool with the *InstanceId* value of 11223344. The Pool*Dynamicquota* 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 Pool*Dynamicquota* transparent data row requested is returned.

HTTP Status Code

200

Response Content

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

```
  <Dynamicquota name="PDQ1">
    <Type>topup</Type>
    <InstanceId>11223344</InstanceId>
    <Priority>4</Priority>
    <InterimReportingInterval>200</InterimReportingInterval>
    <Duration>20</Duration>
  </DynamicQuota>
</definition>]]>
  </data>
</pool>
```

6.4.3 Delete Row

Description

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

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

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

The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to delete a row called DayPass is successful, but an attempt to delete a row called DAYPASS fails

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

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

Prerequisites

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

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

Request URL

Without Instance Identifier

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

With Instance Identifier

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

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

  Values: 1 to 999999999999999999999

- *transparentDataType*: A user defined type/name for the transparent data
  - Value is poolquota for the Poolquota transparent data
  - Value is pooldynamicquota for the Pool*Dynamicquota* transparent data
- *rowIdValue*: The row name value that identifies the row in the transparent data blob
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance
  - Value is cid or Type for the *Poolquota* transparent data
  - Value is InstanceId or Type for the Pool*Dynamicquota* transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

Request Content

None.

Response Content

None.

Table 65 Delete Row Response Status/Error Codes

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

Delete Row 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 pool*dynamicquota* transparent data. The PDQ1 data row exists in the Pool*Dynamicquota* 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 pool*dynamicquota* transparent data was deleted.

HTTP Status Code

204

Response Content

None

6.5 Pool Data Row Field Commands

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

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

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

Table 66: Summary of Pool Data Row Field Commands

Command	Description	Keys	Command Syntax
Get Row Field	Retrieve values for the specified field	*PoolId* and Row	`GET {baseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/fieldname`

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

6.5.1 Get Row Field

Description

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

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

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

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

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

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

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- The *transparentDataType* must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier in the transparent data exists for the pool.
- The field name specified must be a valid field for the Entity as defined in the SEC.

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 to 22 digits in length

  Values: 1 to 999999999999999999999

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

  Value is poolquota for the Poolquota transparent data

  Value is pooldynamicquota for the *PoolDynamicquota* transparent data

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

Request Content

None.

Response Content

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

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

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

Table 67 Get Row Field 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

HTTP Status Code	Error Code	Description
404	MSR4059	Data row does not exist
404	MSR4065	Field is not set
204		No Content returned

Get Row Field 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 pool*dynamicquota* 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>
```

6.5.2 Get Row Field Value

Description

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

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

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

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

The *fieldValue* is case-sensitive. An attempt to get the value Data from a current field value of Data is successful, but an attempt to get the value DATA fails

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

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- The *transparentDataType* must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier in the transparent data exists for the pool.
- The field name specified must be a valid field for the Entity as defined in the SEC.
- The field 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 to 22 digits in length

  Values: 1 to 999999999999999999999

- *transparentDataType*: A user defined type/name for the transparent data
  - Value is poolquota for the Poolquota transparent data
  - Value is pooldynamicquota for the *PoolDynamicquota* transparent data
- *rowIdValue*: The row name value that identifies the row in the transparent data blob
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance
  - Value is cid or Type for the *Poolquota* transparent data
  - Value is InstanceId or Type for the Pool*Dynamicquota* transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*
- *fieldName*: A user defined field in the transparent data row
- *fieldValue*: Corresponding field value assigned to *fieldName*

Request Content

None.

Response Content

A `<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 pool*dynamicquota* for the *PoolDynamicquota* transparent data

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

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

Table 68 Get Row Field Value Response Status/Error Codes

HTTP Status Code	Error Code	Description
200		Requested data row field/value exists for pool
404	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
204		No Content returned

Get Row Field Value 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=*error>
```

Request 3

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

Request URL

```
GET {BaseURI}/msr/pool/100000/data/poolquota/Weekday/inputVolume/3220
```

Request Content

None

Response 3

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

HTTP Status Code

200

Response Content

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

Request 4

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

Request URL

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

Request Content

None

Response 4

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

HTTP Status Code

200

Response Content

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

6.5.3 Update Row Field

Description

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

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

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

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

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

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

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- The *transparentDataType* must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier in the transparent data exists for the pool.
- The field name specified must be a valid field for the Entity as defined in the SEC. The field must be updatable.

Request URL

Without Instance Identifier

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

With Instance Identifier

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

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

  Values: 1 to 999999999999999999999

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

  Value is poolquota for the Poolquota transparent data

  Value is pool*dynamicquota* for the *PoolDynamicquota* transparent data

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

Request Content

None.

Response Content

None.

Table 69 Update Row Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
201		Requested transparent data row field was successfully created
400	MSR4051	Invalid value for a field
400	MSR4056	Field is not updatable
400	MSR4067	Multiple matching rows found
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

Update Row Field 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=*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 Pool*Dynamicquota* 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 Pool*Dynamicquota* transparent data was updated.

HTTP Status Code

201

Response Content

None

6.5.4 Delete Row Field

Description

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

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

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

If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and the row or field data is not deleted.

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

If the deleted field is mandatory, and has a default value, then the field is not removed, but has the default value assigned.

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

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

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- The *transparentDataType* must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier in the transparent data exists for the pool.
- The field name specified must be a valid field for the Entity as defined in the SEC. The field must be updatable.

Request URL

Without Instance 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 to 22 digits in length

  Values: 1 to 999999999999999999999

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

  Value is poolquota for the Poolquota transparent data

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

- *fieldName*: A user defined field in the transparent data row

Request Content

None.

Response Content

None.

Table 70 Delete Row Field Response Status/Error Codes

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

Delete Row Field 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=*error>
```

Request 3

A request is made to delete the InitialTotalVolume field in the PDQ1 data row of the pool*dynamicquota* 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 Pool*Dynamicquota* transparent data was deleted.

HTTP Status Code

204

Response Content

None

6.6 Pool Data Field Commands

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

```
<property>
  <name>X</name>
```

```
  <value>Y</value>
</property>
```

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

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

Table 71: Summary of Pool Data Field Commands

Command	Description	Keys	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 in transparent data of the specified type		DELETE {baseURI}/msr/pool/*poolId*/data/*transparentDataType*/*fieldName*

6.6.1 Set Data Field

Description

This operation creates or updates a field in a transparent data for the pool identified by the *poolId*.

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

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

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

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

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

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

Prerequisites

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

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

Request URL

Format 1

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

Format 2

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

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

  Values: 1 to 9999999999999999999

- *transparentDataType*: A user defined type/name for the transparent data
- Value is poolstate for the PoolState transparent data
- *fieldName*: A user defined field in the transparent data
  - For the PoolState entity this corresponds to a property in the entity
  - The *fieldName* is stored exactly as it is sent in the request. The case of *fieldName* changes if an update is done using a different case.
- *fieldValue*: Corresponding field value assigned to *fieldName*.

Request Content

None.

Response Content

None.

Table 72 Set Data Field Response Status/Error Codes

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

Set Data Field 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=*error>
```

6.6.2 Get Data Field

Description

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

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

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

The transparent data field is specified in *fieldName*.

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

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

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- The *transparentDataType* must reference a valid Pooled transparent Entity in the Interface Entity Map table in the SEC.
- A field in the transparent data exists for the pool.

Request URL

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

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

Values: 1 to 999999999999999999999

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

  Value is poolstate for the PoolState transparent data

- *fieldName*: A user defined field in the transparent data

  NOTE: For the PoolState entity this corresponds to a property in the entity

Request Content

None.

Response Content

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

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

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

  Value is poolstate for the PoolState transparent data

- *cdataFieldValue*: Contents of the XML data blob, with the field from the transparent data

Table 73 Get Data Field 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
204		No Content returned

Get Data Field 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=*error>
```

6.6.3 Delete Data Field

Description

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

The field identifier is specified in *fieldName*.

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

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

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

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

Prerequisites

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

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

Request URL

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

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

  Values: 1 to 999999999999999999999

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

  Value is poolstate for the PoolState transparent data

- *fieldName*: A user defined field in the transparent data

  NOTE: For the PoolState entity this corresponds to a property in the entity

Request Content

None.

Response Content

None.

Table 74 Delete Data Field Response Status/Error Codes

HTTP Status Code	Error Code	Description
204		Requested transparent data field was successfully deleted
400	MSR4067	Multiple matching fields found
400	MSR4064	Occurrence constraint violation
404	MSR4001	Pool is not found
404	MSR4049	Data type is not defined
404	MSR4058	Data type not found

Delete Data Field 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

400

Response Content

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

6.7 Additional Pool Commands

Table 75: Summary of Additional Pool Commands

Command	Description	Keys	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`
Get All Pool Members	Retrieve pool member subscribers from all local or local/remote systems by *PoolId*	*PoolId*	`GET {BaseURI}/msr/pool/poolId/allmembers/AddressList/addresslist/PSO/pso`

6.7.1 Add Member to Pool

Description

This operation adds a subscriber to a pool.

When the PSO flag is enabled, a pool member added to a pool on a Non Pool Host UDR is not treated as a pool member until the pool is created or pool profile updated on the Pool Host UDR when the connection between the two is active. Subscribers are only treated as a pool member for REST behavior, not for signaling.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

A subscriber with the key of the *keyName*/*keyValue* supplied must exist.

The subscriber must not be a member of a pool.

The pool must have less than the maximum number of member subscribers allowed.

Request URL

`POST {BaseURI}/msr/pool/poolId/member/subKeyName/subKeyValue`

- *poolId*: *PoolId* value of the pool. Numeric value, 1 to 22 digits in length

  Values: 1 to 999999999999999999999

- *subKeyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId
- *subKeyValue*: Corresponding key field value assigned to *keyName*

Request Content

None.

Response Content

None.

Table 76 Add Member to Pool Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	NA	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

Add Member to Pool Examples

Request 1

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is not a member of a pool.

Request URL

```
POST {BaseURI}/msr/pool/100000/member/MSISDN/380561234567
```

Request Content

None

Response 1

The request is successful, and the subscriber is added to the pool.

HTTP Status Code

204

Response Content

None

Request 2

A request is made to add a subscriber to a pool. The subscriber exists, but the pool does not.

Request URL

```
POST {BaseURI}/msr/pool/100009/member/IMSI/184569547984229
```

Request Content

None

Response 2

The request fails because the pool does not exist.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*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

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

Request 4

A request is made to add a subscriber to a pool. Both the pool and the subscriber exist. The subscriber is a member of a pool.

Request URL

```
POST {BaseURI}/msr/pool/100000/member/AccountId/10404723525
```

Request Content

None

Response 4

The request fails because the subscriber is a member of a pool.

HTTP Status Code

409

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

Request 5

A request is made to add a subscriber to a basic pool. Both the pool and the subscriber exist. The subscriber is not a member of a pool. The basic pool has the maximum number of members allowed.

Request URL

```
POST {BaseURI}/msr/pool/100000/member/MSISDN/15141234567
```

Request Content

None

Response 5

The request fails because the pool has the maximum number of members allowed.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

6.7.2 Remove Member from Pool

Description

This operation removes a subscriber from a pool.

Prerequisites

- A pool with the key of the *poolId* supplied must exist.
- A subscriber with the key of the *keyName*/*keyValue* supplied must exist.
- The subscriber must be a member of the specified pool.

Request URL

`DELETE {BaseURI}/msr/pool/`*`poolId`*`/member/`*`subKeyName`*`/`*`subKeyValue`*

- *poolId*: *PoolId* value of the pool. Numeric value, 1 to 22 digits in length

  Values: 1 to 999999999999999999999

- *subKeyName*: A key field in the subscriber profile

  Value is either IMSI, MSISDN, NAI, or AccountId

- *subKeyValue*: Corresponding key field value assigned to *keyName*

Request Content

None.

Response Content

None.

Table 77 Remove Member from Pool Response Status/Error Codes

HTTP Status Code	Error Code	Description
204	NA	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

Remove Member from 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

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

6.7.3 Get Pool Members

Description

This operation gets the list of subscriber members of a pool by *poolId*. This operation only gets the list of subscribers and addresses for a local pool on the UDR instance where the request was received, regardless if the pool is a PSO pool or not.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

Request URL

```
GET {BaseURI}/msr/pool/poolId/member
```

- *poolId*: *PoolId* value of the pool. Numeric value, 1 to 22 digits in length

  Values: 1 to 999999999999999999999

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*

Table 78 Get Pool Members Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	NA	Pool exists, and membership returned OK
404	MSR4061	Specified pool does not exist

Get Pool Members Examples

Request 1

A request is made to get the list of subscribers for a pool.

Request URL

```
GET {BaseURI}/msr/pool/100000/member
```

Request Content

None

Response 1

The request is successful, and the 3 member subscribers are returned.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  </member>
  <member>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
    <id><name>NAI</name><value>joe@wireless.com</value></id>
    <id><name>NAI</name><value>p12321@mynet.com</value></id>
  </member>
</members>
```

Request 2

A request is made to get the list of subscribers for a pool. The pool exists, but does not have member subscribers.

Request URL

```
GET {BaseURI}/msr/pool/200000/member
```

Request Content

None

Response 2

The request is successful, and member subscribers are not returned.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
</members>
```

Request 3

A request is made to get the list of subscribers for a pool. The pool does not exist.

Request URL

```
GET {BaseURI}/msr/pool/300000/member
```

Request Content

None

Response 3

The request fails, because the pool was not found.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

6.7.4 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 to 22 digits in length

  Values: 1 to 999999999999999999999

Table 79 Get *PoolId* Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	NA	Subscriber pool membership successfully returned
404	MSR4001	Subscriber is not found
404	MSR4062	Subscriber is not a member of a pool

Get PoolId *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

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

6.7.5 Get All Pool Members

Description

This operation has the option to get the list of subscriber members of a pool by *poolId*, from the local or from the local and all remote UDR systems. The Pool Spanning UDRs feature needs to be enabled in order to retrieve remote pool members. This command can be used on a single local Pool Host UDR as well, when the Pool Spanning UDRs feature is not enabled.

Prerequisites

A pool with the key of the *poolId* supplied must exist.

Request URL

```
GET {BaseURI}/msr/pool/poolId/allmembers/AddressList/addressList/PSO/pso
```

- *poolId*: *PoolId* value of the pool. Numeric value, 1 to 22 digits in length

  Values: 1 to 999999999999999999999

- *addressList:* The requested address type of the subscriber for a pool is returned in the response

  Values:

  - IMSI—only pool members with an IMSI address type are returned in the response
  - All—all pool members are returned in the response (default)

- *pso:* Indicates whether the response includes pool members across all UDR instances in the pool network or only the UDR instance where the request is received

  Values:

  - yes—response includes pool members across all UDR instances in the pool network
  - no—response only includes pool members on the local UDR instance (default)

NOTES:

- If a timeout occurs while waiting for a response from a remote UDR instance, then the UDR fails the request with a REQUEST_TIMEOUT error.
- If a request fails to be sent to a remote UDR instance, then the UDR fails the provisioning request with a REQUEST_TIMEOUT.
- This request can be used even when the Pool Spanning UDRs feature is not enabled.

Request Content

None.

Response Content

A <members> element that contains a <member> element for every subscriber that is a member of the pool. The <member> element is optional. There can be zero, one or many <member> elements. It is only present if the pool has member subscribers. One instance is present for every subscriber that is a member of the pool. A <member> element contains details about a single subscriber, containing requested user identities for that subscriber, one user identity per <id> element. There can be one or many <id> elements per <member> element.

The response includes pool members across all UDR instances in the pool network if the pso parameter was set to yes in the request and the PSO feature is enabled. Otherwise, only pool members on the local UDR instance are returned.

The response includes only the address types specified in the addresslist parameter in the request. (that is, those where *keyNameX* is in the list). Only the requested address types for a subscriber is returned in the response. If a subscriber is a pool member but does not have any of the required address types, then the pool member is not included in the response.

If the PSO feature is enabled, then each pool member returned also includes the *udrId* that corresponds to the UDR instance where they reside in the pool network.

```
<members>
[
  <member>
[   <udrId>udrId</udrId> ]
    <id><name>keyName1</name><value>keyValue1</value></id>
[
    <id><name>keyName2</name><value>keyValue2</value></id>
    :
    <id><name>keyNameN</name><value>keyValueN</value></id>
]
  </member>
]
[
  <member>
[   <udrId>udrId</udrId> ]
    <id><name>keyName1</name><value>keyValue1</value></id>
[
    <id><name>keyName2</name><value>keyValue2</value></id>
    :
    <id><name>keyNameN</name><value>keyValueN</value></id>
]
  </member>
  :
  <member>
[   <udrId>udrId</udrId> ]
    <id><name>keyName1</name><value>keyValue1</value></id>
[
    <id><name>keyName2</name><value>keyValue2</value></id>
          :
    <id><name>keyNameN</name><value>keyValueN</value></id>
]
  </member>
```

```
]
</members>
```

- *udrId*: (Optional) A subscriber irange that is hosted by each UDR in the remote pool network. The response only includes the *udrId* if the Pools Spanning UDRs feature is enabled.

  Value: 1 to 10 digits

- *keyNameX*: A key field for the member subscriber

  Value is either IMSI, MSISDN, NAI, or AccountId

- *keyValueX*: Corresponding key field value assigned to *keyNameX*

Table 80 Get All Pool Members Response Status/Error Codes

HTTP Status Code	Error Code	Description
200	NA	Pool exists, and membership returned OK
404	MSR4061	Specified pool does not exist
404	MSR4102	Provisioning Request Timeout, as no response was received from Remote UDR

Get All Pool Members Examples

Request 1

A request is made to get the list of subscriber members for a pool. The PSO feature is not enabled.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/no
```

Request Content

None

Response 1

The request is successful, and the 3 subscriber members are returned. Only pool members from the local UDR instance are returned.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  </member>
  <member>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
    <id><name>NAI</name><value>joe@wireless.com</value></id>
    <id><name>NAI</name><value>p12321@mynet.com</value></id>
```

```
  </member>
</members>
```

Request 2

A request is made to get the list of subscriber members for a pool. The pool exists, but does not have subscriber members. The PSO feature is enabled.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/no
```

Request Content

None

Response 2

The request is successful, and subscriber members are not returned.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
</members>
```

Request 3

A request is made to get the list of subscriber members for a pool. The pool does not exist (local or any remote UDR instances). The PSO feature is enabled and the pso parameter is set to yes.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None

Response 3

The request fails. The error value indicates that the pool does not exist on all queried UDR instances (local or remote).

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

Request 4

A request is made to get the list of subscriber members for a pool across all UDR instances in the pool network. The PSO feature is enabled and the pso parameter is set to yes.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None

Response 4

The request is successful, and the 3 member subscribers are returned. The response includes pool members across all UDR instances in the pool network.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <udrId>1</udrId>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  </member>
  <member>
     <udrId>1</udrId>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
    <udrId>2</udrId>
    <id><name>NAI</name><value>joe@wireless.com</value></id>
    <id><name>NAI</name><value>p12321@mynet.com</value></id>
  </member>
</members>
```

Request 5

A request is made to get the list of pool members across all UDR instances in the pool network. The pso parameter is set to yes. The PSO feature is enabled. There is a connection issue between the local and remote UDRs.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None

Response 5

The request fails. The *error* value indicates a provisioning request timeout, the request was not sent to the remote UDR due to the connection being down.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<error code=*error>
```

Request 6

A request is made to get the list of subscribers for a pool on all UDRs. The PSO feature is enabled. For this case, a response is not received from the remote UDR.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None

Response 6

The request fails. The *error* value indicates a provisioning request timeout; a response was not received from the remote UDR.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

Request 7

A request is made to get the list of subscriber members for a pool when the PSO feature is disabled and the pso parameter is set to yes.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None

Response 7

The request is successful, and the 3 member subscribers are returned. The response includes local pool members only.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>IMSI</name><value>311480100532432</value></id>
    <id><name>NAI</name><value>dad@operator.com</value></id>
  </member>
  <member>
    <id><name>MSISDN</name><value>380561234777</value></id>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
  <member>
```

```
    <id><name>NAI</name><value>joe@wireless.com</value></id>
    <id><name>NAI</name><value>p12321@mynet.com</value></id>
  </member>
</members>
```

Request 8

A request is made to get the list of subscriber members for a pool. The PSO feature is enabled. The pso parameter is set to yes and addressList is set to IMSI.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/IMSI/PSO/yes
```

Request Content

None

Response 8

The request is successful, and the 2 member subscribers are returned. The response includes only IMSI keys for all pool members.

HTTP Status Code

200

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
  <member>
    <udrId>1</udrId>
    <id><name>IMSI</name><value>311480100000001</value></id>
    <id><name>IMSI</name><value>311480100532432</value></id>
  </member>
  <member>
     <udrId>2</udrId>
    <id><name>IMSI</name><value>311480100000999</value></id>
  </member>
</members>
```

Request 9

A request is made to get the list of subscribers for a pool on all UDRs. The PSO feature is enabled and activated on 4 UDRs. For this case, response is not received from one remote UDR.

Request URL

```
GET {BaseURI}/msr/pool/100000/allmembers/AddressList/All/PSO/yes
```

Request Content

None

Response 9

The request fails. The error value indicates a provisioning request timeout; a response was not received from a remote UDR.

HTTP Status Code

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*error>
```

6.8 Pool Special Operation Commands

A transparent data entity may contain data that is organized in rows. An example of a row is a specific quota in the *Poolquota* entity.

The required row is identified in the request by the *rowIdValue* .

A specific instance of a quota (a specified row) in the *Poolquota* transparent data entity can have its fields reset to pre-defined values using a provisioning command.

Table 81: Summary of Pool Special Operation Commands

Command	Description	Keys	Command Syntax
Reset Pool Quota	Reset the fields in the specified pool quota	*PoolId* and Row Identifier	`POST {BaseURI}/msr/pool/keyName/data/transparentDataType/rowIdValue`
	Reset the fields in the specified pool quota and instance identifier	*PoolId*, Row Identifier, and Instance Identifier	`POST {BaseURI}/msr/pool/keyName/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue`

6.8.1 Reset Pool Quota

Description

This operation resets a particular quota row in the *Poolquota* transparent data associated with a pool.

If more than one row matches the requested *rowIdValue* , then the reset request fails.

If the pool has *Poolquota* transparent data, then the configured values in the specified quota row are reset to the configured reset values.

The *rowIdValue* is case-sensitive. If a row exists called DayPass, then an attempt to reset a quota row called DayPass is successful, but an attempt to reset a quota row called DAYPASS fails.

When a *Poolquota* instance is reset using the `Pool Reset Quota` command, each resettable field is set to its defined reset value. If the field does not exist, it is not created. But, if a resettable field does not exist, and the field has a default value, then the field is created with the default value.

If PSO is enabled, this operation is ignored on an NPHO and a success is returned. No updates are made to the database for these requests on NPHO.

Prerequisites

A pool with the key of the *keyName*supplied must exist.

The *Poolquota* transparent data must exist for the pool.

The specified quota row must exist in the *Poolquota* transparent data.

Request URL

Without Instance Identifier

```
POST {BaseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue
```

With Instance Identifier

```
POST
{BaseURI}/msr/pool/poolId/data/transparentDataType/rowIdValue/row/instanceFieldName/instanceFieldValue
```

- *poolId*: *PoolId* value of the pool. Numeric value, 1 to 22 digits in length

  Values: 1 to 999999999999999999999

- *transparentDataType*: A user defined type/name for the transparent data

  Value is poolquota for the Poolquota transparent data

- *rowIdValue*: The row name value that identifies the row in the transparent data blob
- *instanceFieldName*: A user defined field in the data row that is used to define a unique row instance

  Value is cid for the Poolquota transparent data

- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

Request Content

None.

Response Content

None.

Table 82 Reset Pool Quota Response Status/Error Codes

HTTP Status Code	Error Code	Description
204		Requested transparent data row was successfully reset
400	MSR4067	Multiple matching rows found
404	MSR4001	*PoolId* 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

Reset Pool Quota 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

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*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

400

Response Content

```
<?xml version="1.0" encoding="UTF-8"?>
<error code=*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=*error>
```

Request 5

A request is made to reset the PQ1 Poolquota row for a pool having cid of value 45678. The pool has *Poolquota* transparent data, and the Poolquota transparent data contains a Poolquota row called PQ1 having cid of value 45678.

Request URL

```
POST {baseURI}/msr/pool/10000/data/poolquota/PQ1/row/cid/45678
```

Request Content

None

Response 5

The request is successful, and the specified *Poolquota* row was reset.

HTTP Status Code

204

Response Content

None

Appendix A. REST Interface System Variables

The REST interface has a set of system variables that affect its operation as it runs. REST interface variables (Table 83) can be set through the UDR GUI and can be changed at runtime to effect dynamic server reconfiguration.

Table 83: REST Interface System variables

Parameter	Description
REST Interface Port	REST Interface TCP Listening Port. **NOTE:** Changes to the TCP listening port do not take effect until the udrprov process is restarted. Also, you must specify a different port than the SOAP interface. DEFAULT is 8787; Range is 0 to 65535
REST Interface Idle Timeout	The maximum time (in seconds) that an open REST connection remains active without a request being sent, before the connection is dropped. DEFAULT is 1200; Range is 1 to 86400
Maximum REST Connections	Maximum number of simultaneous REST Interface client connections. Default is 100; Range is 1 to 100
Allow REST Connections	Whether or not to allow incoming provisioning connections on the REST Interface. Default is UNCHECKED
REST Secure Mode	Whether the REST Interface operates in secure mode (using TLS), or unsecure mode (plain text). **NOTE:** Changes to the Secure Mode do not take effect until the udrprov process is restarted. Default is Unsecure
Transaction Durability Timeout*	The amount of time (in seconds) allowed between the committal of a transaction and it becoming durable. If Transaction Durability Timeout lapse, DURABILITY_TIMEOUT response is sent to the originating client. The associated request is resent to ensure that the request was committed. Default is 5; Range is 2 to 3600
Compatibility Mode*	Indicates whether backwards compatibility is enabled. **NOTE:** Change to Compatibility Mode may cause the existing provisioning connections to drop. Default is R10.0+

Appendix B. Legacy SPR Compatibility Mode

UDR can be configured to run in a compatibility mode with the legacy SPR.

When the Compatibility Mode system option (see Appendix A), which is configurable by the UDR GUI, is set to R10.0+ then UDR behaves as described in the main body of this document. When Compatibility Mode is set to R9.x, then the differences contained in this appendix apply.

Enabling this configuration option results in the format of some requests and responses being different from the default UDR behavior. This appendix lists the different request or responses that enabling the option applies to.

B.1 GET ROW RESPONSE FORMAT

UDR returns a data row in the format it is defined or 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, in the CDATA. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <field name="cid"/>
  <field name="time"/>
  <field name="totalVolume">0</field>
  <field name="inputVolume">0</field>
  <field name="outputVolume">0</field>
  <field name="serviceSpecific"/>
  <field name="nextResetTime"/>
  <field name="Type">quota</field>
  <field name="grantedTotalVolume">0</field>
  <field name="grantedInputVolume">0</field>
  <field name="grantedOutputVolume">0</field>
  <field name="grantedTime"/>
  <field name="grantedServiceSpecific"/>
  <field name="QuotaState">Valid/Inactive</field>
  <field name="RefInstanceId"/>
  <field name="name">test</field>
</usage>
]]>
  </data>
</subscriber>
```

If more than one matching row is found, then multiple <quota> rows are returned. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  :
</usage>
]]>
  </data>
</subscriber>

<subscriber>
  <data name="quota">
<![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  :
```

```
</usage>
]]>
  </data>
</subscriber>
```

Appendix C. My Oracle Support

My Oracle Support (*https://support.oracle.com*) is your initial point of contact for all product support and training needs. A representative at Customer Access Support (CAS) can assist you with My Oracle Support registration.

Call the CAS main number at 1-800-223-1711 (toll-free in the US), or call the Oracle Support hotline for your local country from the list at *http://www.oracle.com/us/support/contact/index.html*. When calling, make the selections in sequence on the Support telephone menu:

1. Select **2** for New Service Request
2. Select **3** for Hardware, Networking and Solaris Operating System Support
3. Select one of the options:
    - For Technical issues such as creating a Service Request (SR), Select **1**
    - For Non-technical issues such as registration or assistance with MOS, Select **2**

You are connected to a live agent who can assist you with My Oracle Support registration and opening a support ticket.

My Oracle Support is available 24 hours a day, 7 days a week, 365 days a year.

Appendix D. Locate Product Documentation on the Oracle Help Center Site

Oracle Communications customer documentation is available on the web at the Oracle Help Center site, *http://docs.oracle.com*. You do not have to register to access these documents. Viewing these files requires Adobe Acrobat Reader, which can be downloaded at *http://www.adobe.com*.

1. Access the Oracle Help Center site at *http://docs.oracle.com*
2. Click **Industries**.
3. Under the Oracle Communications subheading, click **Oracle Communications documentation**.

   The Communications Documentation page appears. Most products covered by these documentation sets are listed under the headings Network Session Delivery and Control Infrastructure or Platforms.

4. Click the product and then the release number.

   A list of the documentation set for the selected product and release appears.

5. To download a file, right-click the **PDF** link, select **Save target as** (or similar command based on your browser), and save to a local folder.