

# User Data Repository SOAP Provisioning Interface Specification for Release 12.11.0

F56662-01

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, see Appendix D.

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

F56662-01

Copyright ©2014, 2018, 2022 Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

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

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

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

## Table of Contents

|  |           |
|--|-----------|
| <b>CHAPTER 1. INTRODUCTION.....</b>                | <b>11</b> |
| 1.1 Purpose and Scope.....                         | 11        |
| 1.2 References.....                                | 11        |
| 1.3 Acronyms .....                                 | 11        |
| <b>CHAPTER 2. SYSTEM ARCHITECTURE .....</b>        | <b>13</b> |
| 2.1 Overview .....                                 | 13        |
| 2.2 Provisioning Interface.....                    | 14        |
| 2.2.1 Subscriber Commands .....                    | 14        |
| 2.2.2 Pool Commands .....                          | 15        |
| 2.3 XML SOAP Application Server (XSAS) .....       | 15        |
| 2.4 Provisioning Clients.....                      | 15        |
| 2.5 Security .....                                 | 16        |
| 2.5.1 Client Server IP Address White List .....    | 16        |
| 2.5.2 Secure Connection using TLS.....             | 16        |
| 2.6 Multiple Connections .....                     | 18        |
| 2.7 Request Queue Management .....                 | 19        |
| 2.8 Database Transactions .....                    | 19        |
| 2.8.1 Block Transaction Mode .....                 | 19        |
| 2.8.2 ACID-Compliance .....                        | 24        |
| 2.9 Connection Management .....                    | 25        |
| 2.9.1 Connections Allowed .....                    | 25        |
| 2.9.2 Disable Provisioning .....                   | 25        |
| 2.9.3 Idle Timeout .....                           | 26        |
| 2.9.4 Maximum Simultaneous Connections .....       | 26        |
| 2.9.5 TCP Port Number .....                        | 26        |
| 2.10 Behavior During Low Free System Memory .....  | 26        |
| 2.11 Multiple Subscriber Key Processing.....       | 26        |
| 2.12 Congestion Control .....                      | 27        |
| 2.13 Pools Spanning UDRs .....                     | 27        |
| 2.14 Enterprise Pools .....                        | 27        |
| <b>CHAPTER 3. SOAP INTERFACE DESCRIPTION .....</b> | <b>29</b> |
| 3.1.1 SOAP Header Format .....                     | 29        |
| 3.1.2 Status Codes and Error Messages .....        | 31        |

|  |           |
|--|-----------|
| <b>CHAPTER 4. SOAP INTERFACE MESSAGE DEFINITIONS .....</b> | <b>33</b> |
| 4.1 Message Conventions.....                               | 33        |
| 4.2 Basic XML Message Format .....                         | 33        |
| 4.3 Encoding of Multiple Embedded CDATA Sections .....     | 37        |
| 4.4 Case Sensitivity.....                                  | 38        |
| 4.5 List of Messages .....                                 | 40        |
| <b>CHAPTER 5. UDR DATA MODEL .....</b>                     | <b>43</b> |
| 5.1 Subscriber Data .....                                  | 45        |
| 5.1.1 Subscriber Profile .....                             | 45        |
| 5.1.2 Quota .....  | 47        |
| 5.1.3 State .....  | 49        |
| 5.1.4 Dynamic Quota .....                                  | 49        |
| 5.2 Pool Data .....  | 50        |
| 5.2.1 Pool Profile .....                                   | 50        |
| 5.2.2 Pool Quota.....                                      | 52        |
| 5.2.3 Pool State .....                                     | 52        |
| 5.2.4 Pool Dynamic Quota.....                              | 52        |
| 5.3 Date/Timestamp Format.....                             | 52        |
| <b>CHAPTER 6. SUBSCRIBER PROVISIONING .....</b>            | <b>54</b> |
| 6.1 Subscriber Profile Commands.....                       | 54        |
| 6.1.1 Create Profile.....                                  | 54        |
| 6.1.2 Get Profile.....                                     | 60        |
| 6.1.3 Delete Profile .....                                 | 64        |
| 6.2 Subscriber Profile Field Commands .....                | 68        |
| 6.2.1 Add Field Value .....                                | 68        |
| 6.2.2 Get Field .....                                      | 73        |
| 6.2.3 Update Field .....                                   | 80        |
| 6.2.4 Delete Field.....                                    | 86        |
| 6.2.5 Delete Field Value .....                             | 90        |
| 6.3 Subscriber Opaque Data Commands.....                   | 93        |
| 6.3.1 Create Opaque Data .....                             | 94        |
| 6.3.2 Get Opaque Data .....                                | 98        |
| 6.3.3 Update Opaque Data.....                              | 102       |
| 6.3.4 Delete Opaque Data.....                              | 105       |
| 6.4 Subscriber Data Row Commands .....                     | 107       |
| 6.4.1 Create Row.....                                      | 107       |

|                   |   |            |
|-------------------|---|------------|
| 6.4.2             | Get Row .....                               | 114        |
| 6.4.3             | Delete Row .....                            | 121        |
| 6.5               | Subscriber Data Row Field Commands .....    | 125        |
| 6.5.1             | Get Row Field .....                         | 126        |
| 6.5.2             | Update Row Field .....                      | 132        |
| 6.5.3             | Delete Row Field .....                      | 137        |
| 6.6               | Subscriber Data Field Commands .....        | 141        |
| 6.6.1             | Create Data Field .....                     | 142        |
| 6.6.2             | Get Data Field .....                        | 146        |
| 6.6.3             | Update Data Field .....                     | 149        |
| 6.6.4             | Delete Data Field .....                     | 152        |
| 6.7               | Subscriber Special Operation Commands ..... | 155        |
| 6.7.1             | Reset Quota .....                           | 155        |
| 6.7.2             | Get Subscription Status .....               | 160        |
| <b>CHAPTER 7.</b> | <b>POOL PROVISIONING .....</b>              | <b>164</b> |
| 7.1               | Pool Profile Commands .....                 | 164        |
| 7.1.1             | Create Pool .....                           | 164        |
| 7.1.2             | Get Pool .....                              | 168        |
| 7.1.3             | Delete Pool .....                           | 170        |
| 7.2               | Pool Field Commands .....                   | 172        |
| 7.2.1             | Add Field Value .....                       | 172        |
| 7.2.2             | Get Field .....                             | 175        |
| 7.2.3             | Update Field .....                          | 179        |
| 7.2.4             | Delete Field .....                          | 183        |
| 7.2.5             | Delete Field Value .....                    | 185        |
| 7.3               | Pool Opaque Data Commands .....             | 188        |
| 7.3.1             | Create Opaque Data .....                    | 188        |
| 7.3.2             | Get Opaque Data .....                       | 193        |
| 7.3.3             | Update Opaque Data .....                    | 196        |
| 7.3.4             | Delete Opaque Data .....                    | 198        |
| 7.4               | Pool Data Row Commands .....                | 200        |
| 7.4.1             | Create Row .....                            | 201        |
| 7.4.2             | Get Row .....                               | 205        |
| 7.4.3             | Delete Row .....                            | 212        |
| 7.5               | Pool Data Row Field Commands .....          | 216        |
| 7.5.1             | Get Row Field .....                         | 216        |
| 7.5.2             | Update Row Field .....                      | 222        |

|  |            |
|--|------------|
| 7.5.3 Delete Row Field .....   | 226        |
| 7.6 Pool Data Field Commands.....  | 230        |
| 7.6.1 Create Data Field .....  | 231        |
| 7.6.2 Get Data Field.....  | 235        |
| 7.6.3 Update Data Field.....   | 238        |
| 7.6.4 Delete Data Field.....   | 241        |
| 7.7 Additional Pool Commands .....   | 243        |
| 7.7.1 Add Member to Pool.....  | 244        |
| 7.7.2 Remove Member from Pool.....   | 248        |
| 7.7.3 Get Pool Members.....  | 250        |
| 7.7.4 Get PoolID .....   | 253        |
| 7.7.5 Get All Pool Members.....  | 256        |
| 7.8 Pool Special Operation Commands.....   | 262        |
| 7.8.1 Reset PoolQuota .....  | 263        |
| <b>APPENDIX A. SOAP INTERFACE ERROR CODES .....</b>                                    | <b>267</b> |
| <b>APPENDIX B. SOAP INTERFACE SYSTEM VARIABLES.....</b>                                | <b>270</b> |
| <b>APPENDIX C. LEGACY SPR COMPATIBILITY MODE.....</b>                                  | <b>271</b> |
| C.1 Legacy SPR SOAP Request Format .....   | 272        |
| C.2 Enable AE Key Already Exists Error.....  | 275        |
| <b>APPENDIX D. MY ORACLE SUPPORT.....</b>  | <b>277</b> |
| <b>APPENDIX E. LOCATE PRODUCT DOCUMENTATION ON THE ORACLE HELP CENTER<br/>SITE 278</b> |            |

## List of Figures

|  |     |
|--|-----|
| Figure 1: User Data Repository High Level Architecture ..... | 14  |
| Figure 2: SOAP Header Format.....                            | 29  |
| Figure 3: SOAP Request Format .....                          | 30  |
| Figure 4: SOAP Response Format .....                         | 31  |
| Figure 5: Data Model .....                                   | 45  |
| Figure 6: Legacy SPR SOAP Request Format.....                | 272 |
| Figure 7: Legacy SPR SOAP Response Format .....              | 273 |

## List of Tables

|   |     |
|---|-----|
| Table 1: Acronyms .....   | 11  |
| Table 2: TLS X.509 Certificate and Key PEM-encoded Files.....   | 17  |
| Table 3: TLS Supported Cipher Suites.....                       | 18  |
| Table 4: Message Conventions.....                               | 33  |
| Table 5 keyValue Validation for keyType .....                   | 35  |
| Table 6: Summary of Supported Subscriber Commands.....          | 40  |
| Table 7: Summary of Supported Pool Commands .....               | 41  |
| Table 8: Subscriber Profile Entity Definition .....             | 46  |
| Table 9: Quota Entity Definition.....                           | 48  |
| Table 10: State Entity Definition.....                          | 49  |
| Table 11: Dynamic Quota Entity Definition .....                 | 49  |
| Table 12: Pool Profile Entity Definition .....                  | 51  |
| Table 13: Summary of Subscriber Profile Commands .....          | 54  |
| Table 14: Summary of Subscriber Profile Field Commands .....    | 68  |
| Table 15: Summary of Subscriber Opaque Data Commands .....      | 93  |
| Table 16: Summary of Subscriber Data Row Commands .....         | 107 |
| Table 17: Summary of Subscriber Data Row Field Commands .....   | 125 |
| Table 18: Summary of Subscriber Data Commands .....             | 142 |
| Table 19: Summary of Subscriber Special Operation Commands..... | 155 |
| Table 20: Summary of Pool Profile Commands.....                 | 164 |
| Table 21: Summary of Pool Field Commands.....                   | 172 |
| Table 22: Summary of Pool Opaque Data Commands .....            | 188 |
| Table 23: Summary of Pool Data Row Commands.....                | 200 |
| Table 24: Summary of Pool Data Row Field Commands .....         | 216 |
| Table 25: Summary of Pool Data Commands.....                    | 230 |
| Table 26: Summary of Additional Pool Commands.....              | 243 |
| Table 27: Summary of Pool Special Operation Commands.....       | 263 |
| Table 28: SOAP Interface System variables.....                  | 270 |



## Error Codes

|   |     |
|---|-----|
| Error Codes 1: Create Profile .....           | 56  |
| Error Codes 2: Get Profile .....              | 62  |
| Error Codes 3: Delete Profile .....           | 65  |
| Error Codes 4: Add Field Value .....          | 70  |
| Error Codes 5: Get Field.....                 | 75  |
| Error Codes 6: Update Field .....             | 82  |
| Error Codes 7: Delete Field.....              | 87  |
| Error Codes 8: Delete Field Value.....        | 91  |
| Error Codes 9: Create Opaque Data .....       | 95  |
| Error Codes 10: Get Opaque Data .....         | 99  |
| Error Codes 11: Update Opaque Data .....      | 103 |
| Error Codes 12: Delete Opaque Data .....      | 106 |
| Error Codes 13 Create Row .....               | 110 |
| Error Codes 14: Get Row .....                 | 116 |
| Error Codes 15: Delete Row .....              | 123 |
| Error Codes 16: Get Row Field.....            | 128 |
| Error Codes 17: Update Row Field .....        | 134 |
| Error Codes 18: Delete Row Field.....         | 139 |
| Error Codes 19: Create Data Field .....       | 144 |
| Error Codes 20: Get Data Field .....          | 148 |
| Error Codes 21: Update Data Field .....       | 151 |
| Error Codes 22: Delete Data Field .....       | 154 |
| Error Codes 23: Reset Quota .....             | 157 |
| Error Codes 24: Get Subscription Status ..... | 157 |
| Error Codes 25: Create Pool .....             | 161 |
| Error Codes 26: Get Pool .....                | 169 |
| Error Codes 27: Delete Pool .....             | 171 |
| Error Codes 28: Add Field Value.....          | 174 |
| Error Codes 29: Get Field.....                | 177 |
| Error Codes 30: Update Field .....            | 181 |
| Error Codes 31: Delete Field.....             | 184 |
| Error Codes 32: Delete Field Value.....       | 187 |
| Error Codes 33: Create Opaque Data .....      | 189 |
| Error Codes 34: Get Opaque Data .....         | 195 |
| Error Codes 35: Update Opaque Data .....      | 197 |
| Error Codes 36: Delete Opaque Data .....      | 199 |
| Error Codes 37: Create Row .....              | 203 |
| Error Codes 38: Get Row .....                 | 207 |
| Error Codes 39: Delete Row .....              | 214 |
| Error Codes 40: Get Row Field.....            | 219 |
| Error Codes 41: Update Row Field .....        | 223 |
| Error Codes 42: Delete Row Field.....         | 227 |
| Error Codes 43: Create Data Field .....       | 232 |
| Error Codes 44: Get Data Field .....          | 236 |

|   |     |
|---|-----|
| Error Codes 45: Update Data Field .....       | 240 |
| Error Codes 46: Delete Data Field .....       | 242 |
| Error Codes 47: Add Member to Pool .....      | 245 |
| Error Codes 48: Remove Member from Pool ..... | 249 |
| Error Codes 49: Get Pool Members.....         | 252 |
| Error Codes 50: Get PoolID.....               | 254 |
| Error Codes 51: Get All Pool Members.....     | 258 |
| Error Codes 52: Reset PoolQuota.....          | 264 |
| Error Codes 53: SOAP Interface.....           | 267 |

## Chapter 1. Introduction

### 1.1 Purpose and Scope

This document presents the SOAP Provisioning interface to be used by provisioning client applications to administer the Provisioning Database of the Oracle Communications User Data Repository (UDR) system. Through SOAP 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 have a need to use the SOAP interface.

### 1.2 References

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

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

### 1.3 Acronyms

This section lists terms and acronyms specific to this document.

**Table 1: Acronyms**

| Acronym/Term | Definition   |
|--------------|--|
| ACID         | Atomic, Consistent, Isolatable, Durable  |
| BLOB         | Binary Large Object  |
| CFG          | Configuration Data—data for components and system identification and configuration |
| CPS          | Customer Provisioning System   |
| DP           | Database Processor   |
| FRS          | Feature Requirements Specification   |
| FTP          | File Transfer Protocol   |
| GUI          | Graphical User Interface   |
| IMEI         | International Mobile Equipment Identity  |
| IMSI         | International Mobile Subscriber Identity, or IMSI [im-zee]                         |
| IP           | Internet Protocol  |

|        |   |
|--------|---|
| KPI    | Key Performance Indicator                                 |
| MEAL   | Measurements, Events, Alarms, and Logs                    |
| MP     | Message Processor   |
| MSISDN | Mobile Subscriber ISDN Number                             |
| NA     | Not Applicable  |
| NE     | Network Element   |
| NPA    | Numbering Plan Area (Area Code)                           |
| 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                                       |
| 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 an SPR, which is a database system that acts as a single logical repository that stores subscriber data. The subscriber data that traditionally has been stored into the HSS, HLR, AuC, or Application Servers is stored in UDR as specified in 3GPP UDC information model [3]. UDR facilitates the share and the provisioning of user related data throughout services of 3GPP system. Several Applications Front Ends, such as one or more PCRF or HSS or HLR or AuCFEs can be served by UDR.

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

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

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

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

Oracle Communications User Data Repository consists of several functional blocks. The Message Processors (MP) provide support for a variety of protocols that entail the front-end signaling to peer network nodes. The back-end UDR database resides on the N-OAMP servers.

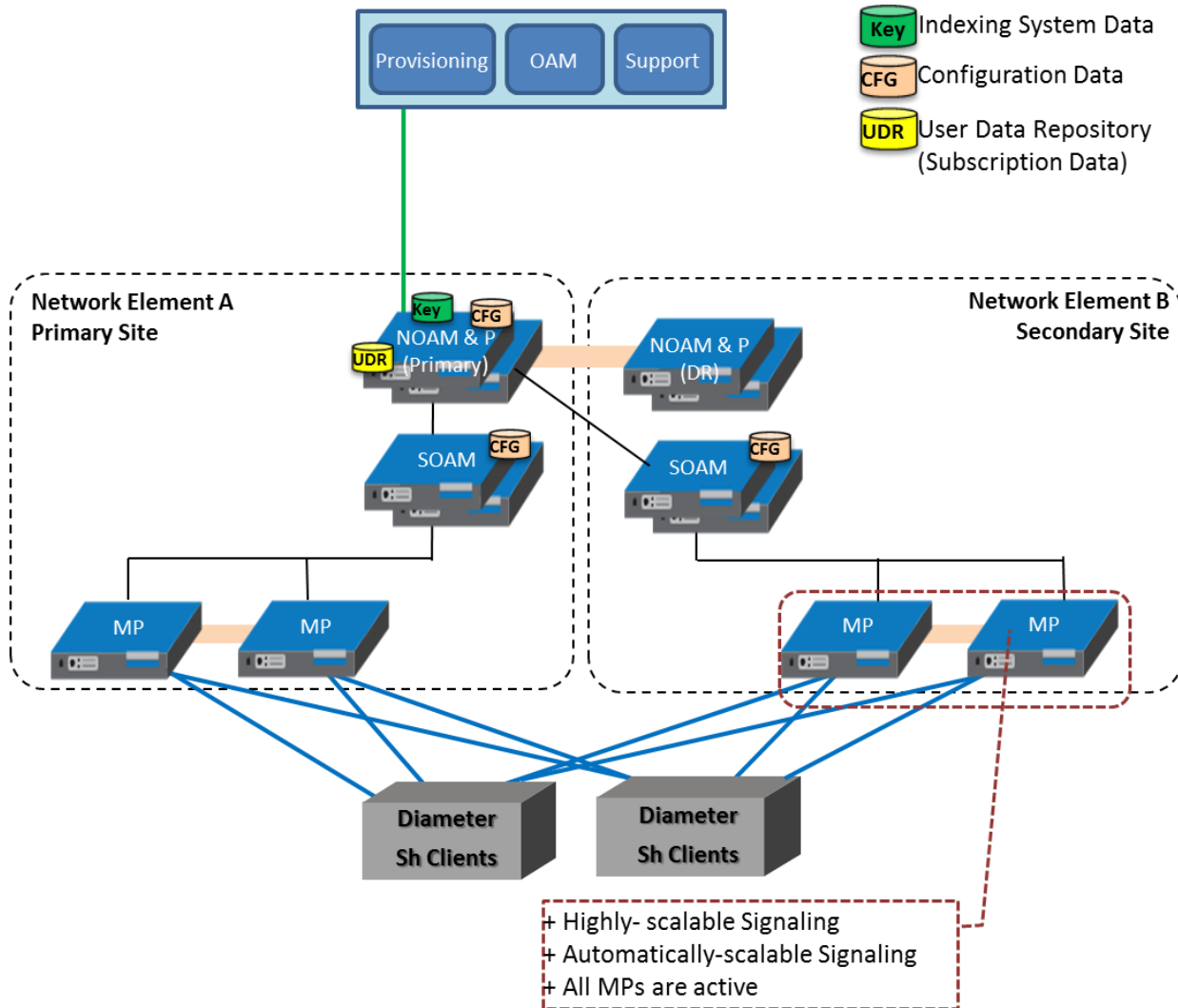
As the product evolves forward, the subscriber profiles in UDR can be expanded to support data associated with additional applications. Along with that, the MPs can be expanded to support additional Diameter interfaces associated with these applications. The IPFE can be integrated with the product to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

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

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

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

**Figure 1: User Data Repository High Level Architecture**



## 2.2 Provisioning Interface

The SOAP provisioning interface provides data manipulation commands for subscriber and Pool.

### 2.2.1 Subscriber Commands

- Subscriber profile create, retrieve, and delete
- Subscriber profile field create, retrieve, modify, and delete
- Subscriber opaque data create, retrieve, modify, and delete  
Quota, State, and Dynamic Quota
- Subscriber data row and field create, modify, and delete  
Quota, State and Dynamic Quota
- Subscriber transparent data create, retrieve, modify, delete

### Quota, State and Dynamic Quota

- Reset of Subscriber Quota transparent data row

## 2.2.2 Pool Commands

- Pool profile create, retrieve, and delete
- Pool profile field create, 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

- Pool data row and field create, modify, and delete

### Pool Quota, Pool State and Pool Dynamic Quota

- Pool subscriber membership operations

- o Add and remove from pool
- o Get pool subscriber membership
- o Get pool for subscriber

- Reset of Pool Quota transparent data row

## 2.3 XML SOAP Application Server (XSAS)

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

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

## 2.4 Provisioning Clients

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

CPSs use SOAP to send requests to manipulate and query data in the Provisioning Database. Provisioning Clients establish TCP/IP connections to the XSAS running on the active NOAMP using the VIP for the primary NOAMP.

Provisioning clients need to re-establish connections with the XSAS using the VIP for the primary UDR after switchover from the active server for the primary to the standby UDR server. Provisioning clients also need to 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, the client drops the connection and re-establishes it before trying again.

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

## 2.5 Security

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

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

### 2.5.1 Client Server IP Address White List

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

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

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

TLS is an industry standard protocol for clients needing to establish secure (TCP-based) TLS-enabled network connections. TLS provides data confidentiality, data integrity, and server and client authentication based on digital certificates that comply with X.509v3 standard and public 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 following capabilities of TLS address several fundamental concerns about communication over TCP/IP networks:

- TLS server authentication  
Allows a client application to confirm the identity of the server application. The client application through TLS uses standard public-key cryptography to verify that the certificate and public key for the server are valid and has been signed by a trusted certificate authority (CA) that is known to the client application.
- TLS client authentication  
Allows a server application to confirm the identity of the client application. The server application through TLS uses standard public-key cryptography to verify that the certificate and public key for the client are valid and has been signed by a trusted certificate authority (CA) that is known to the server application.
- An encrypted TLS connection  
Requires all information being sent between the client and server application to be encrypted. The sending application is responsible for encrypting the data and the receiving application is responsible for decrypting the data. In addition to encrypting the data, TLS provides message integrity. Message integrity provides a means to determine if the data has been tampered with since it was sent by the partner application.



Depending on the mode the XSAS is configured to operate in (secure or unsecure), provisioning clients can connect using unsecure or secure connections to the well-known TCP/TLS listening port for the XSAS (configured using the SOAP Secure Mode configuration variable via UDR GUI).

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

### ***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 SOAP server and provisioning clients needing to establish a TLS-enabled connection with the SOAP server. These files can be found on the UDR server in the `/usr/TKLC/udr/ss1` directory. These files are copied to the server running the provisioning client.

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

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

### ***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. The following table lists the set of TLS cipher suites from the relevant specification and their OpenSSL equivalents that are supported by the XSAS 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 XSAS supports multiple connections and each connection is considered persistent unless declared otherwise. The HTTP persistent connections do not use separate keep-alive messages, they just allow multiple requests to use a same TCP/IP connection. However, connections are closed after being idle for a time limit configured in idle timeout (See section 2.9.3).

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

**NOTE:** In order to achieve the maximum provisioning TPS rate that the UDR SOAP 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 SOAP Connections (see 7.8.1Appendix B) are 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.
- When calculating the provisioning TPS rate, if any <tx> transactions are sent (see section 2.8.1), then the TPS rate is calculated using the number of requests contained in the <tx>. A <tx> request does *not* count as 1 TPS.

## 2.7 Request Queue Management

If multiple clients simultaneously issues requests, each request is queued and processed in the order in which it was received on a per connection basis. The client is not required to 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 on multiple connections from a single client are handled on a first-in, first-out basis. Generally, requests are answered in the order in which they are received, but this is not always guaranteed. A client sends a number of valid update requests, which are performed, and run in the order they are received. If the client were to then send an invalid request (such as if the XML could not be parsed) on a different connection, this is responded to immediately, potentially before the any, some, or all of the previous requests have been responded to.

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

## 2.8 Database Transactions

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

### 2.8.1 Block Transaction Mode

The block database transaction mode requires explicit `<tx>` XML tags around all of the requests in a transaction.

The block transaction is sent as one XML request, and all requests contained in the block are run in the sequence supplied in a database transaction. If any request fails the entire transaction is automatically rolled back. If all requests are successful then the transaction is automatically committed.

If a block transaction fails, the request in the block that encountered an error has the error code set, all requests except the one that failed has error code=1 (NOT\_PROCESSED) which indicates that the request was not performed or has been rolled back.

All block transactions must also satisfy limits indicated by the `Maximum Requests in SOAP <tx>` XML and `Transaction Durability Timeout` system variables, which are defined in 7.8.1Appendix B. If any of those limits are exceeded, the transaction is aborted and automatically rolled back.

If a block transaction is sent which contains more than `Maximum Requests in SOAP <tx>` XML requests, then the request fails with a SOAP error `<message error="20">` (see section 3.1.2).

**NOTE:** The relevant transaction related measurements are incremented once per `<tx>` request (by +1). The relevant request based measurements are incremented once per request contained in the `<tx>`. All requests share the same outcome. For example, if a `<tx>` request contained 5 requests, and the transaction was successful, then measurement `RxXsasProvMsgsSuccessful` are incremented by 5. If the first 3 requests in the transaction were successful, and the 4th request failed, then the transaction fails, get rolled back, and `RxXsasProvMsgsFailed` are incremented by 5.

### Request Format

```
<tx [resonly="resonly"]>
  <req ... >
    request1
  </req>
  <expr><attr name="keyName1"/><value val="keyValue1"/></expr>
[
```

```

<req ... >
  request2
</req>
:
<req ... >
  requestN
</req>
]
</tx>

```

- *resonly*: (Optional) Indicates whether the all request responses in the transaction consist of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

**NOTE:** Any *resonly* value supplied in the <tx> takes precedence on any *resonly* value supplied in a contained request in <req>. If a *resonly* value is not supplied in the <tx>, then the value supplied in a contained request in <req> takes precedence. The default value for *resonly* when not supplied is *n*.

- *requestX*: SOAP XML request contained in the transaction

**NOTE:** The maximum number of requests that can be included in a <tx> transaction is defined in the `Maximum Requests` in SOAP <tx> XML system variable, which is defined in Appendix B.

### Response Format

```

<tx nbreq="nbreq" [resonly="resonly"]>
  <req ... >
  [
    originalXMLRequest1 ]
    <res error="error" affected="affected"/>
  </req>
  [
    <req ... >
  [
    originalXMLRequest2 ]
    <res error="error" affected="affected"/>
  </req>
  :
    <req ... >
  [
    originalXMLRequest3 ]
    <res error="error" affected="affected"/>
  </req>
  ]
</tx>

```

- *nbreq*: The number of requests contained in the original XML <tx> request
- *resonly*: (Optional) The *resonly* value from the original XML <tx> request, if supplied
- *originalXMLRequestN*: (Optional) The text of the original XML request that was contained in the <tx> request.

Values: A string with 1 to 4096 characters

- *error*: Error code indicating outcome of request. 0 means success. A value of 1 (NOT\_PROCESSED) indicates that the request was not performed or had been rolled back. Other values are dependent on the request being performed, and are listed in the description for that specific request
- *affected*: The number of subscribers affected by the request. A value of 1 is expected for success

**NOTE:** This value may be non-zero for requests that were valid in a transaction, but where a subsequent request failed, and the transaction was rolled back. The *affected* value is given to indicate that the request is successful if committed

**NOTES**

- For a transaction to be considered successful, all error values in all request responses must be 0.
- Results for a select request may be returned a response even if the transaction failed. Based on the error values for all request responses, it is up to the provisioning client sending the request to use the returned information for the select if the transaction itself was not successful.

**Examples****Request:**

This request creates 2 subscribers, and gets a 3rd subscriber.

```
<tx resonly="y">
  <req name="insert">
    <ent name="Subscriber"/>
    <set>
      <expr>
        <attr name="MSISDN"/>
        <value val="19195551234"/>
      </expr>
      <expr>
        <attr name="BillingDay"/>
        <value val="1"/>
      </expr>
      <expr>
        <attr name="Entitlement"/>
        <value val="DayPass,DayPassPlus"/>
      </expr>
    </set>
  </req>
  <req name="insert">
    <ent name="Subscriber"/>
    <set>
      <expr>
        <attr name="MSISDN"/>
        <value val="15141234567"/>
      </expr>
      <expr>
        <attr name="BillingDay"/>
        <value val="1"/>
      </expr>
      <expr>
        <attr name="Entitlement"/>
        <value val="DayPass,DayPassPlus"/>
      </expr>
    </set>
  </req>
  <req name="select">
    <ent name="Subscriber"/>
    <select>
      <expr>
        <attr name="IMSI"/>
      </expr>
      <expr>
        <attr name="MSISDN"/>
      </expr>
      <expr>
        <attr name="NAI"/>
      </expr>
    </select>
    <where>
      <expr>
        <attr name="IMSI"/>
        <op value="="/>
        <value val="302370123456789"/>
      </expr>
    </where>
  </req>
</tx>
```

**Response 1**

In this example, all requests were successful, and the transaction was committed.

```
<tx nbreq="3" resonly="y">
  <req name="insert">
    <res error="0" affected="1"/>
  </req>
  <req name="insert">
    <res error="0" affected="1"/>
  </req>
  <req name="select">
    <res error="0" affected="1"/>
    <rset>
      <row>
        <rv>302370123456789</rv>
        <rv>15145551234</rv>
        <rv>person@operator.com</rv>
      </row>
    </rset>
  </req>
</tx>
```

**Response 2**

In this example, the first request was successful, but the second request failed. The transaction was rolled back.

The second request failed due to error FIELD\_NOT\_FOUND. The third command was not attempted.

```
<tx nbreq="3" resonly="y">
  <req name="insert">
    <res error="1" affected="1"/>
  </req>
  <req name="insert">
    <res error="70012" affected="0"/>
  </req>
  <req name="select">
    <res error="1" affected="0"/>
  </req>
</tx>
```

**Response 3**

In this example, the second request is invalid due to an unknown entity. The transaction was not attempted.

```
<tx nbreq="3" resonly="y">
  <req name="insert" resonly="y">
    <res error="1" affected="0"/>
  </req>
  <req name="insert" resonly="y">
    <res error="70000" affected="0"/>
  </req>
  <req name="select" resonly="y">
    <res error="1" affected="0"/>
  </req>
</tx>
```

**Response 4**

In this example, all requests are valid, but the commit of the transaction failed. The transaction was rolled back.

```
<tx nbreq="3" resonly="y">
  <req name="insert">
    <res error="70038" affected="1"/>
  </req>
  <req name="insert">
    <res error="70038" affected="1"/>
  </req>
  <req name="select">
    <res error="70038" affected="1"/>
    <rset>
```

```

    <row>
      <rv>302370123456789</rv>
      <rv>15145551234</rv>
      <rv>person@operator.com</rv>
    </row>
  </rset>
</req>
</tx>

```

### Request 5

This request creates 2 subscribers, and gets a 3rd subscriber.

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

```

<tx resonly="y">
  <req name="insert">
    <ent name="Subscriber"/>
    <set>
      <expr>
        <attr name="IMEI"/>
        <value val="98765439876546"/>
      </expr>
      <expr>
        <attr name="IMSI"/>
        <value val="987654398765460"/>
      </expr>
    </set>
  </req>
  <req name="insert">
    <ent name="Subscriber"/>
    <set>
      <expr>
        <attr name="IMEI"/>
        <value val="98765439876547"/>
      </expr>
      <expr>
        <attr name="IMSI"/>
        <value val="987654398765470"/>
      </expr>
    </set>
  </req>
  <req name="select">
    <ent name="Subscriber"/>
    <select>
      <expr>
        <attr name="IMEI"/>
      </expr>
      <expr>
        <attr name="IMSI"/>
      </expr>
    </select>
    <where>
      <expr>
        <attr name="IMEI"/>
        <op value="="/>
        <value val=" 98765439876546"/>
      </expr>
    </where>
  </req>
</tx>

```

### Response 5

In this example, all requests were successful, and the transaction was committed.

```

<tx nbreq="3" resonly="y">
  <req name="insert">
    <res error="0" affected="1"/>
  </req>
  <req name="insert">
    <res error="0" affected="1"/>

```

```

</req>
<req name="select">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>98765439876546</rv>
      <rv>987654398765460</rv>
    </row>
  </rset>
</req>
</tx>

```

**Request 6**

This request creates 1 subscriber within the specified IMEI range and gets a third subscriber which exists.

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

```

<tx resonly="y">
  <req name="insert" resonly="y">
    <ent name="Subscriber"/>
    <set>
      <expr>
        <attr name="IMEI"/>
        <value val="97765439876546"/>
      </expr>
      <expr>
        <attr name="IMEI"/>
        <value val="98765439876546"/>
      </expr>
    </set>
  </req>
  <req name="select">
    <ent name="Subscriber"/>
    <select>
      <expr>
        <attr name="IMEI"/>
      </expr>
    </select>
    <where>
      <expr>
        <attr name="IMEI"/>
        <op value="="/>
        <value val="12345671234567"/>
      </expr>
    </where>
  </req>
</tx>

```

**Response 6**

In this example, request was successful, and the transaction was committed.

```

<tx nbreq="3" resonly="y">
  <req name="insert">
    <res error="0" affected="1"/>
  </req>
</tx>

```

**2.8.2 ACID-Compliance**

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

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

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

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

## **2.9 Connection Management**

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

### **2.9.1 Connections Allowed**

The configuration variable `ALLOW_SOAP_CONNECTIONS` (see 7.8.1Appendix B) controls whether SOAP 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_SOAP_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 6.1.1.

### 2.9.3 Idle Timeout

HTTP connection between Provisioning client and XSAS is handled persistent fashion. The configuration variable SOAP Interface Idle Timeout (see 7.8.1Appendix B) indicates the time to wait before closing the connection due to inactivity (requests are not received).

### 2.9.4 Maximum Simultaneous Connections

The configuration variable Maximum SOAP Connections (see 7.8.1Appendix B) defines the maximum number of simultaneous SOAP interface client connections. If an attempt is made to connect more than the number of SOAP connections allowed, the connection is rejected by the SOAP server.

### 2.9.5 TCP Port Number

The configuration variable SOAP Interface Port (see 7.8.1Appendix B) defines the SOAP 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 MEMORY\_FULL. Before this happens, memory threshold alarms are raised indicating the impending behavior if the critical level is reached.

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

```
<res error="70042" affected="0"/>
```

## 2.11 Multiple Subscriber Key Processing

UDR allows multiple key values for a subscriber to be supplied in some requests in the <where> element.

When multiple keys are supplied in a request (such as an IMSI and an MSISDN), UDR looks up all supplied keys, and only consider the subscriber record found if all supplied keys correspond to the same subscriber.

If any key value does not exist, then KEY\_NOT\_FOUND returns. If multiple keys are provided, and all keys exist, but do not correspond to the same subscriber, then the error MULTIPLE\_KEYS\_NOT\_MATCH returns.

Example request:

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><value val="23"/></expr>
    <expr><attr name="Tier"/><value val="Gold"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/><value val="302370123456789"/></expr>
    <expr><attr name="MSISDN"/><op value="="/><value val="15145551234"/></expr>
  </where>
</req>
```

Multiple values for the same key type are also allowed, such as if a subscriber has two provisioned IMSIs, the following is allowed.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><value val="23"/></expr>
    <expr><attr name="Tier"/><value val="Gold"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/><value val="302370123456789"/></expr>
    <expr><attr name="IMSI"/><op value="="/><value val="206224111222333"/></expr>
    <expr><attr name="MSISDN"/><op value="="/><value val="15145551234"/></expr>
  </where>
```

```
</req>
```

UDR supports as many keys that are allowed for a subscriber in the request.

**NOTE:** For pool based requests, only a single PoolID is allowed. Do not mix PoolID and subscriber key values in the same request, doing so results in an INVALID\_XML error response. For example, the following is not allowed:

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
  :
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/><value val="302370123456789"/></expr>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

## 2.12 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 reqname, see section 5.2.1).

- If the minor CPU usage threshold is crossed (CL1), then UDR rejects select requests
- If the major CPU usage threshold is crossed (CL2), then UDR rejects select, update, operation, and tx (transaction) requests
- If the critical CPU usage threshold is crossed (CL3), then UDR rejects all requests

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

```
<res error="70045" affected="0"/>
```

## 2.13 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 passes 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.

## 2.14 Enterprise Pools

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

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

Pools spanning UDRs support the Enterprise pool feature. With this feature, a pool profile on a pool host UDR can be provisioned as an enterprise pool (the type field set to enterprise in the pool profile). A PSO that is provisioned as an Enterprise pool on the pool host UDR is considered 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.

## Chapter 3. SOAP Interface Description

Oracle Communications User Data Repository supports a SOAP based provisioning interface for management of subscriber data. This interface supports querying, creation, modification and deletion of subscriber and pool data. The SOAP Messages and SOAP Replies are transported over the HTTP protocol.

Each SOAP Message/Reply contains an UDR format XML request/response. The following XML request types are supported:

- Update
- Insert
- Delete
- Select
- Operation

A SOAP provisioning client application is responsible for:

- Establishing a TCP/IP connection with the SOAP server using the VIP for the primary UDR and the SOAP XSAS listening port (as specified in section 2.9).
- Creating and sending SOAP request messages (as specified in section 5.2.1) to the SOAP server.
- Receiving and processing SOAP response messages (as specified in section 5.2.2) received from the SOAP server.
- Detecting and handling connection errors. It is recommended that the TCP keep-alive interval for the client on the TCP/IP connection is set to detect and report a disconnection problem promptly.

### 3.1.1 SOAP Header Format

In SOAP messages, the authentication of the username and password is part of the SOAP envelope header. When the authentication feature is enabled, UDR validates the username and password received in the header of the SOAP request to validate the identity of the user who generated the request. Any requests that do not match valid users are rejected. The username and password provided in the SOAP Header are ignored when Authentication feature is disabled and the request is processed. In the context of SOAP requests, the SOAP header is outlined below in Figure 2.

**NOTE:** Usernames and passwords must not contain:

- Extra spaces
- Additional characters
- New line char

**Figure 2: SOAP Header Format**

```
<SOAP-ENV:Header>
  <ns1:UserName
    SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
    SOAP-ENV:mustUnderstand="1"
    xsi:type=" SOAP-ENC:string"
    xmlns:ns1="http://www.oracle.com/udr/"
    xmlns:SOAP-ENC c="http://schemas.xmlsoap.org/soap/encoding/">[Add UserName
here]</ns1:UserName>
  <ns1:Passwd
    SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
    SOAP-ENV:mustUnderstand="1"
    xsi:type=" SOAP-ENC:string"
    xmlns:ns2="http://www.oracle.com/udr/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">[Add Password
here]</ns1:Passwd>
</ SOAP-ENV:Header>
```

## SOAP Request/Response Format

The SOAP interface uses SOAP as wrapper of XML requests and responses. The detailed format of the request is illustrated in Figure 3, and the response format in Figure 4.

**Figure 3: SOAP Request Format**

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">

  <SOAP-ENV:Body>

    <ns1:processTransaction>

      <![CDATA[REQUEST]]>

    </ns1:processTransaction>

  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

### Example SOAP Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">

  <SOAP-ENV:Body>
    <ns1:processTransaction>
      <![CDATA[
        <req name="insert">
          <ent name="Subscriber"/>
          <set>
            <expr><attr name="MSISDN"/>
              <value val="33628323201"/></expr>
            <expr><attr name="BillingDay"/>
              <value val="12"/></expr>
          </set>
        </req>
      ]]>
    </ns1:processTransaction>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP interface uses the following wrapper for the XML response and error codes. Note that either the `<ns1:message>` or the `<SOAP-ENV:Fault>` element is present, but not both. The contents of the `<SOAP-ENV:Fault>` are dependent on the SOAP error that occurs and can vary, and thus are not listed here:

**Figure 4: SOAP Response Format**

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
<
  <SOAP-ENV:Body>
    <ns1:message error="ErrorCode">
      <![CDATA[RESPONSE]]>
    </ns1:message>
  </SOAP-ENV:Body>
  |
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Fault>
      ...
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
>
</SOAP-ENV:Envelope>

```

### 3.1.2 Status Codes and Error Messages

If an error occurred in processing the request or with the format of the message, an error result code is sent:

- <message error="0">: normal, request transaction was sent and processed
- <message error="0"> but the message content has <res error=error code number ... >. This implies there is a problem with the content of the request message (for example,, a problem with format or value out of range). The Error code numbers are generated by UDR
- <message error="10">: Communication problem, unable to process the request transaction. The response does not contain any other response/error content
- <message error="20">: Unable to parse the request transaction. The response does not contain any other response/error content

Example of a Response message indicating success:

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="0">
      <![CDATA[<req name="insert" resonly="y">
        <res error="0" affected="1"/>
      </req]]>
    </ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Example of a Response message with an error code returned:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="0">
      <! [CDATA[<req name="insert" resonly="y">
        <res error="70019" affected="0"/>
        </req>]]>
      </ns1:message>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

#### Example of a Response message when a communications error occurred:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="10"></ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

#### Example of a Response message when a request parsing failure occurred:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="20"></ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

#### Example of a Response message when a SOAP Fault occurred:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>Method 'processTransaction' not implemented: method name or namespace not
recognized</faultstring>
      <detail></detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### Error Codes

The list of error codes is described in Appendix B.

### Legacy SPR Format SOAP Request/Response

UDR can be configured to operate in a compatibility mode for legacy SPR customers, which affects the SOAP request/response format. See Appendix C for more details.



## Chapter 4. SOAP Interface Message Definitions

### 4.1 Message Conventions

XML message specification syntax follows several conventions to convey what parameters are required or optional and how they and their values must be specified.

**Table 4: Message Conventions**

| Symbol  | Description   |
|---------|---|
| italics | Parameter values that are replaced by an actual parameter name or numeric value.  |
| spaces  | Spaces (zero or more space characters, " ") may be inserted anywhere except in a single name or number. At least one space is required to separate adjacent names or numbers.   |
| ...     | Variable number of repeated entries.<br>For example: dn DN1, dn DN2, ...,dn DN7, dn DN8   |
| <>      | Angle brackets are used to enclose parameter values that are choices or names. For example, in parameter1 <1 2 3>, the numbers represent specific value choices. In parameter2 <ServerName>, the <i>ServerName</i> represents the actual value. In parameter3 <0...3600>, the numbers represent a choice in the range from 0 to 3600. |
| [ ]     | Square brackets are used to enclose an optional parameter and its value, such as [, parameter1 <1 2 3>]. A parameter and its value that are not enclosed in square brackets are mandatory.  |
|         | When the pipe symbol is used in a parameter value list, such as Parameter1 <1 2 3>, it indicates a choice between available values.   |
| ,       | A literal comma is used in the message to separate each parameter that is specified.  |

### 4.2 Basic XML Message Format

#### Request

The following describes the basic layout of an XML request, with all different options and parameters included. UDR requests are made up of different combinations of the parameters. All are listed for illustrative purposes. Proper examples of which parameters are relevant for each request are described in the section that follows.

```
<req name="reqname" [resonly="resonly"] [id="id"] [odk="odk"]>
  <ent name="entityname"/>

  <select>
    <expr><attr name="fieldName"/>
  </select>

  <set>
    <expr><attr name="fieldName"/><value val="fieldValue"/></expr>
    <expr><attr name="fieldName"/><op value="="/><value val="" isnull="y"/></expr>
    <oper name="AddToSet">
```

```

    <expr><attr name="setFieldName"/><value val="setFieldValue"/></expr>
  </oper>

  <oper name="RemoveFromSet">
    <expr><attr name="setFieldName"/><value val="setFieldValue"/></expr>
  </oper>

  <expr><attr name="cdataFieldName"/><op value=""/><cdata>
<![CDATA[
cdataFieldValue
]]></cdata></expr>

</set>

<where>
  <expr><attr name="keyName"/><op value=""/><value val="keyValue"/></expr>
  <expr><attr name="rowKeyName"/><op value=""/><value val="rowKeyValue"/></expr>
  <expr><attr name="instanceFieldName"/><op value=""/>
    <value val="instanceFieldValue"/></expr>
</where>

<oper name="operName">
  <expr><attr name="fieldName"/><value val="fieldValue"/></expr>
</oper>

</req>

```

The reqname attribute indicates what type of request is being sent. Values are either insert, update, delete, select, or operation, depending on the request.

**NOTE:** The resonly attribute controls whether or not the original request is included along with the response. The resonly attribute is optional, and if set to y, then the original request is not included in the response (result only). If resonly is set to n, then the original request IS included in the response. The default value of the flag (when the resonly attribute is not supplied) is n—that is, return the request in the response.

The id attribute is used by the XSAS client to correlate request and response messages. The id attribute is optional and if specified, is an integer between 1 and 4294967295 expressed as a decimal number in ASCII. If the user specifies the id attribute in a request, the same id attribute and value are returned by XSAS in the corresponding response, so a unique id value must be sent in each request message to differentiate responses.

The odk attribute (on duplicate key) allows an insert request to convert the insert request to an update request if the target entity exists. The odk attribute is optional, and if set to yes, then if the entity being inserted exists, the entity is updated instead of the request failing. The default value of the flag (for when the attribute is not supplied) is to not convert the insert request to an update request. Hence, if the target entity exists, the request fails.

The entityname attribute identifies the provisioning entity type on which the request is being performed on. Values are either subscriber, pool, QuotaEntity, or PoolQuotaEntity depending on the request, which match the configured Entity values in the SEC.

The namespace attribute identifies the database namespace in which the data relating to the request is stored. This is not used in UDR, but is retained for backwards compatibility. Value is always set to policy. This attribute is optional, and can be supplied for backwards compatibility with the legacy SPR. Any value supplied is not validated, and ignored.

When a field value is included to be set (for example in an insert/update request), a <set> element is present. In this, one, or many <expr><attr name="fieldName"/><value val="fieldValue"/></expr> elements are present. The fieldName indicates the name of the field being set, and the fieldValue is the value to set it to. When the value of a field is deleted, this is performed by setting the fieldValue as empty (""), and additionally specifying the attribute isnull="y".

**NOTE:** When specifying fields in a <set> element, field order is not important. The fields defined for an entity do not have to be specified in the order they are defined in the SEC.

When a field value is included to be retrieved (for example in a select request), a <select> element is present. In this, one, or many <expr><attr name="fieldName"/></expr> elements are present. The fieldName indicates the name of the field being retrieved. For a select request, at least one field value must be requested. Only the fields requested returns in the response.

When a field is a list type (such as entitlement in profile), an embedded operation request is used to add/remove values from the list. This is performed by including the element <oper name="operName"> where operName is either AddToSet (to add a values to a list) or RemoveFromSet (to remove a values from a list). The name of the field being modified is specified in setFieldName, and the values being added/removed are specified in setFieldValue. Multiple comma separated values can be specified in setFieldValue, or with each individual value in a separate <expr><attr name="setFieldName"/><value val="setFieldValue"/></expr> element.

**NOTE:** The ns attribute is optional, and can be supplied for backwards compatibility with the legacy SPR. Any value supplied is not validated, and ignored.

To set a field as an XML data blob, the cdataFieldValue contains a <cdata> element, and the data in the constructs of an XML CDATA section. The CDATA section starts with <![CDATA[, then the cdataFieldValue containing the XML data blob, and the CDATA section ends with ]]>.

Most commands identify the subscriber for which the provisioning request is being made by specifying the subscriber address in the <where> element. When present, a key type/value must be provided. Depending on the command, keyType can be IMSI, MSISDN, IMEI,NAI, AccountId, or PoolID. The value of the key (of the indicated key type) is set in keyValue.

Depending on the keyType, the keyValue is validated as showin in Table 1.

**Table 5 keyValue Validation for keyType**

| keyType   | keyValue Validation  |
|-----------|--|
| IMEI      | 8 to 14 numeric digits   |
| IMSI      | 10 to 15 numeric digits  |
| MSISDN    | 8 to 15 numeric digits.<br><b>NOTE:</b> A preceding + (plus) symbol is not supported, and is rejected. |
| NAI       | For NAI supported formats see Table 8  |
| AccountId | 1 to 255 characters (allowed values are any ASCII printable characters in the range x20 to x7e)        |
| PoolID    | 1 to 22 numeric digits, minimum value 1  |

When a request is performing an action on a specific row in an entity (such as updating a field value in a specific quota instance), the row key field name used to select the row is specified in rowKeyName. The value of key is specified in rowKeyValue. If a field in the row can indicate uniqueness, in the case of more than one row having the same rowKeyName/rowKeyValue, then this field is specified in instanceFieldName/instanceFieldValue.

When the reqname is set to operation, the <oper> element is present. This defines the operation name in operName.

## XML Comments in a Request

A SOAP request may contain XML comments, such as:

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

- If the comment is in the request, it is ignored.
- If the comment is contained with the XML blob for an opaque entity, in the CDATA constraint, then the comment gets stored in the XML blob.
- If the comment is contained with the XML blob for a transparent entity, in the CDATA constraint, then the comment does not get stored in the XML blob.

## Response

The following describes the basic layout of an XML response, with all different options and parameters included. UDR responses are made up of different combinations of the parameters. All are listed for illustrative purposes. Proper examples of which parameters are relevant for each response are described in the section that follows.

```
<req name="reqname" [resonly="resonly"] [id="id"]>
  originalXMLRequest
  <res error="error" affected="affected"/>
  <rset>
    <row>
      <rv>rowValue</rv>
      <rv>
<![CDATA[
cdataRowValue
]]>
      <rv></rv>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

The `reqname` attribute contains the same value as supplied in the request. Values are either `insert`, `update`, `delete`, `select`, or `operation`, depending on the request.

If the `resonly` attribute was included in the request, the same value is returned in the response.

If the `id` attribute was included in the request, the same value is returned in the response.

The `originalXMLRequest` element is the text of the original XML request that was sent. This is only present if the `resonly="n"` attribute is set in the original request (or the `resonly` attribute was not supplied, as the default value is `n`).

- The `error` attribute indicates the outcome of the request. A value of 0 indicates success. Any other value indicates failure. The possible errors for each request are detailed in the following section for each request. The list of error codes is described in Appendix B.
- The `affected` attribute indicates the number of affected subscribers. A value of 1 (or more) is expected (for success) and 0 for failure.

If a `select` request has been performed (or with some operation requests), result data returned is contained in a `<rset>` (rowset) element. In an `<rset>` can be zero (if matching data was not found) or one `<row>` element (UDR does not support returning multiple `<row>` elements). In a `<row>` are one or more `<rv>` (row value) elements containing a `rowValue` detailing the requested field value. One `<rv>` element corresponds for every `fieldValue` requested in the `select` request. The `<rv>` elements are given in the same order as the `fieldValues` are specified.

## NOTES

- An `<rv>` element can contain an entire XML CDATA section, starting with `<![CDATA[`, then the `cdataRowValue` containing the XML data blob, and the CDATA section ends with `]]>`. If the `<rv>` element represents a valid field that is not present in the XML data blob, then this is indicated with `<rv`

nu11="y">. If the field is present in the XML blob, but has an empty value, this is indicated with <rv></rv>.

- Whenever XML blob data is returned, fields may not be returned in the order they are defined in the SEC. The fields may be returned in any order.

### 4.3 Encoding of Multiple Embedded CDATA Sections

Requests and responses may contain multiple embedded CDATA sections—that is, one CDATA section that completely contains another CDATA section, because the SOAP envelope begins with a CDATA section to contain the XML requests/responses. These requests/responses require special formatting.

Sections Chapter 6 and Chapter 7 describe CDATA sections in the UDR commands without any reference on how these are represented after are they embedded in the SOAP envelope.

The following sections here give examples of the complete SOAP HTTP requests/response to indicate how to format requests when requests are sent to UDR by a provisioning client, and how responses returned by UDR returns to the UDR client.

#### Request

When physically encoding the XML data to be sent, all embedded CDATA start and end sequences must be changed (the opening and closing sequences for the initial CDATA in the <message> element does not need to be changed).

- Replace all embedded occurrences of <![CDATA[ with &lt;t;![CDATA[
- Replace all embedded occurrences of ]]> with ]]&gt;;

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:processTransaction>
      <![CDATA[
        <req name="insert" resonly="y">
          <ent name="Subscriber"/>
          <set>
            <expr><attr name="Quota"/>
              <op value="="/><cdata>&lt;t;![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9999</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
      ]]&gt;</cdata>
    </expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="13123654862"/></expr>
    </where>
  </req>
]]>
```

```

    </ns1:processTransaction>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

## Response

When a response is received, every < and > character in the <message> element is replaced with &lt; and &gt; respectively (including for the initial CDATA).

For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.oracle.com/udr/">
  <SOAP-ENV:Body>
    <ns1:message error="0">
      &lt;![CDATA[
        &lt;?xml version="1.0" encoding="UTF-8"?&gt;
        &lt;req name="select" resonly="y"&gt;
          &lt;res affected="1" error="0"/&gt;
          &lt;rset&gt;
            &lt;row&gt;
              &lt;rv&gt;
                &lt;![CDATA[
                  &lt;?xml version="1.0"?&gt;
                  &lt;usage&gt;
                    &lt;version&gt;1&lt;/version&gt;
                    &lt;quota name="AggregateLimit"&gt;
                      &lt;cid&gt;9999&lt;/cid&gt;
                      &lt;time&gt;3422&lt;/time&gt;
                      &lt;totalVolume&gt;1000&lt;/totalVolume&gt;
                      &lt;inputVolume&gt;980&lt;/inputVolume&gt;
                      &lt;outputVolume&gt;20&lt;/outputVolume&gt;
                      &lt;serviceSpecific&gt;12&lt;/serviceSpecific&gt;
                      &lt;nextResetTime&gt;2010-05-22T00:00:00-05:00&lt;/nextResetTime&gt;
                    &lt;/quota&gt;
                  &lt;/usage&gt;]]]]&gt;
                &lt;/rv&gt;
              &lt;/row&gt;
            &lt;/rset&gt;
          &lt;/req&gt;
        ]]&gt;
      </ns1:message>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

```

## 4.4 Case Sensitivity

The constructs that XML requests are made up of (such as <req>, <ent>, <set>, <where> and so on) are case-sensitive. Exact case must be followed for all the commands described in this document, or the request fails.

For example, the following is valid:

```

<req name="delete">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    </where>
  </req>

```

But the following is not:

```

<req name="delete">
  <Ent name="Subscriber"/>
  <where>

```

```

    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

- Entity names as specified in an *entityName* are not case sensitive.
- Entity field names, key names, and row element/identifiers names are not case-sensitive, for example *fieldName*, *setFieldName*, *keyName*, *instanceFieldName*, and *rowIdName*.
- Entity field values, and key values are case-sensitive, for example *fieldValue*, *setFieldValue*, *keyValue*, *rowIdValue*, and *instanceFieldValue*.
- Operation names as specified in an *operName* are not case sensitive.

### Examples

- When accessing a *fieldName* defined as *inputVolume* in the SEC, then *inputvolume*, *INPUTVOLUME* or *inputVolume* are valid field names. Field names do not have to be specified in a request as they are defined in the SEC

A field name is used to specify an *entire entity* (for example a *fieldName*, *cdataFieldName* or *opaqueDataType*) is also *not* case-sensitive

- When a field is returned in a response, it is returned *as defined in the SEC*. For example, if the above field is created using the name *INPUTVOLUME*, then it returns 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—also known as *rowIdName* (for example for Quota, the element `<quota name="AggregateLimit">` contains the attribute called *name*) the row identifier name is *not* case-sensitive
- When a *rowIdValue* is used to find a row (such as when using the Get Row command), the row identifier value *is* case-sensitive. If an entity contained a row called *DayPass*, and the Get Row command was used to get the row *DAYPASS*, then this fails
- When an *instanceFieldName* is used to find a row (such as when using the Get Row command), the row instance identifier field name is *not* case-sensitive
- When an *instanceFieldValue* is used to find a row (such as when using the Get Row command), the row instance identifier field value *is* case-sensitive. If an entity contained a row called with a field with the value *Data*, and the Get Row command was used to get the row with the field value *DATA*, then this fails
- When a *keyName* is specified in a `<where>` or `<set>` element (such as *MSISDN*), the key name is *not* case-sensitive
- When a *keyValue* is specified in the `<where>` element (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
- When an element in the XML blob contains the row element name (for example for Quota, the row `<quota name="AggregateLimit">` contains the element called *quota*) the row element name is *not* case-sensitive
- When an operation name is specified in an *operName* (such as when using the GetPoolID operation), the operation name is *not* case-sensitive
- When an entity name is specified in an *entityName* (such as when using the Create Row command), the entity name is *not* case-sensitive

## 4.5 List of Messages

The following table provides a list of operations/messages for subscriber data. Each row of the table represents a command.

**Table 6: Summary of Supported Subscriber Commands**

| Operation Data/Type          | Command            | SOAP                                     |
|------------------------------|--------------------|--|
| Subscriber Profile           | Create Profile     | insert                                   |
|                              | Get Profile        | select                                   |
|                              | Delete Profile     | delete                                   |
| Subscriber Field             | Add Field Value    | update (using "AddToSet" operation)      |
|                              | Get Field          | select                                   |
|                              | Update Field       | update                                   |
|                              | Delete Field       | update, null                             |
|                              | Delete Field Value | update (using "RemoveFromSet" operation) |
| Subscriber Opaque Data       | Create Opaque Data | insert                                   |
|                              | Get Opaque Data    | select                                   |
|                              | Update Opaque Data | update                                   |
|                              | Delete Opaque Data | update (using "isnull" attribute)        |
| Subscriber Data Row          | Create Row         | insert                                   |
|                              | Get Row            | select                                   |
|                              | Delete Row         | delete                                   |
| Subscriber Data Row Field    | Get Row Field      | select                                   |
|                              | Update Row Field   | update                                   |
|                              | Delete Row Field   | update (using "isnull" attribute)        |
| Subscriber Data Field        | Create Data Field  | insert                                   |
|                              | Get Data Field     | select                                   |
|                              | Update Data Field  | update                                   |
|                              | Delete Data Field  | update (using "isnull" attribute)        |
| Subscriber Special Operation | Reset Quota        | operation                                |



Table 7 provides a list of operations/messages for pool data. Similar to the previous table, each row of the table represents a command.

**Table 7: Summary of Supported Pool Commands**

| Operation Data/Type      | Command                  | SOAP                                     |
|--------------------------|--------------------------|--|
| Pool Profile             | Create Pool              | insert                                   |
|                          | Get Pool                 | select                                   |
|                          | Delete Pool              | delete                                   |
| Pool Field               | Add Field Value          | update (using "AddToSet" operation)      |
|                          | Get Field                | select                                   |
|                          | Update Field             | update                                   |
|                          | Delete Field             | update (using "isNull" attribute)        |
|                          | Delete Field Value       | update (using "RemoveFromSet" operation) |
| Pool Opaque Data         | Create Opaque Data       | insert                                   |
|                          | Get Opaque Data          | select                                   |
|                          | Update Opaque Data       | update                                   |
|                          | Delete Opaque Data       | update (using "isNull" attribute)        |
| Pool Data Row            | Create Row               | insert                                   |
|                          | Get Row                  | select                                   |
|                          | Delete Row               | delete                                   |
| Pool Data Row Field      | Get Row Field            | select                                   |
|                          | Update Row Field         | update                                   |
|                          | Delete Row Field         | update (using "isNull" attribute)        |
| Pool Data Field          | Create Data Field        | insert                                   |
|                          | Get Data Field           | select                                   |
|                          | Update Data Field        | update                                   |
|                          | Delete Data Field        | update (using "isNull" attribute)        |
| Additional Pool Commands | Add Member to Pool       | operation                                |
|                          | Remove Member from Pool  | operation                                |
|                          | Get Pool Members         | operation                                |
|                          | Get Pool by Member (key) | operation                                |

| <b>Operation Data/Type</b> | <b>Command</b>   | <b>SOAP</b> |
|----------------------------|------------------|-------------|
| Pool Special Operation     | Reset Pool Quota | operation   |

## Chapter 5. UDR Data Model

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

The subscriber-related data includes:

- **Profile and subscriber data**  
Pre-provisioned information that describes the capabilities of each subscriber. This data is typically written by the OSS system (via a provisioning interface) and referenced by the PCRF (via the Sh interface).
- **Quota:**  
Information that represents the use of managed resources (quota, pass, top-up, roll-over) by the subscriber. Although the UDR provisioning interfaces allow quota data to be manipulated, this data is typically written by the PCRF and only referenced using the provisioning interfaces.
- **State:**  
Subscriber-specific properties. Like quota, this data is typically written by the PCRF, and referenced using the provisioning interfaces.
- **Dynamic Quota:**  
Dynamically configured information related to managed resources (pass, top-up). 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 (quota, pass, top-up, roll-over) by the pool
- **Pool State:** Pool-specific properties
- **Pool Dynamic Quota:** dynamically configured information related to managed resources (pass, top-up)
- **Pool Membership:** List of subscribers that are associated with a pool

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

The Index contains information on the following:

- **Subscription**
  - o A subscription exists for every individual subscriber
  - o Maps a subscription to the user identities through which it can be accessed
  - o Maps an individual subscription to the pool of which they are a member

- Pool Subscription
  - o A pool subscription exists for every pool
  - o Maps a pool subscription to the pool identity through which it can be accessed
  - o 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

  - o IMSI, MSISDN,IMEI, NAI and AccountId map to an individual subscription
  - o 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:
  - o Each individual subscriber

Defined entities stored in the registers are:

    - Profile
    - Quota
    - State
    - Dynamic Quota
  - o 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 or pool data. The indexing system allows access to the SDO via IMSI, MSISDN, IMEI, NAI or AccountId. The pool SDO can be accessed via PoolID.

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

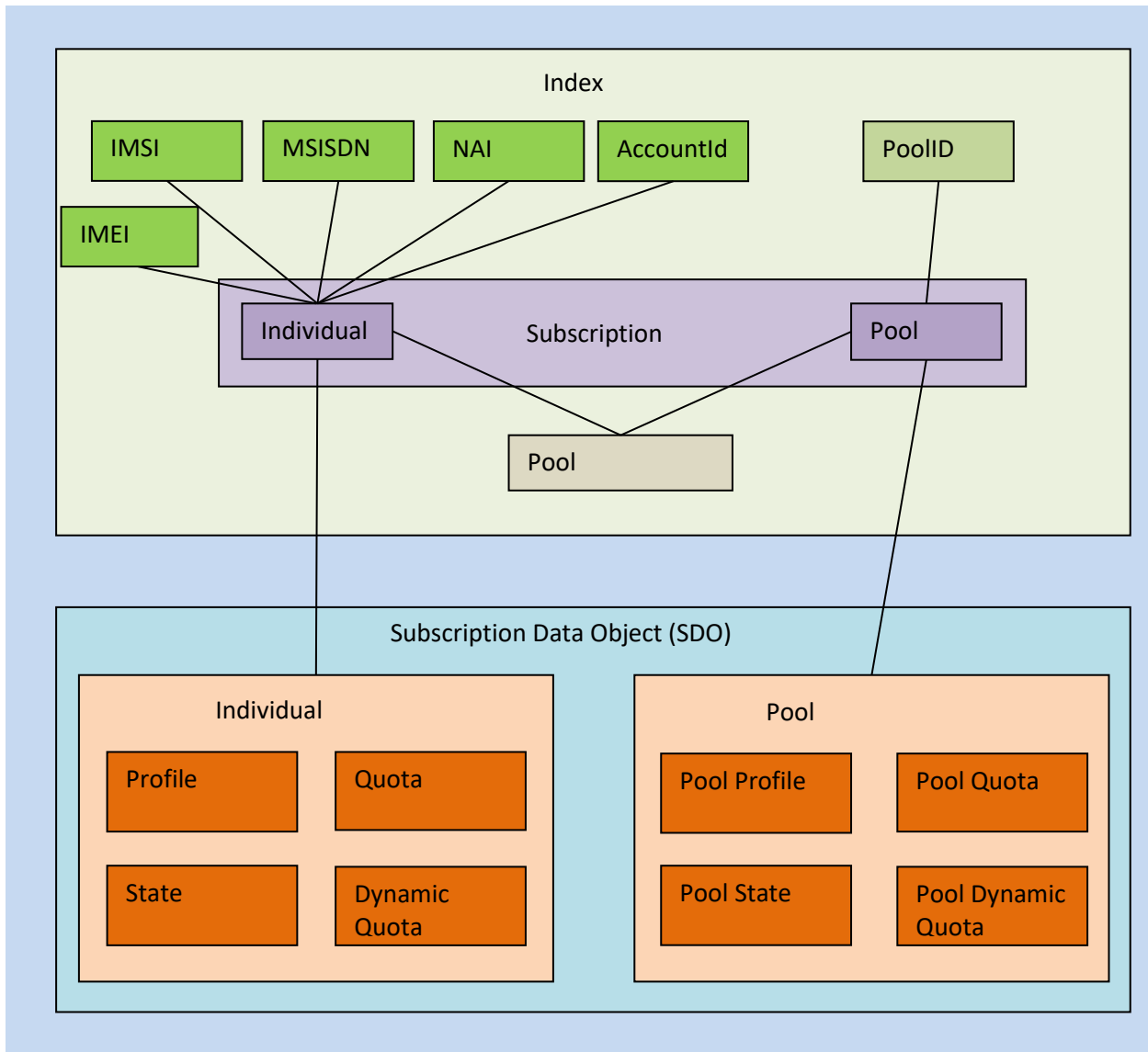
A field in an entity can have a default value. If an entity is created, and the field is not specified, it are created with the default value.

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 creation fails.

A field in an entity can have a reset value. If a reset command is used on the entity, 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.

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

**Figure 5: Data Model**



## 5.1 Subscriber Data

### 5.1.1 Subscriber Profile

The subscriber profile represents the identifying attributes associated with the user. In addition to the base fields indicated their level of service, it also includes a set of custom fields that the 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 following sequence of attributes. Each record must have at least one key value: 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.

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

**Table 8: Subscriber Profile Entity Definition**

| Name (XML tag)    | Type   | Description  |
|-------------------|--------|--|
| <b>subscriber</b> |        | <b>Sequence (multiplicity = 1)</b>   |
| 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.   |
| 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.   |
| IMSI              | String | List of IMSIs (10 to 15 numeric digits). A separate entry is included for each IMSI associated with the profile for the subscriber.  |
| NAI               | String | <p>User or Domain length is between 0 to 63 characters</p> <p><b>NOTE:</b> The limitation for 0 to 63 characters is because a NAI beyond 63 characters may not be possible to transfer through all devices. The user needs to ensure the combination of User and Domain does not exceed 63 characters (not including the @ character).</p> <p>List of NAIs. A separate entry is included for each NAI associated with the profile for the subscriber. Allowable formats are user@domain, user or @domain</p> <p>The user or domain can be empty.</p> <p>Characters that are allowed for the user: !, %, \$, A to Z, a to z, 0 to 9, ., -, _, /, *, =, ^, ` ,  , #, ', +, ?, {, }, ~</p> <p>Characters that are allowed for the domain: A to Z, a to z, 0 to 9, ., -, _</p> <p><b>Example NAI Formats</b></p> <pre> bob@privatecorp.example.net fred\$@example.com eng.example.net!nancy@example.net eng%nancy@example.net bob#+ ?@example.net </pre> |
| AccountId         | String | <p>Any string that can be used to identify the account for the subscriber (1 to 255 characters).</p> <p>Allowed values are any ASCII printable character, values x20 to x7e.</p>   |
| BillingDay        | String | <p>Allowed values are 0 to 31.</p> <p>The day of the month 1 to 31 when the associated quota for the subscriber is reset.</p> <p>0 indicates that the default value configured at the PCRF level is used. This is automatically set in any record where BillingDay is not specified.</p>   |
| Entitlement       | String | List of entitlements. A separate entry is included for each entitlement associated with the profile for the subscriber.  |
| Tier              | String | Tier for the subscriber.   |
| Custom1           | String | Fields used to store customer-specific data.   |
| Custom2           | String | Fields used to store customer-specific data.   |
| Custom3           | String | Fields used to store customer-specific data.   |
| Custom4           | String | Fields used to store customer-specific data.   |

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

### 5.1.2 Quota

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

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

#### NOTES

- The Quota entity contains a version number. Different attributes may be present based on the version number value of the entity being accessed. In UDR, only v3 of Quota is supported.
- The default value given in the table is used either:
  - o When a Quota instance is created, and a value is not supplied for the field. In this case, the field is created with the value indicated
  - o When a Quota instance is reset using the Reset command. If a field is defined as resettable, and the field exists, then it is set to the value indicated. If the field does not exist in the Quota, it is *not* created.

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

**Table 9: Quota Entity Definition**

| Name (XML tag)         | Type   | Default Value   | Description  | Quota Versions |
|------------------------|--------|-----------------|--|----------------|
| <b>usage</b>           |        |                 | <b>Sequence (multiplicity is 1)</b>  | 1/2/3          |
| version                | String |                 | Version of the schema.   | 1/2/3          |
| <b>quota</b>           |        |                 | <b>Sequence (multiplicity is N)</b>  | 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.<br>See section 5.3 for format details.  | 1/2/3          |
| Type                   | String | Empty string "" | Type of the resource in use.   | 2/3            |
| grantedTotalVolume     | String | "0"             | For pool quota, Granted Total Volume, represents the granted total volume of all the subscribers in the pool. For individual quota, it represents the granted volume to all the PDN connections for that subscriber. | 2/3            |
| grantedInputVolume     | String | "0"             | Granted Input Volume.  | 2/3            |
| grantedOutputVolume    | String | "0"             | Granted Output Volume.   | 2/3            |
| grantedTime            | String | Empty string "" | Granted Total Time.  | 2/3            |
| grantedServiceSpecific | String | Empty string "" | Granted Service Specific Units.  | 2/3            |
| QuotaState             | String | Empty string "" | State of the resource in use.  | 3              |
| RefInstanceid          | String | Empty string "" | Instance-id of the associated provisioned pass, top-up or roll-over.   | 3              |



### 5.1.3 State

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

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

**NOTE:** The default fields configured are not:

- Resettable
- Defaultable

The State entity supports the following sequence of attributes:

**Table 10: State Entity Definition**

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

### 5.1.4 Dynamic Quota

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

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

**NOTE:** The default fields configured are not:

- Resettable
- Defaultable

The DynamicQuota entity supports the following sequence of attributes:

**Table 11: Dynamic Quota Entity Definition**

| Name (XML tag)      | Type   | Description   |
|---------------------|--------|---|
| <b>definition</b>   |        | <b>Sequence (multiplicity is 1)</b>   |
| version             | String | Version of the schema   |
| <b>DynamicQuota</b> |        | <b>Sequence (multiplicity is N)</b>   |
| 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.  |

| Name (XML tag)           | Type   | Description  |
|--------------------------|--------|--|
| Priority                 | String | An integer represented as a string. This number allows service providers to specify when one pass or top-up is used before another pass or top-up.                         |
| InitialTime              | String | An integer represented as a string. The number of seconds initially granted for the pass/top-up.   |
| InitialTotalVolume       | String | An integer represented as a string. The number of bytes of total volume initially granted for the pass/top-up.   |
| InitialInputVolume       | String | An integer represented as a string. The number of bytes of input volume initially granted for the pass/top-up.   |
| InitialOutputVolume      | String | An integer represented as a string. The number of bytes of output volume initially granted for the pass/top-up.  |
| InitialServiceSpecific   | String | An integer represented as a string. The number of service specific units initially granted for the pass/top-up.  |
| activationdatetime       | String | The date/time after which the pass or top-up may be active.<br>See section 5.3 for format details.   |
| expirationdatetime       | String | The date/time after which the pass or top-up is considered to be exhausted.<br>See section 5.3 for format details.   |
| purchasedatetime         | String | The date/time when a pass was purchased.<br>See section 5.3 for format details.  |
| Duration                 | String | The number of seconds after first use in which the pass must be used or expired. If both Duration and expirationdatetime are present, the closest expiration time is used. |
| InterimReportingInterval | String | The number of seconds after which the GGSN/DPI/Gateway revalidates quota grants with the PCRF.   |

## 5.2 Pool Data

### 5.2.1 Pool Profile

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

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

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

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

**Table 12: Pool Profile Entity Definition**

| Name (XML tag) | Type   | Description   |
|----------------|--------|---|
| <b>pool</b>    |        | <b>Sequence (multiplicity is 1)</b>   |
| 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.<br>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.<br>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.  |

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

### 5.2.2 Pool Quota

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

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

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

### 5.2.3 Pool State

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

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

**NOTE:** The default fields configured are not:

- Resetable
- Defaultable

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

### 5.2.4 Pool Dynamic Quota

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

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

**NOTE:** The default fields configured are not:

- Resetable
- Defaultable

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

## 5.3 Date/Timestamp Format

The Date/Timestamp format used by many fields is:

`CCYY-MM-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

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

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

## Chapter 6. Subscriber Provisioning

**NOTE:** For command responses, the error code values described are listed in Appendix B.

### 6.1 Subscriber Profile Commands

**Table 13: Summary of Subscriber Profile Commands**

| Command        | Description   | Keys                         | Command Syntax  |
|----------------|---|------------------------------|---|
| Create Profile | Create a subscriber or subscriber profile   |                              | <code>&lt;req name="insert"&gt;<br/>&lt;ent name="Subscriber"/&gt;</code> |
| Get Profile    | Get subscriber profile data   | MSISDN,<br>IMSI,             | <code>&lt;req name="select"&gt;<br/>&lt;ent name="Subscriber"/&gt;</code> |
| Delete Profile | Delete all subscriber profile data and all opaque data associated with the subscriber | IMEI, NAI<br>or<br>AccountId | <code>&lt;req name="delete"&gt;<br/>&lt;ent name="Subscriber"/&gt;</code> |

#### 6.1.1 Create Profile

##### 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 request as part of the `where` element. 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.

##### NOTES

- The subscriber profile data provided is fully validated against the definition in the SEC. If the validation check fails, then the request is rejected.
- An entire entity for the subscriber can be created by specifying a `cdataFieldName` corresponding to the interface entity name in the SEC, and supplying the entire XML blob value in `cdataFieldValue`.
- 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

```
<req name="insert" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="keyName1"/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><value val="keyValueN"/></expr>
  ]
  [
    <expr>
    <
      <attr name="fieldName1"/><value val="fieldValue1"/>
    |
      <attr name="cdataFieldName1"/><op value=""/>
      <cdata><![CDATA[CDATA[cdataFieldValue1]]></cdata>
    >
  ]
</expr>
```

```

    <expr>
  <
    <attr name="fieldName2"/><value val="fieldValue2"/>
  |
    <attr name="cdataFieldName2"/><op value="="/>
    <cdata><![CDATA[ cdataFieldValue2 ]></cdata>
  >
  </expr>
  :
  <expr>
  <
    <attr name="fieldNameN"/><value val="fieldValueN"/>
  |
    <attr name="cdataFieldNameN"/><op value="="/>
    <cdata><![CDATA[ cdataFieldValueN ]></cdata>
  >
  </expr>
]
</set>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- **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*
- **cdataFieldNameX:** A user defined field in the subscriber profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC  
Value is either Quota, State, or DynamicQuota
- **cdataFieldValueX:** Contents of the XML data blob for *cdataFieldNameX*

## NOTES

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

## Response

```

<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see Appendix A for other values
- *affected*: The number of subscribers created by the request. A value of 1 is expected for success

#### Error Codes 1: Create Profile

| Error Code            | Description   |
|-----------------------|---|
| FIELD_VAL_INVALID     | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION  | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| INVALID_SOAP_XML      | Invalid SOAP XML  |
| FIELD_UNDEFINED       | Field not defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_EXISTS            | Key exists. A subscriber or pool exists with the given key  |
| MULT_VER_TAGS_FOUND   | Multiple Version Tags Found   |
| INVAL_REPEATABLE_ELEM | Invalid Repeatable Element  |
| INTF_ENTY_NOT_FOUND   | Interface Entity Not Found  |
| INVALID_XML           | Invalid Input XML   |
| AE_KEY_EXISTS         | An AE subscriber exists with the given key. Only applicable when option enableAEKeyAlreadyExistsErrCode is enabled                      |

#### Create Profile Examples

##### Request 1

A subscriber is created, with an AccountId, MSISDN, and IMSI keys. The BillingDay and Entitlement fields are set. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="AccountId"/><value val="10404723525"/></expr>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
  </set>
</req>
```

##### Response 1

The request is successful, and the subscriber was created.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```



**Request 2**

A subscriber is created, with an AccountId, MSISDN and IMSI keys. Another subscriber exists with the given IMSI.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="AccountId"/><value val="10404723525"/></expr>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPassPlus"/></expr>
  </set>
</req>
```

**Response 2**

The request fails. The error value indicates a subscriber exists with the given IMSI, and the affected rows are 0.

```
<req name="insert" resonly="y">
  <res error="70020" affected="0"/>
</req>
```

**Request 3**

A subscriber is created, with an AccountId, MSISDN and IMSI keys. The BillingDay and Entitlement fields are set. The request is not required in the response. Provisioning has been disabled.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="AccountId"/><value val="10404723525"/></expr>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
  </set>
</req>
```

**Response 3**

The request fails. The error value indicates that provisioning has been disabled.

```
<req name="insert" resonly="y">
  <res error="70031" affected="0"/>
</req>
```

**Request 4**

A subscriber is created, with an AccountId, 2 MSISDNs and IMSI keys. The BillingDay and Entitlement fields are set. The Quota and State entities are also created. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="AccountId"/><value val="178322212122"/></expr>
    <expr><attr name="MSISDN"/><value val="15145551234,15141234567"/></expr>
    <expr><attr name="IMSI"/><value val="302370123456789"/></expr>
    <expr><attr name="BillingDay"/><value val="6"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
    <expr><attr name="Quota"/><op value=""/>
      <CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Weekend">
            <totalVolume>100</totalVolume>
            <Type>quota</Type>
            <QuotaState>active</QuotaState>
            <nextResetTime>2014-01-10T02:00:00</nextResetTime>
```

```

                </quota>
                <quota name="Evenings">
                    <totalVolume>100</totalVolume>
                    <Type>quota</Type>
                    <QuotaState>active</QuotaState>
                    <nextResetTime>2014-02-01T00:00:00</nextResetTime>
                </quota>
            </usage]]>
        </cdata>
    </expr>
    <expr><attr name="State"/><op value="="/>
        <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
            <state>
                <version>1</version>
                <property>
                    <name>mcc</name>
                    <value>302</value>
                </property>
                <property>
                    <name>expire</name>
                    <value>2014-02-09T11:20:32</value>
                </property>
            </state>]]>
        </cdata>
    </expr>
</set>
</req>

```

**Response 4**

The request is successful, and the subscriber was created.

```

<req name="insert" resonly="y">
    <res error="0" affected="1"/>
</req>

```

**Request 5**

A subscriber is created, with an invalid username and/or password in the SOAP header. The request is not required in the response.

```

<soapenv:Header>
<ns1:UserName
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="1"
    xsi:type="soapenc:string"
    xmlns:ns1="http://www.oracle.com/udr/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">[bad UserName]</ns1:UserName>
<ns1:Passwd
    soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
    soapenv:mustUnderstand="1"
    xsi:type="soapenc:string"
    xmlns:ns2="http://www.oracle.com/udr/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">[invalid Password]</ns1:Passwd>
</soapenv:Header>
<req name="insert" resonly="y">
    <ent name="Subscriber"/>
    <set>
        <expr><attr name="AccountId"/><value val="10404723525"/></expr>
        <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
        <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
        <expr><attr name="BillingDay"/><value val="1"/></expr>
        <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
    </set>
</req>

```

**Response 5**

The request fails. The error value indicates that username/password authentication failed.

```

<req name="insert" resonly="y">

```

```
<res error="70054" affected="0"/>
</req>
```

**Request 6**

A subscriber is created, with an AccountId, MSISDN and IMSI keys. An AE subscriber exists with the given IMSI and enableAEKeyAlreadyExistsErrCode option is enabled.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="AccountId"/><value val="10404723525"/></expr>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPassPlus"/></expr>
  </set>
</req>
```

**Response 6**

The request fails. The error value indicates an AE subscriber exists with the given IMSI, and the affected rows are 0.

```
<req name="insert" resonly="y">
  <res error="70055" affected="0"/>
</req>
```

**Request 7**

A subscriber is created, with an AccountId, MSISDN and IMSI keys. An AE subscriber exists with the given IMSI and “enableAEKeyAlreadyExistsErrCode” option is disabled (the default case).

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="AccountId"/><value val="10404723525"/></expr>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="IMSI"/><value val="184569547984229"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPassPlus"/></expr>
  </set>
</req>
```

**Response 7**

The request fails. The error value indicates a subscriber exists with the given IMSI, and the affected rows are 0.

```
<req name="insert" resonly="y">
  <res error="70020" affected="0"/>
</req>
```

**Request 8**

A subscriber is created, with IMEI and IMSI keys. The request is not required in the response.

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

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr>
      <attr name="IMEI"/>
      <value val="98765439876546"/>
    </expr>
    <expr>
      <attr name="IMSI"/>
```

```

    <value val="987654398765460"/>
  </expr>
</set>
</req>

```

**Response 8**

The request is successful, and the subscriber was created.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 9**

A subscriber is created with IMEI range as key. The request is not required in the response.

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

```

<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr>
      <attr name="IMEI"/>
      <value val="441111111111111"/>
    </expr>
    <expr>
      <attr name="IMEI"/>
      <value val="444411111111111"/>
    </expr>
    <expr>
      <attr name="WL"/>
      <value val="1"/>
    </expr>
    <expr>
      <attr name="BL"/>
      <value val="0"/>
    </expr>
    <expr>
      <attr name="GL"/>
      <value val="0"/>
    </expr>
  </set>
</req>

```

**Response 9**

The request is successful, and the subscriber was created.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**6.1.2 Get Profile****Description**

This operation retrieves all field-value pairs created for a subscriber that is identified by the Keys specified in keyNameX and keyValueX.

The keyNameX and keyValueX values 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 the Keys of the keyNameX/keyValueX values supplied must exist.

## Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *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*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

## Response

```
<req name="select" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
  [
    <rset>
      <row>
        <rv>
          <![CDATA[CDATA[CDATA[cdataRowValue]]>
        </rv>
      </row>
    </rset>
  ]
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers returned. A value of 1 is expected for success

*cdataRowValue*: Contents of the subscriber profile XML data blob

**NOTE:** The *<rset>* (row set) element is optional. It is only present if the request was successful. Only a single *<row>* element is returned, with a single *<rv>* (row value) element containing an XML CDATA construct containing the requested subscriber profile data (XML blob).

**Error Codes 2: Get Profile**

| Error Code              | Description  |
|-------------------------|--|
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber             |

**Get Profile Examples****Request 1**

A request is made to get profile data for a subscriber. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the subscriber profile data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?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">Weekpass</field>
            <field name="Entitlement">DayPass</field>
          </subscriber>]]>
      </rv>
    </row>
  </rset>
</req>
```

**Request 2**

A request is made to get profile data for a subscriber. An IMSI and MSISDN are supplied, and both keys are valid for the same subscriber. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, and the subscriber profile data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
```

```

<rset>
  <row>
    <rv>
      <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <subscriber>
          <field name="MSISDN">15145551234</field>
          <field name="IMSI">302370123456789</field>
          <field name="BillingDay">6</field>
          <field name="Tier">Gold</field>
          <field name="Entitlement">Weekpass</field>
        </subscriber>]]>
    </rv>
  </row>
</rset>
</req>

```

**Request 3**

A request is made to get profile data for a subscriber. The request is not required in the response.

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

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="98765439876547"/></expr>
  </where>
</req>

```

**Response 3**

The request is successful, and the subscriber profile data is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <subscriber>
            <field name="IMEI">98765439876547</field>
            <field name="IMSI">987654398765470</field>
            <field name="BL">0</field>
            <field name="GL">0</field>
            <field name="SV">00</field>
            <field name="WL">1</field>
          </subscriber>]]>
      </rv>
    </row>
  </rset>
</req>

```

**Request 4**

A request is made to get profile data for a subscriber. The request is not included in the response.

These are examples with IMEI range for UDR for DSR based EIR solution.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="4411111111111111"/></expr>
  </where>
</req>

```

**Response 4**

The request is successful, and the subscriber profile data is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <subscriber>
            <field name="IMEI">4411111111111</field>
            <field name="IMEI">4444111111111</field>
            <field name="BL">0</field>
            <field name="GL">0</field>
            <field name="WL">1</field>
          </subscriber>]]>
        </rv>
      </row>
    </rset>
  </req>

```

**Request 5**

A request is made to get profile data for a subscriber. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="441111111111122"/></expr>
  </where>
</req>

```

**Response 5**

The request is successful since the IMEI subscriber lies in the range and the subscriber profile data is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <subscriber>
            <field name="IMEI">4411111111111</field>
            <field name="IMEI">4444111111111</field>
            <field name="BL">0</field>
            <field name="GL">0</field>
            <field name="WL">1</field>
          </subscriber>]]>
        </rv>
      </row>
    </rset>
  </req>

```

**6.1.3 Delete Profile****Description**

This operation deletes all profile data (field-value pairs) and opaque data for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

**Prerequisites**

A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.

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

**Request**

```

<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>

```



```

<where>
  <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
  <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
  :
  <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
</where>
</req>

```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *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*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

**Response**

```

<req name="delete" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>

```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values:A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers deleted. A value of 1 is expected for success.

**Error Codes 3: Delete Profile**

| Error Code              | Description   |
|-------------------------|---|
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| SUB_IN_POOL             | Subscriber is pool member. The subscriber is a member of a pool. A subscriber cannot be deleted if they are a pool member |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

**Delete Profile Examples****Request 1**

The subscriber with the given MSISDN is deleted. The subscriber exists. The request (by default) is included in the response.

```
<req name="delete">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the original request is included in the response.

```
<req name="delete">
  <req name="delete">
    <ent name="Subscriber"/>
    <where>
      <expr><attr name="MSISDN"/><op value="="/>
        <value val="33123654862"/></expr>
    </where>
  </req>
  <res error="0" affected="1"/>
</req>
```

**Request 2**

The subscriber with the given MSISDN is deleted. The subscriber does not exist. The request is not included in the response.

```
<req name="delete">
  <ent name="Subscriber" resonly="y"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123655555"/></expr>
  </where>
</req>
```

**Response 2**

The request fails. The error value indicates a subscriber with the given MSISDN does not exist, and the affected rows are 0. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70019" affected="0"/>
</req>
```

**Request 3**

The subscriber with the given MSISDN and IMSI is deleted. Subscribers exist with the specified MSISDN and IMSI, but they are not the same subscriber. The request is not included in the response.

```
<req name="delete">
  <ent name="Subscriber" resonly="y"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123655555"/></expr>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
  </where>
</req>
```

**Response 3**

The request fails. The error value indicates a subscriber with the given MSISDN and IMSI does not exist, and the affected rows are 0. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70043" affected="0"/>
</req>
```

**Request 4**

The subscriber with the given IMEI is deleted. The subscriber exists. The request (by default) is included in the response.

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

```
<req name="delete">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="98765439876547"/></expr>
  </where>
</req>
```

**Response 4**

The request is successful, and the original request is included in the response.

```
<req name="delete">
  <ent name="Subscriber"/>
  <where>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="98765439876547"/>
    </expr>
  </where>
  <res error="0" affected="1"/>
</req>
```

**Request 5**

The subscriber with the given IMEI range is deleted. The subscriber exists. The request (by default) is included in the response.

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

```
<req name="delete" resonly="y">
  <ent name="Subscriber"/>
  <where>
    <expr>
      <attr name="IMEI"/>
      <op value="="/>
      <value val="4411111111111111"/>
    </expr>
    <expr>
      <attr name="IMEI"/>
      <op value="="/>
      <value val="4444111111111111"/>
    </expr>
  </where>
</req>
```

**Response 5**

The request is successful, and the original request is included in the response.

```
<req name="delete" resonly="y">
  <ent name="Subscriber"/>
  <where>
    <expr>
```

```

    <attr name="IMEI"/>
    <op value=""/>
    <value val="4411111111111111"/>
  </expr>
</expr>
<attr name="IMEI"/>
<op value=""/>
<value val="4444111111111111"/>
</expr>
</where>
</req>

```

## 6.2 Subscriber Profile Field Commands

Table 14: Summary of Subscriber Profile Field Commands

| Command            | Description  | Keys                                 | Command Syntax  |
|--------------------|--|--------------------------------------|---|
| Add Field Value    | Add a value to the specified field. This operation does not affect any pre-existing values for the field | MSISDN, IMSI, IMEI, NAI or AccountId | <pre> &lt;req name="update"&gt;   &lt;ent name="Subscriber"/&gt;   &lt;oper name="AddToSet"&gt; </pre>            |
| Get Field          | Retrieve the values for the specified field  |                                      | <pre> &lt;req name="select"&gt;   &lt;ent name="Subscriber"/&gt; </pre>   |
| Update Field       | Update fields to the specified values  |                                      | <pre> &lt;req name="update"&gt;   &lt;ent name="Subscriber"/&gt; </pre>   |
| Delete Field       | Delete all the values for the specified fields   |                                      | <pre> &lt;req name="update"&gt;   &lt;ent name="Subscriber"/&gt;   ... &lt;value val="" isnull="y"/&gt;... </pre> |
| Delete Field Value | Delete a value for the specified field   |                                      | <pre> &lt;req name="update"&gt;   &lt;ent name="Subscriber"/&gt;   &lt;oper name="RemoveFromSet"&gt; </pre>       |

### 6.2.1 Add Field Value

#### Description

This operation adds one or more values to the specified multi-value field for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

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

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

#### NOTES

- If a value being added exists, the request fails.
- 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 being a,b,c,A
- A request to add fields values can also be mixed with a request to update or delete a fields. But, the same field for which an AddToSet operation is being performed cannot also be updated or deleted, else the request fails.
- A request to add fields values using the AddToSet operation can also contain a RemoveFromSet operation to delete fields values. If both operations are included in the same request, the AddToSet is performed before the RemoveFromSet, irrespective of the order in which they are supplied.

## Prerequisites

- A subscriber with the Keys of the *keyNameX/keyValueX* values supplied must exist.
- The field *fieldName* must be a valid field in the subscriber profile, and must be a multi-value field.
- Each *fieldValueX* being added must not be present in the field.

## Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="fieldName1"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    [
      <expr><attr name="fieldName2"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
      :
      <expr><attr name="fieldNameX"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    ]
    </oper>
  </set>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  </where>
</req>
```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

### Values:

- o **y**—Provides the result only, does not include the original request
- o **n**—Includes the original request in the response (default)
- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

## NOTES

- One or more *fieldValueX* values for a *fieldNameX* can be supplied. To add more than one value, either supply a comma separated list of values, or include multiple `<expr>` elements for the field.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

## Response

```
<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers updated. A value of 1 is expected for success

**Error Codes 4: Add Field Value**

| Error Code              | Description   |
|-------------------------|---|
| OCC_CONSTR_VIOLATION    | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field                       |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE     | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| VALUE_EXISTS            | List value added exists   |
| FLD_NOT_MULTI           | Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

**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. The request is not required in the response. An id value is supplied, which is required in the response.

```
<req name="update" resonly="y" id="13579">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the value was added to the Entitlement field. The original request is not included. The id value was included.

```
<req name="update" resonly="y" id="13579">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to add the values HighSpeed and Unlimited to the Entitlement field. The Entitlement field is a valid multi-value field. Neither value is present in the Entitlement field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="HighSpeed"/></expr>
      <expr><attr name="Entitlement"/><value val="Unlimited"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, and the values were added to the Entitlement field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 3**

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 present in the Entitlement field. The request is not required in the response. An id value is supplied, which is required in the response.

```
<req name="update" resonly="y" id="13579">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

**Response 3**

The request fails. The error value indicates the given value is present, and the affected rows are 0. The original request is not included. The id value was included.

```
<req name="update" resonly="y" id="13579">
  <res error="70033" affected="0"/>
</req>
```

**Request 4**

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

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Tier"/><value val="Gold"/></expr>
    </oper>
  </set>
</req>
```

```

    </oper>
  </set>
</where>
  <expr><attr name="MSISDN"/><op value="="/>
    <value val="33123654862"/></expr>
</where>
</req>

```

**Response 4**

The request fails. The error value indicates the field is not a multi-value field, and the affected rows are 0. The original request is not included.

```

<req name="update" resonly="y">
  <res error="70034" affected="0"/>
</req>

```

**Request 5**

A request is made to add the keys 14161234567 and 19191112222 to the MSISDN field, and also add the DayPass value to the Entitlement field. The subscriber has a single MSISDN value of 15145551234. The request is not required in the response.

```

<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="MSISDN"/><value val="14161234567"/></expr>
      <expr><attr name="MSISDN"/><value val="19191112222"/></expr>
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
  </where>
</req>

```

**Response 5**

The request is successful, and the values were added to the MSISDN and Entitlement fields. The subscriber has 3 MSISDN values of 15145551234, 14161234567, and 19191112222. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 6**

A request is made to add the value IMSI field. The IMSI field is a valid multi-value field. There exists only one value for the IMSI field. The request is not required in the response. An id value is supplied, which is required in the response.

```

<req name="update" resonly="y" id="13579">
  <ent name="Subscriber"/>
  <set>
    <oper name="AddToSet">
      <expr> <attr name="IMSI"/><value val="987654398765451"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="98765439876545"/></expr>
  </where>
</req>

```



**Response 6**

The request is successful and the IMSI value was added to the IMSI field. The original request is not included. The id value was included.

```
<req name="update" resonly="y" id="13579">
  <res error="0" affected="1"/>
</req>
```

**6.2.2 Get Field****Description**

This operation retrieves the values for the specified fields for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

**NOTE:** An entire entity for the subscriber can be retrieved by specifying an *opaqueDataType* corresponding to the interface entity name in the SEC.

**Prerequisites**

A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.

Each requested field fieldNameX must be a valid field in the subscriber profile.

Each requested opaqueDataTypeX must reference a valid Entity in the Interface Entity Map table in the SEC.

**Request**

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="fieldName1"/></expr>
  [
    <expr><attr name="fieldName2"/></expr>
    :
    <expr><attr name="fieldNameN"/></expr>
  ]
  [
    <expr><attr name="opaqueDataType1"/></expr>
    <expr><attr name="opaqueDataType2"/></expr>
    :
    <expr><attr name="opaqueDataTypeN"/></expr>
  ]
  </select>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *fieldNameX*: A user defined field in the subscriber profile
- *opaqueDataTypeX*: A user defined field in the subscriber profile, that represents a transparent or opaque data entity  
Value is either Quota, State, or DynamicQuota
- *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*

## NOTES

- At least one *fieldNameX*/*opaqueDataTypeX* field must be requested
- The order in which *fieldNameX*/*opaqueDataTypeX* are specified in the request is not important
- Multiple subscriber key values can be supplied. See section 2.11 for details.

## Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
[
  <rv>rowValue1</rv> | <rv null="y"> | <rv></rv> >
  <rv>rowValue2</rv> | <rv null="y"> | <rv></rv> >
  :
  <rv>rowValueN</rv> | <rv null="y"> | <rv></rv> >
]
[
  <rv>cdataRowValue1</rv> | <rv null="y"> >
  <rv>cdataRowValue2</rv> | <rv null="y"> >
  :
  <rv>cdataRowValueN</rv> | <rv null="y"> >
]
    </row>
  </rset>
]
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers returned. A value of 1 is expected for success
- *rowValueX*: The value of the requested field (for normal fields, not for opaque/transparent entities)  
**NOTE:** For multi-value fields, the value contains a comma separated list of values on a single line. For example, a,b,c
- *cdataRowValueX*: Contents of the XML data blob (for requested fields that are opaque/transparent entities)

**NOTE:** The *<rset>* (row set) element is optional. It is only present if the request was successful. Only a single *<row>* element is returned. One *<rv>* (row value) element exists for every *fieldNameX* or *opaqueDataTypeX*

supplied in the original request. The <rv> elements are ordered the same as the fieldNameX/opaqueDataTypeX fields were specified in the original request. If the field is valid, but not present in the entity, this is indicated with <rv null="y">. If the field is present, but has an empty value, this is indicated with <rv></rv>.

#### Error Codes 5: Get Field

| Error Code              | Description   |
|-------------------------|---|
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found                      |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber                                  |

#### Get Field Examples

##### Request 1

A request is made to get the MSISDN, Entitlement, Tier, and BillingDay fields. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="MSISDN"/></expr>
    <expr><attr name="Entitlement"/></expr>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="BillingDay"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

##### Response 1

The request is successful, and the 4 requested values are returned (the Entitlement is a multi-value field). The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>33123654862</rv>
      <rv>DayPass,WeekPass,Weekend</rv>
      <rv>Prepaid</rv>
      <rv>23</rv>
    </row>
  </rset>
</req>
```

##### Request 2

A request is made to get the IMSI, Entitlement, Tier, and Custom20 fields. The Entitlement and Tier fields are set in the XML blob, the IMSI field is not set, and the Custom20 field is set, but has an empty value. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="IMSI"/></expr>
    <expr><attr name="Entitlement"/></expr>
```

```

    <expr><attr name="Tier"/></expr>
    <expr><attr name="Custom20"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="=" />
      <value val="33123654862"/></expr>
  </where>
</req>

```

### Response 2

The request is successful, and the 4 requested values are returned (the Entitlement is a multi-value field). The IMSI field is indicated as unset, and the Custom20 field is indicated as empty. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
      <rv>1,14,2,8</rv>
      <rv>Prepaid</rv>
      <rv></rv>
    </row>
  </rset>
</req>

```

### Request 3

A request is made to get the Tier, Rating, and BillingDay fields. The Rating field is not a valid field in a subscriber profile. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="Rating"/></expr>
    <expr><attr name="BillingDay"/></expr>
  </select>
  <where>
    <expr><attr name="NAI"/><op value="=" />
      <value val="john.smith@operator.com"/></expr>
  </where>
</req>

```

### Response 3

The request fails. The error value indicates that the Rating field is undefined, and the affected rows are 0. The original request is not included.

```

<req name="select" resonly="y">
  <res error="70015" affected="0"/>
</req>

```

### Request 4

A request is made to get the MSISDN and BillingDay fields, as well as the Quota and State entity data. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="MSISDN"/></expr>
    <expr><attr name="BillingDay"/></expr>
    <expr><attr name="Quota"/></expr>
    <expr><attr name="State"/></expr>
  </select>
  <where>

```

```

    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

#### Response 4

The request is successful, and the 4 requested values are returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>33123654862</rv>
      <rv>23</rv>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="AggregateLimit">
              <cid>9223372036854775807</cid>
              <time>3422</time>
              <totalVolume>1000</totalVolume>
              <inputVolume>980</inputVolume>
              <outputVolume>20</outputVolume>
              <serviceSpecific>12</serviceSpecific>
              <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
            </quota>
          </usage>]]>
      </rv>
      <rv>
        <![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>]]>
      </rv>
    </row>
  </rset>
</req>

```

#### Request 5

A request is made to get the MSISDN field, as well as the DynamicQuota and State entity data. The subscriber does not have any DynamicQuota data. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="MSISDN"/></expr>
    <expr><attr name="DynamicQuota"/></expr>
    <expr><attr name="State"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

**Response 5**

The request is successful, and the 3 requested values are returned. The DynamicQuota is indicated as being not set. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>33123654862</rv>
      <rv null="y"/>
      <rv>
        <![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>]]>
        </rv>
      </row>
    </rset>
  </req>
```

**Request 6**

A request is made to get fields IMSI,GL,WL,BL,SV for the specified IMEI key. The request is not required in the response.

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

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="IMSI"/></expr>
    <expr><attr name="GL"/></expr>
    <expr><attr name="BL"/></expr>
    <expr><attr name="WL"/></expr>
    <expr><attr name="SV"/></expr>
  </select>
  <where>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="98765439876545"/></expr>
  </where>
</req>
```

**Response 6**

The request is successful, and the 5 requested values are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>987654398765450</rv>
      <rv>0</rv>
      <rv>0</rv>
      <rv>0</rv>
      <rv>00</rv>
    </row>
  </rset>
</req>
```

**Request 7**

A request is made to get fields GL,WL,BL for the specified IMEI key in the IMEI range entry. The request is not required in the response.

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

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="GL"/></expr>
    <expr><attr name="BL"/></expr>
    <expr><attr name="WL"/></expr>
  </select>
  <where>
    <expr>
      <attr name="IMEI"/>
      <op value="=">
      <value val="44111111111122"/>
    </expr>
  </where>
</req>
```

**Response 7**

The request is successful since the IMEI is in the range, and the 5 requested values are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>44111111111122</rv>
      <rv>0</rv>
      <rv>0</rv>
      <rv>0</rv>
    </row>
  </rset>
</req>
```

**Request 8**

A request is made to get fields GL,WL,BL for the specified IMEI key range. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="GL"/></expr>
    <expr><attr name="BL"/></expr>
    <expr><attr name="WL"/></expr>
  </select>
  <where>
    <expr>
      <attr name="IMEI"/>
      <op value="=">
      <value val="44111111111111"/>
    </expr>
    <expr>
      <attr name="IMEI"/>
      <op value="=">
      <value val="44441111111111"/>
    </expr>
  </where>
</req>
```

**Response 8**

The request is successful since the IMEI is in the range, and the 5 requested values are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>44111111111122</rv>
      <rv>0</rv>
      <rv>0</rv>
      <rv>0</rv>
    </row>
  </rset>
</req>
```

**6.2.3 Update Field****Description**

This operation updates a fields to the specified values for the subscriber that is identified by the keys specified in *keyNameX* and *keyValueX*. This operation replaces (sets) the values of the fields, which means that any existing values for the fields are deleted first.

For multi-value fields, all existing values are removed and only the new values specified are inserted. Adding values to a current set is accomplished using Add Field Value. 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 d (it is not a,b,c,d).

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

**NOTES**

- If the requested fields are valid, but not present, they are created.
- An entire entity for the subscriber can be replaced by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the entire XML blob value in *cdataFieldValue*.
- 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 a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

**Prerequisites**

- A subscriber with the Keys of the *keyNameX/keyValueX* values supplied must exist.
- Each requested field *fieldName* must be a valid field in the subscriber profile.
- Each requested *cdataFieldName* must be a valid non pooled transparent/opaque interface entity name for a subscriber.

**Request**

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <expr>
<
  <attr name="fieldName1"/><value val="fieldValue1"/>
|
  <attr name="cdataFieldName1"/><op value="="/>
  <cdata><![CDATA[cdataFieldValue1]]></cdata>
>
```



```

    </expr>
  [
    <expr>
  <
    <attr name="fieldName2"/><value val="fieldValue2"/>
  |
    <attr name="cdataFieldName2"/><op value="="/>
    <cdata><![CDATA[cdataFieldValue2]]></cdata>
  >
    </expr>
  :
  <expr>
  <
    <attr name="fieldNameN"/><value val="fieldValueN"/>
  |
    <attr name="cdataFieldNameN"/><op value="="/>
    <cdata><![CDATA[cdataFieldValueN]]></cdata>
  >
    </expr>
  ]
</set>
<where>
  <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
  <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
  :
  <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
</where>
</req>

```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

**NOTE:** For multi-value fields, the value can contain a comma separated list of values on a single line. For example, a,b,c

- *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*
- *cdataFieldNameX*: A user defined field in the subscriber profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC

Value is either Quota, State, or DynamicQuota

- *cdataFieldValueX*: Contents of the XML data blob for *cdataFieldNameX*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

## Response

```

<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest

```

```
]
<res error="error" affected="affected"/>
</req>
```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request  
**Values:**A string with 1 to 4096 characters
- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of subscribers updated. A value of 1 is expected for success

**Error Codes 6: Update Field**

| Error Code              | Description   |
|-------------------------|---|
| FIELD_VAL_INVALID       | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION    | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE     | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found  |
| INVAL_REPEATABLE_ELEM   | Invalid Repeatable Element  |
| INVALID_XML             | Invalid Input XML   |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

**Update Field Examples**

**Request 1**

A request is made to update the value of the BillingDay field to 23, and the Tier field to Gold. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><value val="23"/></expr>
    <expr><attr name="Tier"/><value val="Gold"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="305801234567890"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the BillingDay value was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to update the value of the BillingDay field to 55. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><value val="55"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="305801234567890"/></expr>
  </where>
</req>
```

**Response 2**

The request fails. The error value indicates the value of BillingDay was invalid, and the affected rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70006" affected="0"/>
</req>
```

**Request 3**

A request is made to update the value of the BillingDay field to 23, and the entire State entity. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><value val="23"/></expr>
    <expr><attr name="State"/><op value="="/>
      <CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <state>
          <version>1</version>
          <property>
            <name>mcc</name>
            <value>302</value>
          </property>
          <property>
            <name>expire</name>
            <value>2014-02-09T11:20:32</value>
          </property>
        </state>]]>
      </CDATA>
    </expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="305801234567890"/></expr>
  </where>
</req>
```

**Response 3**

The request is successful, and the BillingDay and State values were updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 4**

A request is made to update the value of the Entitlement field using a single field with multiple values. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Entitlement"/><value val="Weekend,Evening"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="305801234567890"/></expr>
  </where>
</req>
```

**Response 4**

The request is successful, and the Entitlement value was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 5**

A request is made to update the value of the Entitlement field using multiple fields each containing a single value. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Entitlement"/><value val="Weekend"/></expr>
    <expr><attr name="Entitlement"/><value val="Evening"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="305801234567890"/></expr>
  </where>
</req>
```

**Response 5**

The request is successful, and the Entitlement value was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 6**

A request is made to update the value of the MSISDN field to 14161234567. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="MSISDN"/>
      <value val="14161234567"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
  </where>
</req>
```

**Response 6**

The request is successful, and the MSISDN value was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 7**

A request is made to update the value of the MSISDN field to 14161234567. The subscriber has 3 existing MSISDN values of 15145551234, 14161234567, and 19191112222. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="MSISDN"/>
      <value val="14161234567"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
  </where>
</req>
```

**Response 7**

The request is successful, and the MSISDN value was updated, and has a single value of 14161234567. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 8**

A request is made to update the value of the subscribers NAI to two values of mum@foo.com and cust514@op.com. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="NAI"/>
      <value val="mum@foo.com,cust514@op.com"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
  </where>
</req>
```

**Response 8**

The request is successful, the NAI field was updated. The subscriber has 2 NAIs. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 9**

A request is made to update the value of the BL field to 1. The request is not required in the response.

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

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
```

```

<set>
  <expr><attr name="BL"/><value val="1"/></expr>
</set>
<where>
  <expr><attr name="IMEI"/><op value="="/>
    <value val="98765439876545"/></expr>
</where>
</req>

```

**Response 9**

The request is successful, and the BL value was updated. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 10**

A request is made to update the value of the WL field to 0. The request is not required in the response.

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

```

<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="WL"/><value val="0"/></expr>
  </set>
  <where>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="441111111111111"/></expr>
    <expr><attr name="IMEI"/><op value="="/>
      <value val="444411111111111"/></expr>
  </where>
</req>

```

**Response 10**

The request is successful, and the WL value was updated. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**6.2.4 Delete Field****Description**

This operation deletes the specified fields for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

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

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

**Prerequisites**

- A subscriber with the keys of the keyNameX and keyValueX values supplied must exist.
- Each requested field fieldNameX must be a valid field in the subscriber profile.

**Request**

```

<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="fieldName1"/><op value="="/>

```

```

        <value val="" isnull="y"/></expr>
    [
        <expr><attr name="fieldName2"/><op value="="/>
        <value val="" isnull="y"/></expr>
        :
        <expr><attr name="fieldNameN"/><op value="="/>
        <value val="" isnull="y"/></expr>
    ]
</set>
<where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
    [
        <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
        :
        <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
    ]
</where>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- **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

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

**Response**

```

<req name="update" [resonly="resonly"] [id="id"]>
    [
        originalXMLRequest
    ]
    <res error="error" affected="affected"/>
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values:A string with 1 to 4096 characters

- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of subscribers updated. A value of 1 is expected for success

**Error Codes 7: Delete Field**

| Error Code      | Description   |
|-----------------|---|
| FIELD_UNDEFINED | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |

| Error Code              | Description  |
|-------------------------|--|
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found |
| ONE_KEY_REQUIRED        | At least one key is required for a subscriber                          |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber             |

### Delete Field Examples

#### Request 1

A request is made to delete the BillingDay and Tier fields. Both fields are valid subscriber profile fields. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="BillingDay"/><op value="="/>
      <value val="" isnull="y"/></expr>
    <expr><attr name="Tier"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

#### Response 1

The request is successful, and the two fields were deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### Request 2

A request is made to delete the Message field. Message is not a valid subscriber profile fields. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Message"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

#### Response 2

The request fails. The error value indicates the given field was not found, and the affected rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70015" affected="0"/>
</req>
```



**Request 3**

A request is made to delete the MSISDN field. The subscriber has an IMSI and an MSISDN key field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
  </where>
</req>
```

**Response 3**

The request is successful, and the MSISDN field was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 4**

A request is made to delete the NAI field. The subscriber only has an NAI key field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="NAI"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="NAI"/><op value="="/>
      <value val="mum@foo.com"/></expr>
  </where>
</req>
```

**Response 4**

The request fails. The error value indicates the only key cannot be deleted, and the affected rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70044" affected="0"/>
</req>
```

**Request 5**

A request is made to delete the IMSI field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="IMEI"/><op value="="/><value val=" 98765439876545"/></expr>
  </where>
</req>
```

**Response 5**

The request is successful, and the IMSI field was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**6.2.5 Delete Field Value****Description**

This operation deletes one or more values from the specified field for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

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

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

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

**NOTES**

- 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.
- A request to delete fields values can also be mixed with a request to update or delete a fields. But, the same field for which a RemoveFromSet operation is being performed cannot also be updated or deleted, else the request fails.
- A request to delete fields values using the RemoveFromSet operation can also contain a AddToSet operation to add fields values. If both operations are included in the same request, the AddToSet is performed before the RemoveFromSet irrespective of the order in which they are supplied.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied

**Prerequisites**

- A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.
- The field fieldName must be a valid field in the subscriber profile, and must be a multi-value field.
- Each fieldValueX being removed must be present in the field.

**Request**

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="fieldName1"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    [
      <expr><attr name="fieldName2"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
      :
      <expr><attr name="fieldNameX"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    ]
    </oper>
  </set>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
```

```
[
  <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
  :
  <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
</where>
</req>
```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- **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 *fieldName*

**NOTES**

- One or more fieldValueX values for a fieldNameX can be supplied. To remove more than one value, either supply a comma separated list of values, or include multiple <expr> elements for the field.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values:A string with 1 to 4096 characters

- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of subscribers updated. A value of 1 is expected for success

**Error Codes 8: Delete Field Value**

| Error Code      | Description   |
|-----------------|---|
| FIELD_UNDEFINED | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND   | Key Not Found. A subscriber or pool with the given key cannot be found                      |

| Error Code              | Description   |
|-------------------------|---|
| FLD_NOT_MULTI           | Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value |
| ONE_KEY_REQUIRED        | At least one key is required for a subscriber   |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

### Delete Field Value Examples

#### Request 1

A request is made to remove the value DayPass from the Entitlement field. The Entitlement field is a valid multi-value field. The DayPass value is present in the Entitlement field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

#### Response 1

The request is successful, and the value was removed from the Entitlement field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### Request 2

A request is made to remove the values WeekendPass and Unlimited from the Entitlement field. The Entitlement field is a valid multi-value field. The WeekendPass value is present in the Entitlement field, but the Unlimited value is not. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="Entitlement"/><value val="WeekendPass"/></expr>
      <expr><attr name="Entitlement"/><value val="Unlimited"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

#### Response 2

The request is successful, and the WeekendPass value was removed from the Entitlement field. The original request is not included.

```
<req name="update" resonly="y">
```

```
<res error="0" affected="1"/>
</req>
```

**Request 3**

A request is made to remove the key value 14161234567 from the MSISDN field. The subscriber has 3 MSISDN values, 14161234567, 151454551234 and 15141234567. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber">
    <set>
      <oper name="RemoveFromSet">
        <expr><attr name="MSISDN"/><value val="14161234567"/></expr>
      </oper>
    </set>
    <where>
      <expr><attr name="MSISDN"/><op value="="/>
        <value val="15145551234"/></expr>
    </where>
  </req>
```

**Response 3**

The request is successful, and the value was removed from the MSISDN field. The subscriber has 2 MSISDN values, 151454551234 and 15141234567. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 4**

A request is made to remove the value of the IMSI field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber">
    <set>
      <oper name="RemoveFromSet">
        <expr><attr name="IMSI"/><value val="987654398765451"/></expr>
      </oper>
    </set>
    <where>
      <expr><attr name="IMEI"/><op value="="/>
        <value val="98765439876545"/></expr>
    </where>
  </req>
```

**Response 4**

The request is successful, and the value was removed from the IMSI field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**6.3 Subscriber Opaque Data Commands**

**Table 15: Summary of Subscriber Opaque Data Commands**

| Command            | Description                              | Keys                 | Command Syntax   |
|--------------------|--|----------------------|--|
| Create Opaque Data | Create opaque data of the specified type | MSISDN, IMSI, NAI or | <req name="insert">           <ent name="Subscriber"/> |

| Command            | Description                                | Keys      | Command Syntax   |
|--------------------|--|-----------|--|
| Get Opaque Data    | Retrieve opaque data of the specified type | AccountId | <req name="select"><br><ent name="Subscriber"/>                                      |
| Update Opaque Data | Update opaque data of the specified type   |           | <req name="update"><br><ent name="Subscriber"/>                                      |
| Delete Opaque Data | Delete opaque data of the specified type   |           | <req name="update"><br><ent name="Subscriber"/><br>... <value val="" isnull="y"/>... |

### 6.3.1 Create Opaque Data

#### Description

This operation creates the opaque data of the specified opaqueDataType for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

The opaque data is provided in the request in a <cdata> construct.

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

#### Prerequisites

- A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.
- The opaqueDataType must reference a valid Entity in the Interface Entity Map table in the SEC.
- Opaque data of the opaqueDataType does not exist for the subscriber (unless the odk attribute is specified).

#### Request

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="opaqueDataType"/><op value=""/><cdata>
<![CDATA[
CDATAFieldValue
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="keyName1"/><op value=""/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value=""/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value=""/><value val="keyValueN"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

#### Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *odk*: (Optional) Indicates that the insert request is converted to an update if the opaque data type specified exists
- *opaqueDataType*: A user defined type/name for the opaque data  
Value is either Quota, State, or DynamicQuota
- *cdataFieldValue*: Contents of the XML data blob
- *keyNameX*: A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId
- *keyValueX*: Corresponding key field value assigned to *keyNameX*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

**Response**

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values:A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers inserted/updated. A value of 1 is expected for success

**Error Codes 9: Create Opaque Data**

| Error Code            | Description   |
|-----------------------|---|
| INTF_ENTY_NOT_FOUND   | Interface Entity Not Found  |
| MULT_VER_TAGS_FOUND   | Multiple Version Tags Found   |
| FIELD_VAL_INVALID     | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION  | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| INVAL_REPEATABLE_ELEM | Invalid Repeatable Element  |
| INVALID_XML           | Invalid Input XML   |
| FIELD_UNDEFINED       | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_NOT_FOUND         | Key Not Found. A subscriber or pool with the given key cannot be found  |

| Error Code              | Description  |
|-------------------------|--|
| REG_EXISTS              | Register exists  |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber |

### Create Opaque Data Examples

#### Request 1

A request is made to create the Quota opaque data. The Quota XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Quota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]></cdata></expr>
</set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

#### Response 1

The request is successful, and the Quota opaque data was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### Request 2

A request is made to create the State opaque data. The State XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="State"/><op value="="/><cdata>
<![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>
]]></cdata></expr>
</set>
<where>
  <expr><attr name="MSISDN"/><op value="="/>
    <value val="33123654862"/></expr>
</where>
</req>

```

**Response 2**

The request is successful, and the State opaque data was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 3**

A request is made to create the DynamicQuota opaque data. The Quota XML blob is supplied whole. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="DynamicQuota"/><op value="="/><cdata>
<![CDATA[<?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>
    <activationdatettime>2015-03-09T11:20:32</activationdatettime>
    <expirationdatettime>2015-04-09T11:20:32</expirationdatettime>
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
</definition>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

**Response 3**

The request is successful, and the DynamicQuota opaque data was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 4**

A request is made to create the Location opaque data. The Location XML blob is supplied whole. Location is not a valid opaque data type. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="Subscriber"/>

```

```

<set>
  <expr><attr name="Location"/><op value=""/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<location>
  <town>Montreal</town>
  <province>Quebec</province>
  <country>Canada</country>
</location>
]]></cdata></expr>
</set>
<where>
  <expr><attr name="MSISDN"/><op value=""/>
    <value val="33123654862"/></expr>
</where>
</req>

```

#### Response 4

The request fails. The error value indicates the opaque data type is invalid, and the affected rows are 0. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="70015" affected="0"/>
</req>

```

### 6.3.2 Get Opaque Data

#### Description

This operation retrieves the opaque data of the specified opaqueDataType for the subscriber that is identified by the Keys in keyNameX and keyValueX.

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

#### Prerequisites

A subscriber with the Keys of the keyNameX/keyValueX values 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

```

<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="opaqueDataType"/></expr>
  </select>
  <where>
    <expr><attr name="keyName1"/><op value=""/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value=""/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value=""/><value val="keyValueN"/></expr>
  ]
  </where>
</req>

```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *opaqueDataType*: A user defined type/name for the opaque data  
Value is either Quota, State, or DynamicQuota
- *keyNameX*: A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId
- *keyValueX*: Corresponding key field value assigned to *keyNameX*

**Response**

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>
        <![CDATA[CDATA[CDATA[CDATA[cdataRowValue]]]>
      </rv>
    </row>
  </rset>
]
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values:A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers returned. A value of 1 is expected for success
- *cdataRowValue*: Contents of the XML data blob

**NOTES**

- The *<rset>* (row set) element is optional. It is only present if the request was successful. Only a single *<row>* element is returned, with a single *<rv>* (row value) element containing an XML CDATA construct containing the requested opaque data (XML blob).
- Multiple subscriber key values can be supplied. See section 2.11 for details.

**Error Codes 10: Get Opaque Data**

| Error Code              | Description   |
|-------------------------|---|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found  |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found                      |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber                                  |

## Get Opaque Data Examples

### Request 1

A request is made to get the Quota opaque data. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="Quota"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

### Response 1

The request is successful, and the Quota opaque data is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="AggregateLimit">
              <cid>9223372036854775807</cid>
              <time>3422</time>
              <totalVolume>1000</totalVolume>
              <inputVolume>980</inputVolume>
              <outputVolume>20</outputVolume>
              <serviceSpecific>12</serviceSpecific>
              <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
            </quota>
          </usage>]]>
      </rv>
    </row>
  </rset>
</req>

```

### Request 2

A request is made to get the State opaque data. State is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="State"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

### Response 2

The request is successful, and indicates that the requested opaque data type does not exist. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
  </rset>
</req>

```

```

    </row>
  </rset>
</req>

```

**Request 3**

A request is made to get the Location opaque data. Location is not a valid opaque data type. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="Location"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

**Response 3**

The request fails. The error value indicates the opaque data type is invalid, and the affected rows are 0. The original request is not included.

```

<req name="select" resonly="y">
  <res error="70015" affected="0"/>
</req>

```

**Request 4**

A request is made to get the State and Quota Entities. State is a valid opaque data type, but the subscriber does not have this opaque data type. The subscriber does have Quota opaque data though. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Subscriber"/>
  <select>
    <expr><attr name="Quota"/></expr>
    <expr><attr name="State"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

**Response 4**

The request is successful, and indicates that Quota exists and State does not exist. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="AggregateLimit">
              <cid>9223372036854775807</cid>
              <time>3422</time>
              <totalVolume>1000</totalVolume>
              <inputVolume>980</inputVolume>
              <outputVolume>20</outputVolume>
              <serviceSpecific>12</serviceSpecific>
            </quota>
          </usage>
        </CDATA!>
      </rv>
    </row>
  </rset>
</req>

```

```

        <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
    </quota>
</usage]]>
</rv>
<rv null="y"/>
</row>
</rset>
</req>

```

### 6.3.3 Update Opaque Data

#### Description

This operation updates the opaque data of the specified opaqueDataType for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

The opaque data is provided in the request in a <CDATA> construct. The existing opaque data is completely replaced by the data supplied in the request.

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

#### Prerequisites

A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.

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

#### Request

```

<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="opaqueDataType"/><op value=""/><CDATA>
<![CDATA[
CDATAFieldValue
]]></CDATA></expr>
  </set>
  <where>
    <expr><attr name="keyName1"/><op value=""/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value=""/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value=""/><value val="keyValueN"/></expr>
  ]
  </where>
</req>

```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Value is either Quota, State, or DynamicQuota

- *CDATAFieldValue*: Contents of the XML data blob
- *keyNameX*: A key field in the subscriber profile

Value is either IMSI, MSISDN, NAI, or AccountId

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

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers updated. A value of 1 is expected for success

**Error Codes 11: Update Opaque Data**

| Error Code              | Description   |
|-------------------------|---|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found  |
| FIELD_VAL_INVALID       | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION    | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| INVAL_REPEATABLE_ELEM   | Invalid Repeatable Element  |
| INVALID_XML             | Invalid Input XML   |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

**Update Opaque Data Examples**

**Request 1**

A request is made to update the State opaque data. The State XML blob is supplied whole. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="State"/><op value="="/><cdata>
<![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>
]]></CDATA></expr>
</set>
<where>
  <expr><attr name="MSISDN"/><op value="="/>
    <value val="33123654862"/></expr>
</where>
</req>

```

**Response 1**

The request is successful, and the State opaque data was updated. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 2**

A request is made to update the Quota opaque data. Quota is a valid opaque data type, but the subscriber does not have this opaque data type. The Quota XML blob is supplied whole. The request is not required in the response.

```

<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Quota"/><op value="="/><CDATA>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]></CDATA></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

**Response 2**

The request is successful, and the Quota opaque data was created. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="0"/>
</req>

```



### 6.3.4 Delete Opaque Data

#### Description

This operation deletes the opaque data of the specified opaqueDataType for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

Only one opaque data type can be deleted per request.

**NOTE:** The deletion of a non-existent opaque data type (but that is defined in the SEC) is not considered as an error.

#### Prerequisites

- A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.
- The opaqueDataType must reference a valid Entity in the Interface Entity Map table in the SEC.

#### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="opaqueDataType"/><op value=""/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="keyName1"/><op value=""/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value=""/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value=""/><value val="keyValueN"/></expr>
  ]
  </where>
</req>
```

- *keyNameX*: A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId
- *keyValueX*: Corresponding key field value assigned to *keyNameX*
- *opaqueDataType*: A user defined type/name for the opaque data  
Value is either Quota, State, or DynamicQuota

**NOTE:** The data is deleted by setting an empty field value, and also specifying the attribute `isnull="y"`

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

#### Response

```
<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers deleted. A value of 1 is expected for success

**Error Codes 12: Delete Opaque Data**

| Error Code              | Description  |
|-------------------------|--|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found   |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber             |

**Delete Opaque Data Examples****Request 1**

A request is made to delete the State opaque data. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="State"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the State opaque data was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to delete the Quota opaque data. Quota is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Subscriber"/>
  <set>
    <expr><attr name="Quota"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, because an error is not returned if the subscriber does not have the opaque data type.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

## 6.4 Subscriber Data Row Commands

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

**NOTE:** Subscriber data row commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an `OPER_NOT_ALLOWED` error being returned.

**Table 16: Summary of Subscriber Data Row Commands**

| Command    | Description  | Keys   | Command Syntax  |
|------------|--|--|---|
| Create Row | Insert data row into transparent data of the specified type.   | (MSISDN, IMSI, NAI or AccountId) and Row Identifier<br>or<br>(MSISDN, IMSI, NAI or AccountId) and Row Identifier and Instance Identifier | <pre>&lt;req name="insert"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...</pre>   |
| Get Row    | Retrieve data row from transparent data of the specified type. |  | <pre>&lt;req name="select"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;       &lt;/expr&gt; ...   ]</pre> |
| Delete Row | Delete data row in transparent data of the specified type      |  | <pre>&lt;req name="delete"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;       &lt;/expr&gt; ...   ]</pre> |

### 6.4.1 Create Row

#### Description

This operation creates a data row for the subscriber that is identified by the Keys specified in `keyNameX` and `keyValueX`.

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

#### NOTES

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

## Prerequisites

A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.

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

## Request

**NOTE:** This command allows 2 different formats. One with the keyNameX/keyValueX values in the <set> element, and another with the keyNameX/keyValueX values in a <where> element.

### Format 1

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="keyName1"/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  <expr><attr name="rowIdName"/><value val="rowIdValue"/></expr>
  [
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
  </set>
</req>
```

### Format 2

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="rowIdName"/><value val="rowIdValue"/></expr>
  [
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
  </set>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *odk*: (Optional) Indicates that the insert request is converted to an update if the data row for the specified entity exists
- *entityName*: A user defined entity type/name for the transparent data
  - o Value is QuotaEntity for the Quota transparent data
  - o Value is DynamicQuotaEntity for the DynamicQuota transparent data
- *keyNameX*: A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId
- *keyValueX*: Corresponding key field value assigned to *keyNameX*
- *rowIdName*: Name of the XML attribute that identifies the row in the data blob
  - o Value is name for Quota transparent data
  - o Value is name for DynamicQuota transparent data
- *rowIdValue*: The row name value that identifies the row in the data blob
- *fieldNameX*: A user defined field in the data row
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

**NOTE:** For multi-value fields, the value can contain a comma separated list of values on a single line. For example, a,b,c

## NOTES

- Rows that have the same *rowIdName/rowIdValue* are permitted. Where duplicate rows occur, and an additional field is set to define uniqueness (such as *<cid>* in the Quota entity) validation is not performed by UDR to ensure uniqueness. Unique values must be supplied by the provisioning client otherwise operations (such as updating an existing row) may fail if more than one matching row is found.
- If the *odk="yes"* attribute is set (implying that an update be made if the row exists), then if multiple rows exist for the specified *rowIdName/rowIdValue*, then the request fails because it is not known which of the multiple rows to update.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

## Response

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers updated. A value of 1 is expected for success

**Error Codes 13 Create Row**

| Error Code              | Description  |
|-------------------------|--|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found   |
| FIELD_VAL_INVALID       | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC   |
| OCC_CONSTR_VIOLATION    | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field  |
| INVAL_REPEATABLE_ELEM   | Invalid Repeatable Element   |
| INVALID_SOAP_XML        | Invalid SOAP XML   |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC  |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found   |
| MULTIPLE_ROWS_FOUND     | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria<br><b>NOTE:</b> Only returned when the odk="yes" attribute is supplied, and duplicate candidate rows to update are found |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber   |

**Create Row Examples****Request 1**

A request is made to create a data row in the QuotaEntity (Quota) data. The data row identifier field is name, and the value is Q1. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="name"/><value val="Q1"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>
```

**Response 1**

The request is successful, and the data row was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to create a data row in the QuotaEntity (Quota) data, using the alternate request format. The data row identifier field is name, and the value is Q2. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="name"/><value val="Q2"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15141234567"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, and the data row was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 3**

A request is made to create a data row in the QuotaEntity (Quota) data. Quota is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="name"/><value val="Q1"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>
```

**Response 3**

The request is successful, and the data row as well as the Quota entity is created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 4**

A request is made to create a data row in the QuotaEntity (Quota) data. The data row identifier field is name, and the value is Q2. The odk attribute is included requesting the data row be updated if it exists. A single row with the name of Q2 exists in the Quota data. The request is not required in the response.

```

<req name="insert" resonly="y" odk="yes">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="name"/><value val="Q2"/></expr>
    <expr><attr name="cid"/><value val="922337203685488888"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>

```

#### Response 4

The request is successful, and the existing Q2 data row was updated. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

#### Request 5

A request is made to create a data row in the QuotaEntity (Quota) data. The data row identifier field is name, and the value is Q3. The odk attribute is included requesting the data row be updated if it exists. Two rows with the name of Q3 exist in the Quota data. The request is not required in the response.

```

<req name="insert" resonly="y" odk="yes">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="name"/><value val="Q3"/></expr>
    <expr><attr name="cid"/><value val="922337203685477777"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>

```

#### Response 5

The request fails. The error value indicates that multiple existing rows are found (more than one row has a name of Q3, and hence it is not possible to know which of the two rows to update), and the affected rows are 0. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="70035" affected="0"/>
</req>

```

#### Request 6

A request is made to create a data row in the QuotaEntity (Quota) data. The MSISDN, IMSI, and AccountId keys are supplied, which reference the same subscriber. The data row identifier field is name, and the value is Q1. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="IMSI"/><value val="302370123456789"/></expr>
    <expr><attr name="AccountId"/><value val="123456"/></expr>

```



```

    <expr><attr name="name"/><value val="Q1"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>

```

**Response 6**

The request is successful, and the data row was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 7**

A request is made to create a data row in the QuotaEntity (Quota) data, using the alternate request format. The MSISDN and NAI keys are supplied, which reference the same subscriber. The data row identifier field is name, and the value is Q2. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="name"/><value val="Q2"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15141234567"/></expr>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
    <expr><attr name="NAI"/><op value="="/>
      <value val="dad@foo.com"/></expr>
  </where>
</req>

```

**Response 7**

The request is successful, and the data row was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 8**

A request is made to create a data row in the DynamicQuotaEntity (DynamicQuota) data. The MSISDN key is supplied. The data row identifier field is name, and the value is DQ1. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="name"/><value val="DQ1"/></expr>
    <expr><attr name="Type"/><value val="top-up"/></expr>
    <expr><attr name="InstanceId"/><value val="15678"/></expr>
  </set>
</req>

```

```

    <expr><attr name="Priority"/><value val="4"/></expr>
    <expr><attr name="InitialTime"/><value val="135"/></expr>
    <expr><attr name="InitialTotalVolume"/><value val="2000"/></expr>
    <expr><attr name="InitialInputVolume"/><value val="1500"/></expr>
    <expr><attr name="InitialOutputVolume"/><value val="500"/></expr>
    <expr><attr name="InitialServiceSpecific"/><value val="4"/></expr>
    <expr><attr name="activationdatetime"/><value val="2015-05-22T00:00:00-05:00"/></expr>
    <expr><attr name="expirationdatetime"/><value val="2015-05-29T00:00:00-05:00"/></expr>
    <expr><attr name="InterimReportingInterval"/><value val="100"/></expr>
    <expr><attr name="Duration"/><value val="10"/></expr>
  </set>
</req>

```

### Response 8

The request is successful, and the data row was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

## 6.4.2 Get Row

### Description

This operation retrieves a data rows for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

The data row identifier field is specified in rowIdName, and the row identifier value is specified in rowIdValue. An additional field can be specified to indicate a unique row in instanceFieldName/instanceFieldValue.

All data rows that match the requested rowIdName/rowIdValue and instanceFieldName/instanceFieldValue (if specified) are returned.

### NOTES

- The *rowIdValue* is case-sensitive. If a row existed called DayPass, then an attempt to retrieve a row called DayPass is successful, but an attempt to retrieve a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to retrieve a row with a field with the value Data is successful, but an attempt to retrieve a row with a field with the value DATA fails.

### Prerequisites

- A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.
- The entityName must reference a valid transparent Entity in the Interface Entity Map table in the SEC.
- The transparent entity must exist for the subscriber.

### Request

```

<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  <expr><attr name="rowIdName"/><op value="="/>
    <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
</where>

```

```
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
  - o *n*—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response  
Values: 1 to 4294967295
  - *entityName*: A user defined entity type/name for the transparent data
    - o Value is *QuotaEntity* for the Quota transparent data
    - o Value is *DynamicQuotaEntity* for the DynamicQuota transparent data
  - *keyNameX*: A key field in the subscriber profile  
Value is either *IMSI*, *MSISDN*, *NAI*, or *AccountId*
  - *keyValueX*: Corresponding key field value assigned to *keyNameX*
  - *rowIdName*: Name of the XML attribute that identifies the row in the data blob
    - o Value is name for Quota transparent data
    - o Value is name for 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
    - o Value is *cid* or *Type* for the Quota transparent data
    - o Value is *InstanceId* or *Type* for the DynamicQuota transparent data
  - *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

## Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
  <
    <rv>
      <![CDATA[cdataRowValue1]]>
    </rv>
  |
    <rv null="y"/>
  >
  </row>
[
  <row>
    <rv>
      <![CDATA[cdataRowValue2]]>
    </rv>
  </row>
  :
  <row>
    <rv>
      <![CDATA[cdataRowValueN]]>
    </rv>
  </row>
]
]
```

```

</rset>
]
</req>

```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers returned. A value of 1 or more is expected for success, whether or not a row was found
- *cdataRowValueN*: Contents of the XML data blob containing one requested/matching data row

## NOTES

- The `<rset>` (row set) element is optional. It is only present if the request was successful. One `<row>` element is returned per matching row, with a single `<rv>` (row value) element containing an XML CDATA construct containing a single requested data row instance.
- If the transparent entity exists, but the row value was not found, then the `<rv>` (row value) indicates the row does not exist by containing the value `<rv null="y"/>`.
- Multiple subscriber key values can be supplied. See section 2.11 for details.

## Error Codes 14: Get Row

| Error Code              | Description  |
|-------------------------|--|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found   |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found |
| REG_DATA_NOT_FOUND      | Register Data Not Found  |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber             |

## Get Row Examples

### Request 1

A request is made to get the Q1 data row from the Quota data. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>

```

### Response 1

The request is successful, and the Quota data is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>

```

```

<![CDATA[<?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>]]>
</rv>
</row>
</rset>
</req>

```

## Request 2

A request is made to get the Weekend data row from the Quota data. The Quota data contains two rows called Weekend. One with <cid> of 11223344, the other with a <cid> of 99887766. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>

```

## Response 2

The request is successful, and 2 Quota data rows are returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="Weekend">
              <cid>11223344</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>]]>
        </rv>
      </row>
      <row>
        <rv>
          <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
            <usage>
              <version>3</version>
              <quota name="Weekend">
                <cid>99887766</cid>
                <time>1232</time>
                <totalVolume>2000</totalVolume>
                <inputVolume>440</inputVolume>
                <outputVolume>8220</outputVolume>
                <serviceSpecific>99</serviceSpecific>
                <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
              </quota>
            </usage>
          </CDATA>
        </rv>
      </row>
    </rset>
  </req>

```

```

    </usage>]]>
  </rv>
</row>
</rset>
</req>

```

### Request 3

A request is made to get the Weekend data row from the Quota data, with the <cid> value of 11223344. The Quota data contains two rows called Weekend. One with <cid> of 11223344, the other with a <cid> of 99887766. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>

```

### Response 3

The request is successful, and the Quota data with a <cid> of 11223344 is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="Weekend">
              <cid>11223344</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>]]>
        </rv>
      </row>
    </rset>
  </req>

```

### Request 4

A request is made to get the LateNight data row from the Quota data, with the <cid> value of 11223344. The Quota data contains four rows called LateNight. Two with <cid> of 11223344, one with a <cid> of 99887766, and one with a <cid> of 55556666. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="LateNight"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>

```

**Response 4**

The request is successful, and the 2 Quota data rows with a <cid> of 11223344 are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="LateNight">
              <cid>11223344</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>]]>
        </rv>
      </row>
      <row>
        <rv>
          <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
            <usage>
              <version>3</version>
              <quota name="LateNight">
                <cid>11223344</cid>
                <time>1232</time>
                <totalVolume>2000</totalVolume>
                <inputVolume>440</inputVolume>
                <outputVolume>8220</outputVolume>
                <serviceSpecific>99</serviceSpecific>
                <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
              </quota>
            </usage>]]>
          </rv>
        </row>
      </rset>
    </req>
```

**Request 5**

A request is made to get the Weekday data row in the Quota data. The Weekday data row does not exist in the Quota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>
```

**Response 5**

The request is successful, and indicates that the requested row does not exist. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
```

```
</rset>
</req>
```

### Request 6

A request is made to get the Weekday data row in the Quota data. Quota is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>
```

### Response 6

The request fails. The error value indicates the opaque data type is not found, and the affected rows are 0. The original request is not included.

```
<req name="select" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

### Request 7

A request is made to get the DQ1 data row from the DynamicQuota data, with the 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.

```
<req name="select" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="DQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

### Response 7

The request is successful and the 2 DynamicQuota data rows with an InstanceId of 11223344 are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <definition>
            <version>1</version>
            <DynamicQuota name="DQ1">
              <Type>pass</Type>
              <InstanceId>11223344</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>]]>
</rv>
</row>
<row>
<rv>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<definition>
<version>1</version>
<DynamicQuota name="DQ1">
<Type>top-up</Type>
<InstanceId>11223344</InstanceId>
<Priority>3</Priority>
<InitialTime>125</InitialTime>
<InitialTotalVolume>1000</InitialTotalVolume>
<InitialInputVolume>500</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>]]>
</rv>
</row>
</rset>
</req>

```

### 6.4.3 Delete Row

#### Description

This operation deletes a data row for the subscriber that is identified by the Keys specified in `keyNameX` and `keyValueX`.

The data row identifier field is specified in `rowIdName`, and the row identifier value is specified in `rowIdValue`. An additional field can be specified to indicate a unique row in `instanceFieldName/instanceFieldValue`.

If more than one row matches the requested `rowIdName/rowIdValue` and `instanceFieldName/instanceFieldValue` (if specified), then all matching rows are deleted.

#### NOTES

- The *rowIdValue* is case-sensitive. If a row existed called DayPass, then an attempt to delete a row called DayPass is successful, but an attempt to delete a row called DAYPASS fails
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to delete a row with a field with the value Data is successful, but an attempt to delete a row with a field with the value DATA fails.
- The deletion of a non-existent data row is not considered an error.

#### Prerequisites

- A subscriber with the Keys of the `keyNameX/keyValueX` values supplied must exist.
- The `entityName` must reference a valid transparent Entity in the Interface Entity Map table in the SEC.
- The transparent entity must exist for the subscriber.

#### Request

```

<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
  ]
  :

```

```

    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  <expr><attr name="rowIdName"/><op value="="/>
    <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
  </where>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
  - o n—Includes the original request in the response (default)
- **id:** (Optional) Transaction ID value in the request, and passed back in the response  
Values: 1 to 4294967295
  - **entityName:** A user defined entity type/name for the transparent data
    - o Value is QuotaEntity for the Quota transparent data
    - o Value is DynamicQuotaEntity for the DynamicQuota transparent data
  - **keyNameX:** A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId
  - **keyValueX:** Corresponding key field value assigned to *keyNameX*
  - **rowIdName:** Name of the XML attribute that identifies the row in the data blob
    - o Value is name for Quota transparent data
    - o Value is name for 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
    - o Value is cid or type for the Quota transparent data
    - o Value is InstanceId or Type for the DynamicQuota transparent data
  - **instanceFieldValue:** Corresponding field value assigned to *instanceFieldName*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

## Response

```

<req name="delete" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request  
Values: A string with 1 to 4096 characters
- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values

- *affected*: A value of 1 indicates the rows existed and were deleted, or that the row did not exist

**Error Codes 15: Delete Row**

| Error Code              | Description  |
|-------------------------|--|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found   |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found |
| REG_DATA_NOT_FOUND      | Register Data Not Found  |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber             |

**Delete Row Examples****Request 1**

A request is made to delete the Q1 data row in the Quota data. The Q1 data row exists in the Quota data, and is there is only one row called Q1. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the data row in the Quota data was deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to delete the Weekend data row in the Quota data. The Weekend data row does not exist in the Quota data. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, because an error is not returned if the data row is not present. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 3**

A request is made to delete the Q3 data row in the Quota data. The Quota data contains two rows called Q3. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
  </where>
</req>
```

**Response 3**

The request is successful, and the data row in the Quota data was deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 4**

A request is made to delete the Q4 data row from the Quota data, with the <cid> value of 11223344. The Quota data contains two rows called Q4. One with <cid> of 11223344, the other with a <cid> of 99887766. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q4"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

**Response 4**

The request is successful, and the data row in the Quota data was deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 5**

A request is made to delete the Bonus data row in the Quota data. QuotaEntity is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="QuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Bonus"/></expr>
  </where>
</req>
```

**Response 5**

The request fails. The error value indicates the opaque data type is not found, and the affected rows are 0. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

**Request 6**

A request is made to delete the DQ1 data row from the DynamicQuota data, with the Type value of pass. The DynamicQuota data contains two rows called DQ1. One with Type of pass the other with a Type of top-up. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/>
      <value val="DQ1"/></expr>
    <expr><attr name="Type"/><op value="="/>
      <value val="pass"/></expr>
  </where>
</req>
```

**Response 6**

The request is successful, and the data row in the DynamicQuota data was deleted that matched the Type. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**6.5 Subscriber Data Row Field Commands**

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

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

**NOTE:** Subscriber data row field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an OPER\_NOT\_ALLOWED error being returned.

**Table 17: Summary of Subscriber Data Row Field Commands**

| Command          | Description                              | Keys   | Command Syntax  |
|------------------|--|--|---|
| Get Row<br>Field | Retrieve values for the specified fields | (MSISDN, IMSI, NAI or AccountId) and Row Identifier and Field name | <pre>&lt;req name="select"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;     &lt;/expr&gt; ...   ]</pre> |

| Command          | Description                                | Keys  | Command Syntax  |
|------------------|--|---|---|
| Update Row Field | Update fields to the specified values      | or<br>(MSISDN, IMSI, NAI or AccountId) and Row Identifier, Instance Identifier and Field name | <pre>&lt;req name="update"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;       &lt;/expr&gt; ...   ]</pre> |
| Delete Row Field | Delete all values for the specified fields |   | <pre>&lt;req name="update"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;       &lt;/expr&gt; ...   ]</pre> |

### 6.5.1 Get Row Field

#### Description

This operation retrieves a fields in a data row for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

The data row identifier field is specified in rowIdName, and the row identifier value is specified in rowIdValue. An additional field can be specified to indicate a unique row in instanceFieldName/instanceFieldValue. The field names are specified in fieldNameX.

All data rows that match the requested rowIdName/rowIdValue and instanceFieldName/instanceFieldValue (if specified) are returned.

#### NOTES

- If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and empty field data returns.
- The *rowIdValue* is case-sensitive. If a row existed 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.
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to delete a row with a field with the value Data is successful, but an attempt to delete a row with a field with the value DATA fails.

#### Prerequisites

- A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.
- The entityName must reference a valid transparent Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier/instance in the transparent data must exist for the subscriber.
- The field names specified must be valid fields for the Entity as defined in the SEC.

#### Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <select>
    <expr><attr name="fieldName1"/><value val="[fieldName1]"/></expr>
```

```

[
  <expr><attr name="fieldName2"/><value val="[fieldValue2]"/></expr>
  :
  <expr><attr name="fieldNameN"/><value val="[fieldValueN]"/></expr>
]
</select>
<where>
  <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
  <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
  :
  <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
  <expr><attr name="rowIdName"/><op value="="/>
    <value val="rowIdValue"/></expr>
[
  <expr><attr name="instanceFieldName"/><op value="="/>
    <value val="instanceFieldValue"/></expr>
]
</where>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

- o Value is QuotaEntity for the Quota transparent data
- o Value is DynamicQuotaEntity for the DynamicQuota transparent data

- **fieldNameX:** A user defined field in the data row

- **fieldValueX:** (Optional) Corresponding field value assigned to *fieldNameX*

**NOTE:** For multi-value fields, the value can contain a comma separated list of values

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

Value is either IMSI, MSISDN, NAI, or AccountId

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

- **rowIdName:** Name of the XML attribute that identifies the row in the data blob

- o Value is name for Quota transparent data
- o Value is name for 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

- o Value is cid or type for the Quota transparent data
- o Value is InstanceId or type for the DynamicQuota transparent data

- **instanceFieldValue:** Corresponding field value assigned to *instanceFieldName*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

## Response

```
<req name="select" [resonly="resonly"] [id="id"]>
```

```

[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
    <rv>rowValue1</rv> | <rv null="y"> | <rv></rv> >
    [
    <rv>rowValue2</rv> | <rv null="y"> | <rv></rv> >
    :
    <rv>rowValueN</rv> | <rv null="y"> | <rv></rv> >
    ]
    </row>
    [
    <row>
    <rv>rowValue1</rv> | <rv null="y"> | <rv></rv> >
    [
    <rv>rowValue2</rv> | <rv null="y"> | <rv></rv> >
    :
    <rv>rowValueN</rv> | <rv null="y"> | <rv></rv> >
    ]
    </row>
    :
    <row>
    <rv>rowValue1</rv> | <rv null="y"> | <rv></rv> >
    [
    <rv>rowValue2</rv> | <rv null="y"> | <rv></rv> >
    :
    <rv>rowValueN</rv> | <rv null="y"> | <rv></rv> >
    ]
    </row>
    ]
  </rset>
]
</req>

```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.
  - NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request
  - Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers returned. A value of 1 is expected if the specified row exists (whether or not the field was found). A value of 0 is expected if the row does not exist
- *rowValue*: The value of the requested field

**NOTE:** for multi-value fields, the value contains a comma separated list of values on a single line. For example, a,b,c

**NOTE:** The <rset> (row set) element is optional. It is only present if the request was successful. One <row> element is returned per matching row. One <rv> (row value) element exists for every fieldNameX supplied in the original request. The <rv> elements are ordered the same as the fieldNameX fields were specified in the original request. If the field is valid, but not present in the entity, this is indicated with <rv null="y">. If the field is present, but has an empty value, this is indicated with <rv></rv>.

**Error Codes 16: Get Row Field**

| Error Code          | Description                |
|---------------------|----------------------------|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found |



| Error Code              | Description   |
|-------------------------|---|
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found                      |
| REG_DATA_NOT_FOUND      | Register Data Not Found   |
| ROW_NOT_FOUND           | Data row specified is not found   |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber                                  |

### Get Row Field Examples

#### Request 1

A request is made to get the inputVolume field in the Q1 data row of the Quota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="inputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

#### Response 1

The request is successful, and the requested field value 980 is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>980</rv>
    </row>
  </rset>
</req>
```

#### Request 2

A request is made to get the outputVolume and cid fields in the Q2 data row of the Quota data. The Quota data contains two rows called Q2. One with <cid> of 11223344, the other with a <cid> of 99887766. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></expr>
    <expr><attr name="cid"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q2"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, and the requested field values are returned from each row. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>220</rv>
      <rv>11223344</rv>
    </row>
    <row>
      <rv>1050</rv>
      <rv>99887766</rv>
    </row>
  </rset>
</req>
```

**Request 3**

A request is made to get the outputVolume field in the Q3 data row of the Quota data, with the <cid> value of 11223344. The Quota data contains two rows called Q3. One with <cid> of 11223344, the other with a <cid> of 99887766. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

**Response 3**

The request is successful, and the requested field value 4000 is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>4000</rv>
    </row>
  </rset>
</req>
```

**Request 4**

A request is made to get the inputVolume, outputVolume, and totalVolume fields in the Q1 data row of the Quota data. The inputVolume field is present in the Q1 quota, the outputVolume field is present in the Q1 quota, but has an empty value, and the totalVolume field is not present in the Q1 quota (but is a valid field). The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="inputVolume"/></expr>
    <expr><attr name="outputVolume"/></expr>
    <expr><attr name="totalVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
```

```

    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>

```

#### **Response 4**

The request is successful, and the requested field values are returned. The inputVolume field is set to 980, the outputVolume field is indicated as being present, but empty, and the totalVolume field is indicated as not being present. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>980</rv>
      <rv></rv>
      <rv null="y"/>
    </row>
  </rset>
</req>

```

#### **Request 5**

A request is made to get the outputVolume field in the Q1 data row of the Quota data. The Q1 row exists, but the outputVolume field is not set. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>

```

#### **Response 5**

The request is successful, and the field is indicated to not be set. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y">
    </row>
  </rset>
</req>

```

#### **Request 6**

A request is made to get the outputVolume field in the Q2 data row of the Quota data. The Q2 row does not exist. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="QuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q2"/></expr>
  </where>
</req>

```

**Response 6**

The request fails. The error value indicates the row does not exist, and the affected rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70032" affected="0"/>
</req>
```

**Request 7**

A request is made to get the InstanceId, InitialTotalVolume and InitialInputVolume fields in the DQ3 data row of the Dynamic Quota data, with the InstanceId value of 15678. The Dynamic Quota data contains two rows called DQ3. One with InstanceId of 15570, the other with an InstanceId of 15678. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <select>
    <expr><attr name="InstanceId"/></expr>
    <expr><attr name="InitialTotalVolume"/></expr>
    <expr><attr name="InitialInputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
    <expr><attr name="name"/><op value="="/>
      <value val="DQ3"/></expr>
    <expr><attr name="InstanceId"/><op value="="/>
      <value val="15678"/></expr>
  </where>
</req>
```

**Response 7**

The request is successful, and the requested field values of InstanceId, InitialTotalVolume and InitialInputVolume fields are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>15678</rv>
      <rv>2000</rv>
      <rv>1500</rv>
    </row>
  </rset>
</req>
```

**6.5.2 Update Row Field****Description**

This operation updates a fields in a data row for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

The data row identifier field is specified in rowIdName and the row identifier value is specified in rowIdValue. An additional field can be specified to indicate a unique row in instanceFieldName/instanceFieldValue. The field names are specified in fieldNameX. If the specified fields are valid, but do not exist, they are created.

If more than one row matches the requested rowIdName/rowIdValue and instanceFieldName/instanceFieldValue (if specified), then the update request fails.

**NOTES**

- If the specified row does not exist, the request fails.

- If the requested fields are valid, but not present, they are created.
- The *rowIdValue* is case-sensitive. If a row existed 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.
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to delete a row with a field with the value Data is successful, but an attempt to delete a row with a field with the value DATA fails.
- Multiple subscriber key values can be supplied. See section 2.11 for details.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

### Prerequisites

- A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.
- The entityName must reference a valid transparent Entity in the Interface Entity Map table in the SEC.
- The data row with the given identifier/instance in the transparent data must exist for the subscriber.
- The field names specified must be valid fields for the Entity as defined in the SEC.

### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
  </set>
  <where>
    <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
    <expr><attr name="rowIdName"/><op value="="/>
      <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

#### Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *entityName*: A user defined entity type/name for the transparent data
  - o Value is QuotaEntity for the Quota transparent data
  - o Value is DynamicQuotaEntity for the DynamicQuota transparent data

- *fieldNameX*: A user defined field in the data row
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*
  - NOTE:** For multi-value fields, the value can contain a comma separated list of values on a single line. For example, a,b,c
- *keyNameX*: A key field in the subscriber profile
  - Value is either IMSI, MSISDN, NAI, or AccountId
- *keyValueX*: Corresponding key field value assigned to *keyNameX*
- *rowIdName*: Name of the XML attribute that identifies the row in the data blob
  - o Value is name for Quota transparent data
  - o Value is name for Dynamic Quota 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
  - o Value is cid or type for the Quota transparent data
  - o Value is InstanceId or type for the DynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.
  - NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request
  - Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers updated. A value of 1 is expected for success

**Error Codes 17: Update Row Field**

| Error Code           | Description   |
|----------------------|---|
| INTF_ENTY_NOT_FOUND  | Interface Entity Not Found  |
| OCC_CONSTR_VIOLATION | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_VAL_INVALID    | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| FIELD_UNDEFINED      | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE  | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |

| Error Code              | Description   |
|-------------------------|---|
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND      | Register Data Not Found   |
| ROW_NOT_FOUND           | Data row specified is not found   |
| MULTIPLE_ROWS_FOUND     | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

### Update Row Field Examples

#### Request 1

A request is made to update the inputVolume field in the Q1 data row of the Quota data. The Q1 data row exists in the Quota data, and is there is only one row called Q1. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

#### Response 1

The request is successful, and the field in the data row in the Quota data was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### Request 2

A request is made to update the cid field in the Q1 data row in the Quota data. The Q1 data row exists in the Quota data, and is there is only one row called Q1. The cid field is not allowed to be updated. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="cid"/><value val="11223344"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

#### Response 2

The request fails. The error value indicates the cid field cannot be updated, and the affected rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70016" affected="0"/>
</req>
```

**Request 3**

A request is made to update the outputVolume field in the Q6 data row of the Quota data. The Q6 data row exists in the Quota data, but there are two rows called Q6. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q6"/></expr>
  </where>
</req>
```

**Response 3**

The request fails because there was more than one row called Q6. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

**Request 4**

A request is made to update the InitialTotalVolume and InitialInputVolume fields in the DQ1 data row of the DynamicQuota data, with the InstanceId value of 15678. The DQ1 data row exists in the DynamicQuota data, but there are two rows called DQ1 one with InstanceId of 15570, the other with an InstanceId of 15678. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <set>
    <expr><attr name="InitialTotalVolume"/><value val="1000"/></expr>
    <expr><attr name="InitialInputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
    <expr><attr name="name"/><op value="="/>
      <value val="DQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/>
      <value val="15678"/></expr>
  </where>
</req>
```

**Response 4**

The request is successful, and the InitialTotalVolume and InitialInputVolume fields in the data row with InstanceId value of 15678 were updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```



### 6.5.3 Delete Row Field

#### Description

This operation deletes a fields in a data row for the subscriber that is identified by the Keys specified in `keyNameX` and `keyValueX`.

The data row identifier field is specified in `rowIdName`, and the row identifier value is specified in `rowIdValue`. An additional field can be specified to indicate a unique row in `instanceFieldName/instanceFieldValue`. The field names are specified in `fieldNameX`.

If more than one row matches the requested `rowIdName/rowIdValue` and `instanceFieldName/instanceFieldValue` (if specified), then the delete request fails.

#### NOTES

- If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and row/field data is not deleted.
- The `rowIdValue` is case-sensitive. If a row existed 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.
- The `instanceFieldValue` is case-sensitive. If a field contained the value Data, then an attempt to delete a field in a row with a field with the value Data is successful, but an attempt to delete a field in a row with a field with the value DATA fails.
- Multiple subscriber key values can be supplied. See section 2.11 for details.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied

#### Prerequisites

- A subscriber with the Keys of the `keyNameX/keyValueX` values supplied must exist.
- The `entityName` must reference a valid transparent Entity in the Interface Entity Map table in the SEC.
- At least one data row with the given identifier/instance in the transparent data must exist for the subscriber.
- The field names specified must be valid fields for the Entity as defined in the SEC.

#### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><op value="="/>
      <value val="" isnull="y"/></expr>
  [
    <expr><attr name="fieldName2"/><op value="="/>
      <value val="" isnull="y"/></expr>
    :
    <expr><attr name="fieldNameN"/><op value="="/>
      <value val="" isnull="y"/></expr>
  ]
</set>
<where>
  <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  <expr><attr name="rowIdName"/><op value="="/>
    <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
</where>
</req>
```

```
</where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *entityName*: A user defined entity type/name for the transparent data
  - o Value is *QuotaEntity* for the Quota transparent data
  - o Value is *DynamicQuotaEntity* for the DynamicQuota transparent data
- *fieldNameX*: A user defined field in the data row
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

**NOTE:** For multi-value fields, the value can contain a comma separated list of values on a single line. For example, a,b,c

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

Value is either IMSI, MSISDN, NAI, or AccountId
- *keyValueX*: Corresponding key field value assigned to *keyNameX*
- *rowIdName*: Name of the XML attribute that identifies the row in the data blob
  - o Value is name for Quota transparent data
  - o Value is name for 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
  - o Value is *cid* or *Type* for the Quota transparent data
  - o Value is *InstanceId* or *Type* for the DynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

## Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.
 

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values

- *affected*: The number of subscribers updated. A value of 1 indicates the row existed and the field was deleted. A value of 0 indicates the field did not exist

**Error Codes 18: Delete Row Field**

| Error Code              | Description   |
|-------------------------|---|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found  |
| OCC_CONSTR_VIOLATION    | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE     | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND      | Register Data Not Found   |
| ROW_NOT_FOUND           | Data row specified is not found   |
| MULTIPLE_ROWS_FOUND     | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria                   |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

**Delete Row Field Examples****Request 1**

A request is made to delete the inputVolume field in the Q1 data row of the Quota data. The Q1 data row exists in the Quota data, and is there is only one row called Q1. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the field in the data row in the Quota data was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to delete the outputVolume field in the Q3 data row of the Quota data. The Quota data contains two rows called Q3. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
  </where>
</req>
```

**Response 2**

The request fails, because there are two Quota rows called Q3. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

**Request 3**

A request is made to delete the outputVolume field in the Q4 data row of the Quota data with the cid 11223344. The Q4 data row exists in the Quota data, and there are two rows called Q4, one with cid 11223344 and one with cid 99887766. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q4"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

**Response 3**

The request is successful, and the outputVolume field in the Q4 data row in the Quota data was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 4**

A request is made to delete the InitialInputVolume field in the DQ1 data row of the DynamicQuota data with the InstanceId 11223344. The DQ1 data row exists in the DynamicQuota data, and there are two rows called DQ1, one with InstanceId 11223344 and one with InstanceId 99887766. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="DynamicQuotaEntity"/>
  <set>
    <expr><attr name="InitialInputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
```

```

    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
    <expr><attr name="name"/><op value="="/>
      <value val="DQ1"/></expr>
  <expr><attr name="InstanceId"/><op value="="/>
    <value val="11223344"/></expr>
  </where>
</req>

```

**Response 4**

The request is successful, and the InitialInputVolume field in the DQ1 data row in the DynamicQuota data was deleted for the row with the specified InstanceId. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 5**

A request is made to delete the inputVolume field in the Q1 data row of the Quota data. The Q1 data row exists in the Quota data, there is only one row called Q1 and the inputVolume field does not exist. The request is not required in the response.

```

<req name="update" resonly="y">
  <ent name="QuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>

```

**Response 5**

The request is successful and the original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**6.6 Subscriber Data Field Commands**

A transparent data entity may contain data that is organized in fields where each field is defined as a name value pair 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>fieldName</name>
  <value>fieldValue</value>
</property>

```

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

**NOTE:** Subscriber data field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an OPER\_NOT\_ALLOWED error being returned.

**Table 18: Summary of Subscriber Data Commands**

| Command           | Description   | Keys  | Command Syntax  |
|-------------------|---|---|---|
| Create Data Field | Create/update data field in transparent data of the specified type. | (MSISDN, IMSI, NAI or AccountId) and Field Name | <pre>&lt;req name="insert"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="fieldName"&gt;       &lt;value val="fieldValue"&gt;     &lt;/expr&gt; ...</pre> |
| Get Data Field    | Retrieve data field from transparent data of the specified type.    |   | <pre>&lt;req name="select"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="fieldName"&gt;       &lt;value val="fieldValue"&gt;     &lt;/expr&gt; ...</pre> |
| Update Data Field | Update data field in transparent data of the specified type.        |   | <pre>&lt;req name="update"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="fieldName"&gt;       &lt;value val="fieldValue"&gt;     &lt;/expr&gt; ...</pre> |
| Delete Data Field | Delete data field in transparent data of the specified type.        |   | <pre>&lt;req name="update"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="fieldName"&gt;       &lt;value val="fieldValue"&gt;     &lt;/expr&gt; ...</pre> |

### 6.6.1 Create Data Field

#### Description

This operation creates or updates a field in a transparent data for the subscriber identified by the `keyNameX` and `keyValueX`.

The field name is specified in `fieldNameX`, and the field value is specified in `fieldValueX`.

If the specified field does not exist, it is created. If the field does exist, it is updated/replaced only if the optional `odk` flag is set to `yes`.

#### NOTES

- The `fieldNameX` is not case-sensitive. If a field existed called `mcc`, then an attempt to create/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 Keys of the `keyNameX/keyValueX` values supplied must exist.
- The `entityName` must reference a valid transparent Entity in the Interface Entity Map table in the SEC.

#### Request

**NOTE:** This command allows 2 different formats. One with the `keyNameX/keyValueX` values in the `<set>` element, and another with the `keyNameX/keyValueX` values in a `<where>` element.

#### Format 1

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="keyName1"/><value val="keyValue1"/></expr>
  [
```

```

    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]

</set>
</req>

```

## Format 2

```

<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
  <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]

  <where>
  <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
  </where>
  </set>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- **id:** (Optional) Transaction ID value in the request, and passed back in the response  
values: 1 to 4294967295
- **odk:** (Optional) Indicates that the insert request is converted to an update if the field for the specified entity exists
- **entityName:** A user defined entity type/name for the transparent data  
Value is StateEntity for the State transparent data
- **keyNameX:** A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId
- **keyValueX:** Corresponding key field value assigned to *keyNameX*
- **fieldNameX:** The requested user defined field in the subscriber profile.  
For the State entity, this corresponds to a property in the entity
  - o The *fieldNameX* case is stored exactly as it was sent in the request (This means the original case stored changes if an update is received)
- **fieldValueX:** Corresponding field value assigned to *fieldNameX*.

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

### Response

```
<req name="insert" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers updated. A value of 1 is expected for success

### Error Codes 19: Create Data Field

| Error Code              | Description   |
|-------------------------|---|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found  |
| FIELD_VAL_INVALID       | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION    | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| INVAL_REPEATABLE_ELEM   | Invalid Repeatable Element  |
| INVALID_SOAP_XML        | Invalid SOAP XML  |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |
| REG_EXISTS              | The property exists and the odk flag was not set to indicate the insert request be converted to an update.                              |

### Create 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 does not have an existing State property called `mcc`. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
```



```

    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>

```

**Response 1**

The request is successful, and the property mcc with value 315 was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 2**

A request is made to create a property in the State transparent data for a subscriber, using the alternate request format. The property name is mcc and the property value is 315. The subscriber does not have an existing State property called mcc. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="15141234567"/></expr>
  </where>
</req>

```

**Response 2**

The request is successful, and the property mcc with value 315 was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 3**

A request is made to create a property in the State transparent data for a subscriber. State is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="15145551234"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>

```

**Response 3**

The request is successful, and the property as well as the State entity is created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 4**

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 property exists) The odk attribute is included requesting the data be updated if it exists. The request is not required in the response.

```

<req name="insert" resonly="y" odk="yes">
  <ent name="StateEntity"/>

```

```

<set>
  <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
  <expr><attr name="mcc"/><value val="315"/></expr>
</set>
</req>

```

**Response 4**

The request is successful, and the mcc property was updated. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 5**

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 property exists) The odk attribute is not included (requests the data be updated if it exists.) The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="MSISDN"/><value val="33123654862"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>

```

**Response 5**

The request fails. The error value indicates the mcc property exists, and the affected rows are 0. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="70028" affected="0"/>
</req>

```

**6.6.2 Get Data Field****Description**

This operation retrieves a field for the subscriber that is identified by the Keys specified in keyNameX and keyValueX.

The field is specified in fieldNameX. The fields that match the requested fieldNameX are returned.

**NOTES**

- If the specified field does not exist, null="y" is returned.
- The *fieldNameX* is not case-sensitive. If a field existed called mcc, then an attempt to get a field name called MCC is successful.

**Prerequisites**

- A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.
- The entityName must reference a valid transparent Entity in the Interface Entity Map table in the SEC.
- A field with the given identifier in the transparent data must exist for the subscriber.

**Request**

```

<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <select>
    <expr><attr name="fieldName1"/></expr>
  [

```

```

    <expr><attr name="fieldName2"/></expr>
      :
    <expr><attr name="fieldNameN"/></expr>
  ]
</select>
<where>
  <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
[
  <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
  :
  <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
]
</where>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
  - o n—Includes the original request in the response (default)
- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- **entityName:** A user defined entity type/name for the transparent data  
Value is StateEntity for the State transparent data
- **fieldNameX:** The requested user defined field in the subscriber profile.  
For the State entity, this corresponds to a property in the entity
- **keyNameX:** A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId
- **keyValueX:** Corresponding key field value assigned to *keyNameX*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

## Response

```

<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
<   <rv>fieldValue1</rv> | <rv null="y"> | <rv></rv> >
[
<   <rv>fieldValue2</rv> | <rv null="y"> | <rv></rv> >
<   :
<   <rv>fieldValueN</rv> | <rv null="y"> | <rv></rv> >
]
    </row>
  </rset>
]
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers returned. A value of 1 is expected if the specified data exists (whether or not the field was found). A value of 0 is expected if the data does not exist
- *fieldValueX*: The value of the requested field

**NOTE:** The <rset> (row set) element is optional. It is only present if the request was successful. One <row> element is returned per matching data. One <rv> (field value) element exists for every fieldNameX supplied in the original request. The <rv> elements are ordered the same as the fieldNameX properties were specified in the original request. If the field is valid, but not present in the entity, this is indicated with <rv null="y">. If the field is present, but has an empty value, this is indicated with <rv></rv>.

#### Error Codes 20: Get Data Field

| Error Code              | Description   |
|-------------------------|---|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found  |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found                      |
| REG_DATA_NOT_FOUND      | Register Data Not Found   |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber                                  |

#### Get Data Field Examples

##### Request 1

A request is made to get three state properties from the State transparent data for a subscriber. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="StateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
    <expr><attr name="expire"/></expr>
    <expr><attr name="duration"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

##### Response 1

The request is successful, two values are returned and one property was indicated to not be set. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>315</rv>
      <rv>2015-02-09T11:20:32</rv>
      <rv null="y"/>
    </row>
```

```

</rset>
</req>

```

### **Request 2**

A request is made to get the mcc property in the State transparent data for a subscriber. The mcc property is not set. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="StateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

### **Response 2**

The request is successful, and the property is indicated to not be set. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y">
    </row>
  </rset>
</req>

```

### **Request 3**

A request is made to get the mcc property in the State transparent data for a subscriber. The State Entity does not exist. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="StateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
  </select>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>

```

### **Response 3**

The request fails. The error value indicates the state entity does not exist, and the affected subscribers are 0. The original request is not included.

```

<req name="select" resonly="y">
  <res error="70027" affected="0"/>
</req>

```

## **6.6.3 Update Data Field**

### **Description**

This operation updates an existing field in a transparent data for the subscriber identified by the keyNameX and keyValueX.

The field name is specified in fieldNameX, and the field value is specified in fieldValueX.

If more than one existing field matches the requested fieldNameX, then the update request fails.

**NOTES**

- If the requested fields are valid, but not present, they are created.
- The *fieldNameX* is not case-sensitive. If a field name existed called mcc, then an attempt to update a field called MCC is successful.
- Multiple subscriber key values can be supplied. See section 2.11 for details.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

**Prerequisites**

- A subscriber with the Keys of the *keyNameX/keyValueX* values supplied must exist.
- The *entityName* must reference a valid transparent Entity in the Interface Entity Map table in the SEC.

**Request**

```

<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
</set>
<where>
  <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
</where>
</req>

```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

**Values:**

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response  
Values: 1 to 4294967295
- *entityName*: A user defined entity type/name for the transparent data  
Value is StateEntity for the State transparent data
- *fieldNameX*: The requested user defined field in the subscriber profile.  
For the State entity, this corresponds to a property in the entity
  - o The *fieldNameX* case is stored exactly as it was sent in the request (This means the original case stored changes if an update is received)
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*
- *keyNameX*: A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId

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

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
 Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers updated. A value of 1 is expected for success

**Error Codes 21: Update Data Field**

| Error Code              | Description   |
|-------------------------|---|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found  |
| OCC_CONSTR_VIOLATION    | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_VAL_INVALID       | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE     | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND      | Register Data Not Found   |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

**Update Data Field Examples**

**Request 1**

A request is made to update the *mcc* and *approved* properties of the State transparent data for a subscriber. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><value val="302"/></expr>
    <expr><attr name="approved"/><value val="no"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
```

```
</where>
</req>
```

### Response 1

The request is successful, and mcc and approved property values are updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
```

### Request 2

A request is made to update the approved properties in the State data. The State Entity does not exist for the subscriber. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="approved"/><value val="no"/></expr>
  </set>
  <where>
    <expr><attr name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </where>
</req>
```

### Response 2

The request fails because the State entity does not exist for the subscriber. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

## 6.6.4 Delete Data Field

### Description

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

The field identifier is specified in fieldName.

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

### NOTES

- The deletion of a non-existent field is not considered an error.
- The *fieldNameX* is not case-sensitive. If a field existed called mcc, then an attempt to delete a field called MCC is successful.
- Multiple subscriber key values can be supplied. See section 2.11 for details.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied

### Prerequisites

- A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.
- The entityName must reference a valid transparent Entity in the Interface Entity Map table in the SEC.

### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
```



```

    <expr><attr name="fieldName1"/><op value="="/>
      <value val="" isnull="y"/></expr>
  [
    <expr><attr name="fieldName2"/><op value="="/>
      <value val="" isnull="y"/></expr>
    :
    <expr><attr name="fieldNameN"/><op value="="/>
      <value val="" isnull="y"/></expr>
  ]
</set>
<where>
  <expr><attr name="keyName1"/><op value="="/><value val="keyValue1"/></expr>
  [
    <expr><attr name="keyName2"/><op value="="/><value val="keyValue2"/></expr>
    :
    <expr><attr name="keyNameN"/><op value="="/><value val="keyValueN"/></expr>
  ]
</where>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Value is StateEntity for the State transparent data

- **fieldNameX:** The requested user defined field in the subscriber profile.

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

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

Value is either IMSI, MSISDN, NAI, or AccountId

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

## Response

```

<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values: A string with 1 to 4096 characters

- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of subscribers updated. A value of 1 indicates the field was deleted. A value of 0 indicates the field did not exist

**Error Codes 22: Delete Data Field**

| Error Code              | Description   |
|-------------------------|---|
| INTF_ENTY_NOT_FOUND     | Interface Entity Not Found  |
| OCC_CONSTR_VIOLATION    | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_UNDEFINED         | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE     | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND      | Register Data Not Found   |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

**Delete Data Field Examples****Request 1**

A request is made to delete the mcc property of the State transparent data for a subscriber. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful and mcc property is deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to delete the mcc property in State data for a subscriber. The mcc property does not exist for the State data. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
  </where>
</req>
```

### Response 2

The request is successful and the original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

### Request 3

A request is made to delete the mcc property in State data for a subscriber. The StateEntity does not exist for the subscriber. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="StateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
  </where>
</req>
```

### Response 3

The request fails. The error value indicates the subscriber does not have State data, and the affected rows are 0. The original request is not included.

```
<req name="select" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

## 6.7 Subscriber Special Operation Commands

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

The required row is identified in the request by the rowIdName/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 19: 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 | Format 1<br><req name="operation"><br><oper name="Reset" ent="entityName"><br>(different/old supported format)<br>Format 2<br><req name="operation"><br><oper name="ResetQuota"> |
| GetSubscriptions | Retrieve Subscriber's Notification Subscription Data | (MSISDN, IMSI, NAI or AccountId)                    | <req name="operation" resonly="y"><br><oper name="GetSubscriptionStatus">  |

### 6.7.1 Reset Quota

#### Description

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

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

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

## NOTES

- The *rowIdName* is case-sensitive. If a row existed 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 Keys of the *keyNameX/keyValueX* values supplied must exist.
- The Quota transparent data must exist for the subscriber.
- The specified Quota row must exist in the Quota transparent data.

## Request

### Format 1

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="Reset" ent="entityName">
    <expr><param name="keyName1"/><op value="="/>
      <value val="keyValue1"/></expr>
  [
    <expr><param name="keyName2"/><op value="="/>
      <value val="keyValue2"/></expr>
    :
    <expr><param name="keyNameN"/><op value="="/>
      <value val="keyValueN"/></expr>
  ]
  <expr><param name="rowIdName"/><op value="="/>
    <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
  </oper>
</req>
```

### Format 2

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="ResetQuota">
    <expr><param name="keyName1"/><op value="="/>
      <value val="keyValue1"/></expr>
  [
    <expr><param name="keyName2"/><op value="="/>
      <value val="keyValue2"/></expr>
    :
    <expr><param name="keyNameN"/><op value="="/>
      <value val="keyValueN"/></expr>
  ]
  <expr><param name="name"/><op value="="/>
    <value val="quotaName"/></expr>
```

**NOTE:** Format 2 is deprecated and Format 1 must be used (Format 2 is still supported).

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request

- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response  
Values: 1 to 4294967295
- *entityName*: A user defined entity type/name for the transparent data  
Value is QuotaEntity for the Quota transparent data
- *keyNameX*: A key field in the subscriber profile  
Value is either IMSI, MSISDN, NAI, or AccountId
- *keyValueX*: Corresponding key field value assigned to *keyNameX*
- *rowIdName*: Name of the XML attribute that identifies the row in the data blob  
Value is name for Quota transparent data
- *rowIdValue/quotaName*: 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 for the Quota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

**Response**

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values:A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of subscribers reset. A value of 1 is expected for success

**Error Codes 23: Reset Quota**

| Error Code              | Description   |
|-------------------------|---|
| KEY_NOT_FOUND           | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND      | Register Data Not Found   |
| ROW_NOT_FOUND           | Data row specified is not found   |
| MULTIPLE_ROWS_FOUND     | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria |
| MULTIPLE_KEYS_NOT_MATCH | Multiple keys supplied do not refer to the same subscriber  |

**Reset Quota Examples****Request 1**

A request is made to reset the Q1 Quota row for a subscriber. The subscriber has Quota data, and the Quota data contains a Quota row called Q1. The request is not required in the response.

```
<req name="operation">
  <oper name="ResetQuota">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><param name="name"/><op value="="/><value val="Q1"/></expr>
  </oper>
</req>
```

**Response 1**

The request is successful, and the specified Quota row was reset. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to reset the Monthly Quota row. The subscriber does not have Quota data. The request is not required in the response.

```
<req name="operation">
  <oper name="ResetQuota">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="15141234567"/></expr>
    <expr><param name="name"/><op value="="/><value val="Monthly"/></expr>
  </oper>
</req>
```

**Response 2**

The request fails. The error value indicates the subscriber does not have Quota data, and the affected rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70027" affected="0"/>
</req>
```

**Request 3**

A request is made to reset the Q6 Quota row. The subscriber has Quota data, but the Quota data does not contain a Quota row called Q6. The request is not required in the response.

```
<req name="operation">
  <oper name="ResetQuota">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><param name="name"/><op value="="/><value val="Q6"/></expr>
  </oper>
</req>
```

**Response 3**

The request fails, because the Q6 data row was not present. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70032" affected="0"/>
</req>
```

**Request 4**

A request is made to reset the Q1 Quota row for a subscriber. Three keys are provided for the subscriber, and all keys are valid for the subscriber. The subscriber has Quota data, and the Quota data contains a Quota row called Q1. The request is not required in the response.

```
<req name="operation">
  <oper name="ResetQuota">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    <expr><param name="IMSI"/><op value="="/>
      <value val="184569547984229"/></expr>
    <expr><param name="NAI"/><op value="="/>
      <value val="mum@foo.com"/></expr>
    <expr><param name="name"/><op value="="/><value val="Q1"/></expr>
  </oper>
</req>
```

**Response 4**

The request is successful, and the specified Quota row was reset. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 5**

A request is made to reset the Q1 Quota row for a subscriber. The subscriber has Quota data, and the Quota data contains a Quota row called Q1. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="QuotaEntity">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  <expr><param name="name"/><op value="="/><value val="Q1"/></expr>
  </oper>
</req>
```

**Response 5**

The request is successful, and the specified Quota row was reset. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 6**

A request is made to reset the Q1 Quota row for a subscriber. The subscriber has Quota data, the Quota data contains a Quota row called Q1 and instanceFieldName cid field with value of 1234. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="QuotaEntity">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  <expr><param name="name"/><op value="="/><value val="Q1"/></expr>
  <expr><param name="cid" /><op value="="/><value val="1234"/></expr>
  </oper>
</req>
```

**Response 6**

The request is successful, and the specified Quota row was reset. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
```

```
</req>
```

## 6.7.2 Get Subscription Status

### Description

This operation retrieves all the notification subscriptions for a subscriber that is identified by the Keys specified in `keyNameX` and `keyValueX`.

The `keyNameX` and `keyValueX` values are required in the request in order to identify the subscriber. The response content includes Subscription information associated with the subscriber used for PNR purpose.

### Prerequisites

- A subscriber with the Keys of the `keyNameX/keyValueX` values supplied must exist.
- The subscriber should contain atleast one subscription

### Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="GetSubscriptionStatus">
    <expr><param name="keyName1"/><op value="="/>
      <value val="keyValue1"/></expr>
  [
    <expr><param name="keyName2"/><op value="="/>
      <value val="keyValue2"/></expr>
    :
    <expr><param name="keyNameN"/><op value="="/>
      <value val="keyValueN"/></expr>
  ]
  </oper>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *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*

**NOTE:** Multiple subscriber key values can be supplied. See section 2.11 for details.

### Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
  [
    <rset>
      <row>
        <rv>
          <SubscriptionStatus>
            :
            :
          </rv>
        </row>
      </rset>
    ]
  </req>
```



```

        </SubscriptionStatus>
    </rv>
</row>
</rset>
]
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request  
 Values:A string with 1 to 4096 characters
- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of subscribers returned. A value of 1 is expected for success

**NOTE:**

- The `<rset>` (row set) element is optional. It is only present if the request was successful. Only a single `<row>` element is returned.
- The time values(“creation time” and “expiration time”) are returned in epoch format not in DD:MM:YYYY hh:mm:ss format

**Error Codes 24: Get Subscription Status**

| Error Code    | Description  |
|---------------|--|
| KEY_NOT_FOUND | Key Not Found. A subscriber or pool with the given key cannot be found |
| ROW_NOT_FOUND | Data row specified is not found  |

**Get Subscriber Status Examples**

**Request 1**

A request is made to retrieve the subscription information for a subscriber. The subscriber has only one subscription from a single client and on only one key(imsi:222222222). The request is not required in the response.

```

<req name="operation">
  <oper name="GetSubscriptionStatus">
    <expr><param name="MSISDN"/><op value="/">
      <value val="33123654862"/></expr>
  </oper>
</req>

```

**Response 1**

The request is successful. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <SubscriptionStatus>
          <subscribingClient name="seagull1a.seagull1.com">
            <userIdentity name="sip:222222222@ims.mnc480.mcc311.3gppnetwork.org">
              <creationTime>"1545220213"</creationTime>
              <expirationTime>"1545220213"</expirationTime>
            </userIdentity>
          </subscribingClient>
        </SubscriptionStatus>
      </rv>
    </row>
  </rset>
</req>

```

```

        </subscribingClient>
      </SubscriptionStatus>
    </rv>
  </row>
</rset>
</req>

```

**Request 2**

A request is made to retrieve the subscription information for a subscriber. The subscriber has two subscriptions from a single client and on two keys(imsi:222222222 and MSISDN: 33123654862). The request is not required in the response.

```

<req name="operation">
  <oper name="GetSubscriptionStatus">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>

```

**Response 2**

The request is successful. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <SubscriptionStatus>
          <subscribingClient name="seagull1a.seagull1.com">
            <userIdentity name="sip:222222222@ims.mnc480.mcc311.3gppnetwork.org">
              <creationTime>"1545220213"</creationTime>
              <expirationTime>"1545220213"</expirationTime>
            </userIdentity>
            <userIdentity name="tel: 33123654862">
              <creationTime>"1545220213"</creationTime>
              <expirationTime>"1545220213"</expirationTime>
            </userIdentity>
          </subscribingClient>
        </SubscriptionStatus>
      </rv>
    </row>
  </rset>
</req>

```

**Request 3**

A request is made to retrieve the subscription information for a subscriber. The subscriber has three subscriptions from two clients and on two keys(imsi:222222222 and MSISDN: 33123654862). The request is not required in the response.

```

<req name="operation">
  <oper name="GetSubscriptionStatus">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>

```

**Response 3**

The request is successful. The original request is not included.

```

<req name="operation" resonly="y">

```

```

<res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <SubscriptionStatus>
          <subscribingClient name="seagull1a.seagull1.com">
            <userIdentity name="tel: 33123654862">
              <creationTime>"1545220213"</creationTime>
              <expirationTime>"0"</expirationTime>
            </userIdentity>
            <userIdentity name="sip:222222222@ims.mnc480.mcc311.3gppnetwork.org">
              <creationTime>"1545220213"</creationTime>
              <expirationTime>"0"</expirationTime>
            </userIdentity>
          </subscribingClient>
          <subscribingClient name="seagull1b.seagull1.com">
            <userIdentity name="tel: 33123654862">
              <creationTime>"1545220213"</creationTime>
              <expirationTime>"0"</expirationTime>
            </userIdentity>
          </subscribingClient>
        </SubscriptionStatus>
      </rv>
    </row>
  </rset>
</req>

```

**Request 4**

A request is made to retrieve the subscription information for a subscriber who doesn't have any subscription.

```

<req name="operation">
  <oper name="GetSubscriptionStatus">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>

```

**Response 4**

The request fails because the subscriber has no subscription information. The original request is not included.

```

<req name="operation" resonly="y">
  <res affected="0" error="70032" />
</req>

```

## Chapter 7. Pool Provisioning

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

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

**NOTE:** For command responses, the error code values described are listed in Appendix B.

### 7.1 Pool Profile Commands

**Table 20: Summary of Pool Profile Commands**

| Command     | Description   | Keys   | Command Syntax   |
|-------------|---|--------|--|
| Create Pool | Creates a pool profile using the field-value pairs that are specified in the request content. |        | <pre>&lt;req name="insert"&gt;   &lt;ent name="Pool"/&gt; &lt;/req&gt;</pre> |
| Get Pool    | Get pool profile data   |        | <pre>&lt;req name="select"&gt;   &lt;ent name="Pool"/&gt; &lt;/req&gt;</pre> |
| Delete Pool | Delete pool profile data and all opaque data associated with the pool                         | PoolID | <pre>&lt;req name="delete"&gt;   &lt;ent name="Pool"/&gt; &lt;/req&gt;</pre> |

#### 7.1.1 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 request as part of the where element. Request content includes poolid, and field-value pairs, all as specified in the Subscriber Entity Configuration.

##### NOTES

- The pool profile data provided is fully validated against the definition in the SEC. If the validation check fails, then the request is rejected.
- An entire entity for the pool can be created by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the entire XML blob value in *cdataFieldValue*.
- 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**

```

<req name="insert" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolID"/><value val="poolId"/></expr>
  [
    <expr>
  <
    <attr name="fieldName1"/><value val="fieldValue1"/>
  |
    <attr name="cdataFieldName1"/><op value="="/>
    <cdata><![CDATA[cdataFieldValue1]]></cdata>
  >
    </expr>
  <expr>
  <
    <attr name="fieldName2"/><value val="fieldValue2"/>
  |
    <attr name="cdataFieldName2"/><op value="="/>
    <cdata><![CDATA[cdataFieldValue2]]></cdata>
  >
    </expr>
  :
  <expr>
  <
    <attr name="fieldNameN"/><value val="fieldValueN"/>
  |
    <attr name="cdataFieldNameN"/><op value="="/>
    <cdata><![CDATA[cdataFieldValueN]]></cdata>
  >
    </expr>
  ]
  </set>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o **y**—Provides the result only, does not include the original request
- o **n**—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Values: 1 to 99999999999999999999

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

**NOTE:** for multi-value fields, the value contains a comma separated list of values on a single line. For example, a,b,c

- **cdataFieldNameX:** A user defined field in the pool profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC

Value is either PoolQuota, PoolState, or PoolDynamicQuota

- **cdataFieldValueX:** Contents of the XML data blob for *cdataFieldNameX*

**NOTE:** PoolID/field order in the request is not important.

**Response**

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of Pools created. A value of 1 is expected for success

**Error Codes 245: Create Pool**

| Error Code            | Description   |
|-----------------------|---|
| FIELD_VAL_INVALID     | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION  | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| INVALID_SOAP_XML      | Invalid SOAP XML  |
| FIELD_UNDEFINED       | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_EXISTS            | Key exists. A subscriber or pool exists with the given key  |
| INVALID_KEY_VALUE     | The key value supplied is invalid, due to invalid characters, format and so on  |
| MULT_VER_TAGS_FOUND   | Multiple Version Tags Found   |
| INVAL_REPEATABLE_ELEM | Invalid Repeatable Element  |
| INTF_ENTY_NOT_FOUND   | Interface Entity Not Found  |
| INVALID_XML           | Invalid Input XML   |
| OPER_NOT_ALLOWED      | Operation Not Allowed   |

**Examples****Request 1**

A pool is created, with PoolID. The BillingDay and Entitlement fields are set. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
```

```

    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPassPlus"/></expr>
  </set>
</req>

```

**Response 1**

The request is successful, and the pool was created.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 2**

A pool is created, with PoolID. The BillingDay and Entitlement fields are set. The PoolQuota and PoolState entities are also created. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass,DayPassPlus"/></expr>
    <expr><attr name="PoolQuota"/><op value=""/>
      <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <usage>
          <version>3</version>
          <quota name="Weekend">
            <totalVolume>500</totalVolume>
            <Type>quota</Type>
            <QuotaState>active</QuotaState>
            <nextResetTime>2014-01-10T02:00:00</nextResetTime>
          </quota>
          <quota name="Evenings">
            <totalVolume>300</totalVolume>
            <Type>quota</Type>
            <QuotaState>active</QuotaState>
            <nextResetTime>2014-02-01T00:00:00</nextResetTime>
          </quota>
        </usage>]]>
      </cdata>
    </expr>
    <expr><attr name="PoolState"/><op value=""/>
      <cdata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <state>
          <version>1</version>
          <property>
            <name>shared</name>
            <value>yes</value>
          </property>
          <property>
            <name>expire</name>
            <value>2014-02-09T11:20:32</value>
          </property>
        </state>]]>
      </cdata>
    </expr>
  </set>
</req>

```

**Response 2**

The request is successful, and the pool was created.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 3**

A pool is created, with PoolID. PSO feature is enabled. The PoolID falls in a range that is maintained by a different UDR instance. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="BillingDay"/><value val="1"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    <expr><attr name="Entitlement"/><value val="DayPassPlus"/></expr>
    <expr><attr name="Type"/><value val="Enterprise"/></expr>
  </set>
</req>
```

**Response 3**

The request fails. The error indicates this operation is not allowed on non pool host UDR.

```
<req name="insert" resonly="y">
  <res error="70026" affected="0"/>
</req>
```

**7.1.2 Get Pool****Description**

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

A poolId is required in the request in order to identify the pool. 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**

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Values: 1 to 99999999999999999999

**Response**

```
<req name="select" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
```



```
<res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>
        <![CDATA[cdataRowValue]]>
      </rv>
    </row>
  </rset>
]
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
 Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 is expected for success
- *cdataRowValue*: Contents of the pool profile XML data blob

**NOTE:** The `<rset>` (row set) element is optional. It is only present if the request was successful. Only a single `<row>` element is returned, with a single `<rv>` (row value) element containing an XML CDATA construct containing the requested pool profile data (XML blob).

**Error Codes 26: Get Pool**

| Error Code          | Description  |
|---------------------|--|
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found |
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found   |

**Get Pool Examples**

**Request 1**

A request is made to get pool profile data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the pool profile data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?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>
          </pool>
        ]>
      </rv>
    </row>
  </rset>
</req>
```

```

        <field name="Custom15">allo</field>
      </pool>]]>
    </rv>
  </row>
</rset>
</req>

```

### 7.1.3 Delete Pool

#### Description

This operation deletes all profile data (field-value pairs) and opaque data for the pool that is identified by the poolId.

#### Prerequisites

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

The pool must not have subscriber members, or the request fails.

#### Request

```

<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>

```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Values: 1 to 9999999999999999999999

#### Response

```

<req name="delete" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>

```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 is expected for success
- *cdataRowValue*: Contents of the pool profile XML data blob

**Error Codes 257: Delete Pool**

| Error Code          | Description   |
|---------------------|---|
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found    |
| HAS_POOL_MEMBERS    | Has pool members. A pool cannot be deleted when it has member subscribers |
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found  |

**Delete Pool Examples****Request 1**

The pool with the given PoolID is deleted. The pool exists. The request is not included in the response.

```
<req name="delete" resonly="y">
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the pool was deleted.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

The pool with the given PoolID is deleted. The pool does not exist. The request is not included in the response.

```
<req name="delete" resonly="y">
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

**Response 2**

The request fails. The error value indicates a pool with the given PoolID does not exist, and the affected rows are 0. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70019" affected="0"/>
</req>
```

**Request 3**

The pool with the given PoolID is deleted. The pool exists, but has member subscribers. The request is not included in the response.

```
<req name="delete" resonly="y">
  <ent name="Pool"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

**Response 3**

The request fails. The error value indicates the pool has member subscribers, and the affected rows are 0. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="70022" affected="0"/>
</req>
```

**7.2 Pool Field Commands****Table 21: Summary of Pool Field Commands**

| Command            | Description   | Keys   | Command Syntax  |
|--------------------|---|--------|---|
| Add Field Value    | Add a value to the specified field. This operation does not affect any pre-existing values for the field. | PoolID | <pre>&lt;req name="update"&gt;   &lt;ent name="Pool"/&gt;   &lt;oper name="AddToSet"&gt;</pre>            |
| Get Field          | Retrieve values for the specified fields  |        | <pre>&lt;req name="select"&gt;   &lt;ent name="Pool"/&gt;</pre>   |
| Update Field       | Updates fields to the specified values  |        | <pre>&lt;req name="update"&gt;   &lt;ent name="Pool"/&gt;</pre>   |
| Delete Field       | Delete all values for the specified fields  |        | <pre>&lt;req name="update"&gt;   &lt;ent name="Pool"/&gt;   ... &lt;value val="" isnull="y"/&gt;...</pre> |
| Delete Field Value | Delete fields with the specified values   |        | <pre>&lt;req name="update"&gt;   &lt;ent name="Pool"/&gt;   &lt;oper name="RemoveFromSet"&gt;</pre>       |

**7.2.1 Add Field Value****Description**

This operation adds one or more values to the specified multi-value field for the pool identified by the poolId.

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

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

**NOTES**

- If a value being added exists, the request fails.
- 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 being a,b,c,A.
- A request to add fields values can also be mixed with a request to update or delete a fields. But, the same field for which an AddToSet operation is being performed cannot also be updated or deleted, else the request fails.
- A request to add fields values using the AddToSet operation can also contain a RemoveFromSet operation to delete fields values. If both operations are included in the same request, the AddToSet is performed before the RemoveFromSet, irrespective of the order in which they are supplied.

**Prerequisites**

- A pool with a key 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.

- Each *fieldValueX* being added must not be present in the field.

### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="fieldName1"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    [
      <expr><attr name="fieldName2"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
      :
      <expr><attr name="fieldNameX"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    ]
  </oper>
</set>
<where>
  <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
</where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *fieldName*: A user defined field in the pool profile
- *fieldValueX*: Corresponding field value assigned to *fieldName*
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length

Values: 1 to 9999999999999999999999

**NOTE:** One or more *fieldValueX* values for a *fieldNameX* can be supplied. To add more than one value, either supply a comma separated list of values, or include multiple *<expr>* elements for the field.

### Response

```
<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 is expected for success

**Error Codes 268: Add Field Value**

| Error Code           | Description   |
|----------------------|---|
| OCC_CONSTR_VIOLATION | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field                       |
| FIELD_UNDEFINED      | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE  | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND        | Key Not Found. A subscriber or pool with the given key cannot be found  |
| VALUE_EXISTS         | List value added exists   |
| FLD_NOT_MULTI        | Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-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. The request is not required in the response.

```
<req name="update">
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the value was added to the Entitlement field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to add the values HighSpeed and Unlimited to the Entitlement field. The Entitlement field is a valid multi-value field. Neither value is present in the Entitlement field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Entitlement"/><value val="HighSpeed"/></expr>
      <expr><attr name="Entitlement"/><value val="Unlimited"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

```
</req>
```

### Response 2

The request is successful, and the values were added to the Entitlement field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

### 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. The request is not required in the response.

```
<req name="update">
  <ent name="Pool"/>
  <set>
    <oper name="AddToSet">
      <expr><attr name="Tier"/><value val="Gold"/></expr>
    </oper>
  </set>
</where>
  <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
</where>
</req>
```

### Response 3

The request fails. The error value indicates the Tier field is not a multi-value field, and the affected rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70034" affected="0"/>
</req>
```

## 7.2.2 Get Field

### Description

This operation retrieves the values for the specified fields for the pool identified by the specified poolId.

**NOTE:** An entire entity for the pool can be retrieved by specifying an opaqueDataType corresponding to the interface entity name in the SEC.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- Each requested field fieldNameX must be a valid field in the pool profile.
- Each requested opaqueDataTypeX must reference a valid Entity in the Interface Entity Map table in the SEC.

### Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <select>
  [
    <expr><attr name="fieldName1"/></expr>
    <expr><attr name="fieldName2"/></expr>
    :
    <expr><attr name="fieldNameN"/></expr>
  ]
  [
    <expr><attr name="opaqueDataType1"/></expr>
```

```

    <expr><attr name="opaqueDataType2" /></expr>
    :
    <expr><attr name="opaqueDataTypeN" /></expr>
]
</select>
<where>
  <expr><attr name="PoolID" /><op value="=" /><value val="poolId" /></expr>
</where>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o **y**—Provides the result only, does not include the original request
- o **n**—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- **fieldNameX:** A user defined field in the pool profile
- **fieldValueX:** Corresponding field value assigned to *fieldNameX*
- **opaqueDataTypeX:** A user defined field in the subscriber profile, that represents a transparent or opaque data entity

Value is either PoolQuota, PoolState, or PoolDynamicQuota

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

Values: 1 to 99999999999999999999

## NOTES

- At least one *fieldNameX/opaqueDataTypeX* field must be requested.
- The order in which *fieldNameX/opaqueDataTypeX* are specified in the request is not important.

## Response

```

<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
[
  <rv>rowValue1</rv> | <rv null="y"> | <rv></rv> >
<
  <rv>rowValue2</rv> | <rv null="y"> | <rv></rv> >
  :
<
  <rv>rowValueN</rv> | <rv null="y"> | <rv></rv> >
]
[
  <rv>cdataRowValue1</rv> | <rv null="y"> >
<
  <rv>cdataRowValue2</rv> | <rv null="y"> >
  :
<
  <rv>cdataRowValueN</rv> | <rv null="y"> >
]
    </row>
  </rset>
]
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request



Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 is expected for success
- *rowValueX*: The value of the requested field (for normal fields, not for opaque/transparent entities)

**NOTE:** for multi-value fields, the value contains a comma separated list of values on a single line. For example, a,b,c

- *cdataRowValueX*: Contents of the XML data blob (for requested fields that are opaque/transparent entities)

**NOTE:** The <rset> (row set) element is optional. It is only present if the request was successful. Only a single <row> element is returned. One <rv> (row value) element exists for every fieldNameX or opaqueDataTypeX supplied in the original request. The <rv> elements are ordered the same as the fieldNameX/opaqueDataTypeX fields were specified in the original request. If the field is valid, but not present in the entity, this is indicated with <rv null="y">. If the field is present, but has an empty value, this is indicated with <rv></rv>.

#### Error Codes 279: Get Field

| Error Code      | Description   |
|-----------------|---|
| FIELD_UNDEFINED | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND   | Key Not Found. A subscriber or pool with the given key cannot be found                      |

#### Get Field Examples

##### Request 1

A request is made to get the PoolID, Entitlement, Tier, and BillingDay fields. The request is not required in the response.

```
<req name="select" >
  <ent name="Pool"/>
  <select>
    <expr><attr name="PoolID"/></expr>
    <expr><attr name="Entitlement"/></expr>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="BillingDay"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

##### Response 1

The request is successful, and the 4 requested values are returned (the Entitlement is a multi-value field). The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>1000</rv>
      <rv>DayPass,WeekPass,Weekend</rv>
      <rv>Prepaid</rv>
      <rv>23</rv>
```

```

    </row>
  </rset>
</req>

```

### Request 2

A request is made to get the MSISDN, and BillingDay fields, as well as the PoolQuota and PoolState entity data. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="Pool"/>
  <select>
    <expr><attr name="PoolID"/></expr>
    <expr><attr name="BillingDay"/></expr>
    <expr><attr name="PoolQuota"/></expr>
    <expr><attr name="PoolState"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>

```

### Response 2

The request is successful, and the 4 requested values are returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>2000</rv>
      <rv>11</rv>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="AggregateLimit">
              <cid>9223372036854775807</cid>
              <time>3422</time>
              <totalVolume>1000</totalVolume>
              <inputVolume>980</inputVolume>
              <outputVolume>20</outputVolume>
              <serviceSpecific>12</serviceSpecific>
              <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
            </quota>
          </usage>]]>
      </rv>
      <rv>
        <![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>]]>
      </rv>
    </row>
  </rset>
</req>

```

### Request 3

A request is made to get the Custom10, Entitlement, Tier, and Custom20 fields. The Entitlement and Tier fields are set in the XML blob, the Custom10 field is not set, and the Custom20 field is set, but has an empty value. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Pool"/>
  <select>
    <expr><attr name="Custom10"/></expr>
    <expr><attr name="Entitlement"/></expr>
    <expr><attr name="Tier"/></expr>
    <expr><attr name="Custom20"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
  </where>
</req>
```

### Response 3

The request is successful, and the 4 requested values are returned (the Entitlement is a multi-value field). The Custom10 field is indicated as unset, and the Custom20 field is indicated as empty. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
      <rv>1,14,2,8</rv>
      <rv>Prepaid</rv>
      <rv></rv>
    </row>
  </rset>
</req>
```

## 7.2.3 Update Field

### Description

This operation updates a fields to the specified values for the pool identified by the specified poolId. This operation replaces (sets) the values of the fields, which means that any existing values for the fields are deleted first.

For multi-value fields, all existing values are removed and only the new values specified are inserted. Adding values to a current set is accomplished using Add Field Value. 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 d (it is not a,b,c,d).

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

### NOTES

- If the requested fields are valid, but not present, they are created.
- An entire entity for the pool can be replaced by specifying a *cdataFieldName* corresponding to the interface entity name in the SEC, and supplying the entire XML blob value in *cdataFieldValue*.
- 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 a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.

- If a field being updated is specified more than once in a request, the last value specified is used.
- 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 the key of the poolId supplied must exist.
- Each requested field fieldName must be a valid field in the pool profile.
- Each requested cdataFieldName must be a valid pooled transparent/opaque interface entity name for a pool.

### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <expr>
  <
    <attr name="fieldName1"/><value val="fieldValue1"/>
  |
    <attr name="cdataFieldName1"/><op value="="/>
    <cdata><![CDATA[cdataFieldValue1]]></cdata>
  >
  </expr>
  [
    <expr>
  <
    <attr name="fieldName2"/><value val="fieldValue2"/>
  |
    <attr name="cdataFieldName2"/><op value="="/>
    <cdata><![CDATA[cdataFieldValue2]]></cdata>
  >
  </expr>
  :
  <expr>
  <
    <attr name="fieldNameN"/><value val="fieldValueN"/>
  |
    <attr name="cdataFieldNameN"/><op value="="/>
    <cdata><![CDATA[cdataFieldValueN]]></cdata>
  >
  </expr>
  ]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Values: 1 to 99999999999999999999

- *CDATAFieldNameX*: A user defined field in the pool profile, that represents a transparent or opaque data entity, as per the defined interface entity name in the SEC  
Value is either PoolQuota, PoolState, or PoolDynamicQuota
- *CDATAFieldValueX*: Contents of the XML data blob for *CDATAFieldNameX*

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 is expected for success

**Error Codes 28: Update Field**

| Error Code            | Description   |
|-----------------------|---|
| FIELD_VAL_INVALID     | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION  | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_UNDEFINED       | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE   | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND         | Key Not Found. A subscriber or pool with the given key cannot be found  |
| INTF_ENTY_NOT_FOUND   | Interface Entity Not Found  |
| INVAL_REPEATABLE_ELEM | Invalid Repeatable Element  |
| INVALID_XML           | Invalid Input XML   |
| OPER_NOT_ALLOWED      | Operation Not Allowed   |

**Examples**

**Request 1**

A request is made to update the value of the BillingDay field to 23, and the Tier field to Gold. The request is not required in the response.

```
<req name="update">
```

```

<ent name="Pool"/>
<set>
  <expr><attr name="BillingDay"/><value val="23"/></expr>
  <expr><attr name="Tier"/><value val="Gold"/></expr>
</set>
</where>
  <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
</where>
</req>

```

**Response 1**

The request is successful, and the BillingDay and Tier values were updated. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 2**

A request is made to update the value of the BillingDay field to 18, and the entire PoolState entity. The request is not required in the response.

```

<req name="update">
  <ent name="Pool"/>
  <set>
    <expr><attr name="BillingDay"/><value val="18"/></expr>
    <expr><attr name="PoolState"/><op value="="/>
      <CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <state>
          <version>1</version>
          <property>
            <name>shared</name>
            <value>yes</value>
          </property>
          <property>
            <name>expire</name>
            <value>2014-02-09T11:20:32</value>
          </property>
        </state>]]>
      </CDATA>
    </expr>
  </set>
  </where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>

```

**Response 2**

The request is successful, and the BillingDay and PoolState values were updated. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 3**

A request is made to delete the Type field from an enterprise pool that has more than the maximum members allowed for a basic pool. The request is not required in the response. (**NOTE:** deleting the Type field triggers a conversion from an enterprise pool to a basic pool).

```

<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="Type"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>

```

```

    <where>
    <expr><attr name="PoolID"/><op value="="/>
    <value val="100000"/></expr>
    </where>
  </req>

```

**Response 3**

The request fails. The error value indicates the enterprise to basic pool conversion failed because the pool has more members than the maximum threshold for a basic pool. The original request is not included.

```

<req name="update" resonly="y">
  <res error="70052" affected="1"/>
</req>

```

**Request 4**

A request is made to update the Type field. PSO feature is enabled. The PoolID falls in a range that is maintained by a different UDR. The request is not required in the response.

```

<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="Type"/><op value="="/>
    <value val="Enterprise"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
    <value val="100000"/></expr>
  </where>
</req>

```

**Response 4**

The request fails. The error indicates this operation is not allowed on non pool host UDR.

```

<req name="insert" resonly="y">
  <res error="70026" affected="0"/>
</req>

```

**7.2.4 Delete Field****Description**

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

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

**NOTES**

- The field being deleted does not need to have a current value. It can be empty (deleted), and the request succeeds.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied

**Prerequisites**

- A pool with the key of the poolId supplied must exist.
- Each requested field fieldNameX must be a valid field in the pool profile.

**Request**

```

<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>

```

```

    <expr><attr name="fieldName1"/><op value="="/>
      <value val="" isnull="y"/></expr>
  [
    <expr><attr name="fieldName2"/><op value="="/>
      <value val="" isnull="y"/></expr>
    :
    <expr><attr name="fieldNameN"/><op value="="/>
      <value val="" isnull="y"/></expr>
  ]
</set>
<where>
  <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
</where>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- **fieldNameX:** A user defined field in the pool profile
- **poolId:** PoolID value of the pool. Numeric value, 1 to 22 digits in length

Values: 1 to 9999999999999999999999

**Response**

```

<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request  
Values: A string with 1 to 4096 characters
- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of pools updated. A value of 1 is expected for success

**Error Codes 291: Delete Field**

| Error Code      | Description   |
|-----------------|---|
| FIELD_UNDEFINED | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND   | Key Not Found. A subscriber or pool with the given key cannot be found                      |



## Delete Field Examples

### Request 1

A request is made to delete the BillingDay and Tier fields. Both fields are valid pool profile fields. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="BillingDay"/><op value="="/>
      <value val="" isnull="y"/></expr>
    <expr><attr name="Tier"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

### Response 1

The request is successful, and the two fields were deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

## 7.2.5 Delete Field Value

### Description

This operation deletes one or more values from the specified field for the pool identified by the poolId in the request.

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

Each individual value is removed from the pool profile. If a supplied value does not exist, then it is ignored. For example, if a profile contains values A,B,C and a request to delete A,B is made, this succeeds and the profile is left with C as the value. If the profile contains A,B,C and a request is made to delete C,D the request succeeds and the profile is left with A,B as the value.

If all values are removed, the entire field is removed from the pool profile (there is not an XML element present).

### NOTES

- 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.
- A request to delete fields values can also be mixed with a request to update or delete a fields. But, the same field for which a RemoveFromSet operation is being performed cannot also be updated or deleted, else the request fails.
- A request to delete fields values using the RemoveFromSet operation can also contain a AddToSet operation to add fields values. If both operations are included in the same request, the AddToSet is performed before the RemoveFromSet irrespective of the order in which they are supplied.

### 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 must be a multi-value field.
- Each fieldValueX being removed must be present in the field.

## Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="fieldName1"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    [
      <expr><attr name="fieldName2"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
      :
      <expr><attr name="fieldNameX"/>
        <value val="fieldValue1[,fieldValue2[, ... fieldValueN]]"/></expr>
    ]
  </oper>
</set>
<where>
  <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
</where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

**NOTE:** For multi-value fields, the value can contain a comma separated list of values on a single line. For example, a,b,c

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

Values: 1 to 99999999999999999999

**NOTE:** One or more *fieldValueX* values for a *fieldNameX* can be supplied. To remove more than one value, either supply a comma separated list of values, or include multiple `<expr>` elements for the field.

## Response

```
<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 is expected for success

**Error Codes 30: Delete Field Value**

| Error Code      | Description   |
|-----------------|---|
| FIELD_UNDEFINED | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_NOT_FOUND   | Key Not Found. A subscriber or pool with the given key cannot be found  |
| FLD_NOT_MULTI   | Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value |

**Delete Field Value Examples****Request 1**

A request is made to remove the value DayPass from the Entitlement field. The Entitlement field is a valid multi-value field. The DayPass value is present in the Entitlement field. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="Entitlement"/><value val="DayPass"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the value was removed from the Entitlement field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to remove the values WeekendPass and Unlimited from the Entitlement field. The Entitlement field is a valid multi-value field. The WeekendPass value is present in the Entitlement field, but the Unlimited value is not. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <oper name="RemoveFromSet">
      <expr><attr name="Entitlement"/><value val="WeekendPass"/></expr>
      <expr><attr name="Entitlement"/><value val="Unlimited"/></expr>
    </oper>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, and the WeekendPass value was removed from the Entitlement field. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

## 7.3 Pool Opaque Data Commands

**Table 22: Summary of Pool Opaque Data Commands**

| Command            | Description                                     | Keys   | Command Syntax  |
|--------------------|---|--------|---|
| Create Opaque Data | Insert pool opaque data of the specified type   | PoolID | <pre>&lt;req name="insert"&gt;   &lt;ent name="Pool"/&gt;</pre>   |
| Get Opaque Data    | Retrieve pool opaque data of the specified type |        | <pre>&lt;req name="select"&gt;   &lt;ent name="Pool"/&gt;</pre>   |
| Update Opaque Data | Update pool opaque data of the specified type   |        | <pre>&lt;req name="update"&gt;   &lt;ent name="Pool"/&gt;</pre>   |
| Delete Opaque Data | Delete pool opaque data of the specified type   |        | <pre>&lt;req name="update"&gt;   &lt;ent name="Pool"/&gt;   ... &lt;value val="" isnull="y"/&gt;...</pre> |

### 7.3.1 Create Opaque Data

#### Description

This operation creates the pool opaque data of the specified poolOpaqueDataType for the pool identified by the poolId in the request.

The pool opaque data is provided in the request in a <CDATA> construct.

#### NOTES

- The opaque data provided in an XML blob is always checked to be valid XML. If the entity is defined as transparent in the SEC, then the XML blob is fully validated against the definition in the SEC. If either validation check fails, then the request is rejected.
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

#### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The poolOpaqueDataType must reference a valid Entity in the Interface Entity Map table in the SEC.
- Pool opaque data of the poolOpaqueDataType must not exist for the pool (unless the odk attribute is specified).

#### Request

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="poolOpaqueDataType"/><op value=""><CDATA>
<![CDATA[
CDATAFieldValue
]]></CDATA></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value=""><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *odk*: (Optional) Indicates that the insert request is converted to an update if the opaque data type specified exists

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

Value is either PoolQuota, PoolState, or PoolDynamicQuota

- *cdataFieldValue*: Contents of the XML data blob
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length

Values: 1 to 99999999999999999999

### Response

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values:A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools inserted/updated. A value of 1 is expected for success

### Error Codes 313: Create Opaque Data

| Error Code            | Description   |
|-----------------------|---|
| INTF_ENTY_NOT_FOUND   | Interface Entity Not Found  |
| MULT_VER_TAGS_FOUND   | Multiple Version Tags Found   |
| FIELD_VAL_INVALID     | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION  | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| INVAL_REPEATABLE_ELEM | Invalid Repeatable Element  |
| INVALID_XML           | Invalid input XML   |

| Error Code      | Description   |
|-----------------|---|
| FIELD_UNDEFINED | Field not defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND   | Key not found. A subscriber or pool with the given key cannot be found                      |
| REG_EXISTS      | Register exists   |

### Create Opaque Data Examples

#### Request 1

A request is made to create the PoolQuota opaque data. The PoolQuota XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolQuota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

#### Response 1

The request is successful, and the PoolQuota opaque data was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### Request 2

A request is made to create the PoolState opaque data. The PoolState XML blob is supplied whole. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolState"/><op value="="/><cdata>
<![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>
]]></cdata></expr>
</set>
<where>
  <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
</where>
</req>

```

**Response 2**

The request is successful, and the PoolState opaque data was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 3**

A request is made to create the PoolDynamicQuota opaque data. The PoolDynamicQuota XML blob is supplied whole. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolDynamicQuota"/><op value="="/><cdata>
<![CDATA[<?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>2015-03-09T11:20:32</activationdatetime>
    <expirationdatetime>2015-04-09T11:20:32</expirationdatetime>
    <InterimReportingInterval>100</InterimReportingInterval>
    <Duration>10</Duration>
  </DynamicQuota>
</definition>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>

```

**Response 3**

The request is successful, and the PoolDynamicQuota opaque data was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 4**

A request is made to create the PoolLocation opaque data. The PoolLocation XML blob is supplied whole. PoolLocation is not a valid opaque data type. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolLocation"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<location>
  <town>Montreal</town>
  <province>Quebec</province>
  <country>Canada</country>
</location>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>

```

**Response 4**

The request fails. The error value indicates the opaque data type is invalid, and the affected rows are 0. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="70015" affected="0"/>
</req>

```

**Request 5**

A request is made to create the PoolQuota opaque data. The PoolQuota XML blob is supplied whole. The Pool has an associated PoolQuota. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolQuota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="500000"/></expr>
  </where>
</req>

```

**Response 5**

The request fails. The error value indicates the PoolQuota exists, and the affected rows are 0. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="70028" affected="0"/>
</req>

```



**Request 6**

A request is made to create the PoolQuota opaque data. The PoolQuota XML blob is supplied whole. The Pool has an associated PoolQuota. The request includes the odk attribute, indicating that the PoolQuota is updated if it exists. The request is not required in the response.

```
<req name="insert" resonly="y" odk="yes">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolQuota"/><op value="="/><cdata>
<![CDATA[<?xml version="1.0" encoding="UTF-8"?>
<usage>
  <version>3</version>
  <quota name="AggregateLimit">
    <cid>9223372036854775807</cid>
    <time>3422</time>
    <totalVolume>1000</totalVolume>
    <inputVolume>980</inputVolume>
    <outputVolume>20</outputVolume>
    <serviceSpecific>12</serviceSpecific>
    <nextResetTime>2010-05-22T00:00:00-05:00</nextResetTime>
  </quota>
</usage>
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="600000"/></expr>
  </where>
</req>
```

**Response 6**

The request is successful, and the PoolQuota opaque data was updated. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**7.3.2 Get Opaque Data****Description**

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

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

**Prerequisites**

- A pool with the key of the poolId supplied must exist.
- The poolOpaqueDataType must reference a valid Entity in the Interface Entity Map table in the SEC.
- The pool opaque data of the poolOpaqueDataType must exist for the pool.

**Request**

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <select>
    <expr><attr name="poolOpaqueDataType"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Value is either PoolQuota, PoolState, or PoolDynamicQuota

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

Values: 1 to 99999999999999999999

### Response Content

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>
        <![CDATA[cdataRowValue]]>
      </rv>
    </row>
  </rset>
]
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 is expected for success
- *cdataRowValue*: Contents of the XML data blob

### NOTES

- The *<rset>* (row set) element is optional. It is only present if the request was successful. Only a single *<row>* element is returned, with a single *<rv>* (row value) element containing an XML CDATA construct containing the requested pool opaque data (XML blob).
- If the PSO feature is enabled, and the UDR is the non pool host UDR for the specified pool, then empty entity data returns, as pool entity data is only stored on the pool host UDR.

**Error Codes 324: Get Opaque Data**

| Error Code          | Description  |
|---------------------|--|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found   |
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found |

**Get Opaque Data Examples****Request 1**

A request is made to get the PoolQuota opaque data for a pool. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Pool"/>
  <select>
    <expr><attr name="PoolQuota"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the PoolQuota opaque data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="AggregateLimit">
              <cid>9223372036854775807</cid>
              <time>3422</time>
              <totalVolume>1000</totalVolume>
              <inputVolume>980</inputVolume>
              <outputVolume>20</outputVolume>
              <serviceSpecific>12</serviceSpecific>
              <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
            </quota>
          </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

**Request 2**

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. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="Pool"/>
  <select>
    <expr><attr name="PoolQuota"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

## Response 2

The request is successful, and empty data is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
  </rset>
</req>

```

### 7.3.3 Update Opaque Data

#### Description

This operation updates the pool opaque data of the specified poolOpaqueDataType for the pool identified by the poolId in the request.

The pool opaque data is provided in the request in a <cdata> construct. The existing pool opaque data is completely replaced by the data supplied in the request.

#### NOTES

- The opaque data provided in an XML blob is always checked to be valid XML. If the entity is defined as transparent in the SEC, then the XML blob is fully validated against the definition in the SEC. If either validation check fails, then the request is rejected.
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

#### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The poolOpaqueDataType must reference a valid Entity in the Interface Entity Map table in the SEC.

#### Request

```

<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="poolOpaqueDataType"/><op value="="/><cdata>
<![CDATA[
cdataFieldValue
]]></cdata></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>

```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Value is either PoolQuota, PoolState, or PoolDynamicQuota

- *cdataFieldValue*: Contents of the XML data blob
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length

Values: 1 to 99999999999999999999

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
 Values:A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 is expected for success

**Error Codes 335: Update Opaque Data**

| Error Code            | Description   |
|-----------------------|---|
| INTF_ENTY_NOT_FOUND   | Interface Entity Not Found  |
| FIELD_VAL_INVALID     | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION  | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| INVAL_REPEATABLE_ELEM | Invalid Repeatable Element  |
| INVALID_XML           | Invalid Input XML   |
| FIELD_UNDEFINED       | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_NOT_FOUND         | Key Not Found. A subscriber or pool with the given key cannot be found  |

**Update Opaque Data Examples**

**Request 1**

A request is made to update the PoolState opaque data. The PoolState XML blob is supplied whole. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolState"/><op value=""/><cdata>
<![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>
]]></cdata></expr>
</set>
<where>
  <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
</where>
</req>

```

### Response 1

The request is successful, and the PoolState opaque data was updated. The original request is not included.

```

<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>

```

## 7.3.4 Delete Opaque Data

### Description

This operation deletes the opaque data of the specified poolOpaqueDataType for the pool identified by the poolId in the request.

Only one opaque data type can be deleted per request.

### NOTES

- The deletion of a non-existent opaque data type (but that is defined in the SEC) is not considered as an error.
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The poolOpaqueDataType must reference a valid Entity in the Interface Entity Map table in the SEC.

### Request

```

<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="Pool"/>
  <set>
    <expr><attr name="opaqueDataType"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>

```

- *keyName*: A user defined field identified as key for the subscriber
- *keyValue*: A key value identifying the subscriber
- *poolOpaqueDataType*: A user defined type/name for the pool opaque data

Value is either PoolQuota, PoolState, or PoolDynamicQuota

**NOTE:** The data is deleted by setting an empty field value, and also specifying the attribute `isnull="y"`

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

Values: 1 to 99999999999999999999

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values:A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools deleted. A value of 1 is expected for success

**Error Codes 346: Delete Opaque Data**

| Error Code          | Description  |
|---------------------|--|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found   |
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found |

**Delete Opaque Data Examples**

**Request 1**

A request is made to delete the PoolDynamicQuota opaque data. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolDynamicQuota"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the PoolDynamicQuota opaque data was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to delete the PoolState opaque data. PoolState is a valid opaque data type, but the subscriber does not have this opaque data type. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="Pool"/>
  <set>
    <expr><attr name="PoolState"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, because an error is not returned if the pool does not have the opaque data type.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**7.4 Pool Data Row Commands**

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

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

**NOTE:** Pool data row commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an OPER\_NOT\_ALLOWED error being returned.

**Table 23: Summary of Pool Data Row Commands**

| Command    | Description  | Keys  | Command Syntax  |
|------------|--|---|---|
| Create Row | Insert data row into transparent data of the specified type.   | (PoolID and Row Identifier) Or (PoolID, Row Identifier and Instance Identifier) | <pre>&lt;req name="insert"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...</pre>   |
| Get Row    | Retrieve data row from transparent data of the specified type. |   | <pre>&lt;req name="select"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;       &lt;/expr&gt; ...   ]</pre> |



| Command    | Description  | Keys | Command Syntax  |
|------------|--|------|---|
| Delete Row | Delete data row in transparent data of the specified type. |      | <pre> &lt;req name="delete"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;       &lt;/expr&gt; ...     ] </pre> |

### 7.4.1 Create Row

#### Description

This operation creates a data row for the pool identified by the poolId.

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

#### NOTES

- The *rowIdValue* is case-sensitive. If a row existed 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.
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

#### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.

#### Request

**NOTE:** This command allows 2 different formats. One with the poolId in the <set> element, and another with the poolId in a <where> element.

##### Format 1

```

<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="PoolID"/><value val="poolId"/></expr>
    <expr><attr name="rowIdName"/><value val="rowIdValue"/></expr>
  [
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
  </set>
</req>

```

##### Format 2

```

<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="rowIdName"/><value val="rowIdValue"/></expr>
  [

```

```

    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</set>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o **y**—Provides the result only, does not include the original request
  - o **n**—Includes the original request in the response (default)
- **id:** (Optional) Transaction ID value in the request, and passed back in the response
- Values: 1 to 4294967295
- **odk:** (Optional) Indicates that the insert request is converted to an update if the data row for the specified entity exists
  - **entityName:** A user defined entity type/name for the transparent data
    - o Value is `PoolQuotaEntity` for the PoolQuota transparent data
    - o Value is `PoolDynamicQuotaEntity` for the PoolDynamicQuota transparent data
  - **poolId:** PoolID value of the pool. Numeric value, 1 to 22 digits in length
- Values: 1 to 99999999999999999999
- **rowIdName:** Name of the XML attribute that identifies the row in the data blob
    - o Value is name for PoolQuota transparent data
    - o Value is name for PoolDynamicQuota transparent data
  - **rowIdValue:** The row name value that identifies the row in the data blob
  - **fieldNameX:** A user defined field in the data row
  - **fieldValueX:** Corresponding field value assigned to *fieldNameX*

**NOTE:** For multi-value fields, the value can contain a comma separated list of values on a single line. For example, a,b,c

## NOTES

- Rows that have the same *rowIdName/rowIdValue* are permitted. Where duplicate rows occur, and an additional field is set to define uniqueness (such as `<cid>` in the PoolQuota entity) validation is not performed by UDR to ensure uniqueness. Unique values must be supplied by the provisioning client otherwise operations (such as updating an existing row) may fail if more than one matching row is found.
- If the `odk="yes"` attribute is set (implying that an update be made if the row exists), then if multiple rows exist for the specified *rowIdName/rowIdValue*, then the request fails because it is not known which of the multiple rows to update.

## Response

```

<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]

```

```
<res error="error" affected="affected"/>
</req>
```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request  
Values: A string with 1 to 4096 characters
- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of pools updated. A value of 1 is expected for success

#### Error Codes 357: Create Row

| Error Code            | Description   |
|-----------------------|---|
| INTF_ENTY_NOT_FOUND   | Interface Entity Not Found  |
| FIELD_VAL_INVALID     | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC  |
| OCC_CONSTR_VIOLATION  | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field   |
| INVAL_REPEATABLE_ELEM | Invalid Repeatable Element  |
| INVALID_SOAP_XML      | Invalid SOAP XML  |
| FIELD_UNDEFINED       | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_NOT_FOUND         | Key Not Found. A subscriber or pool with the given key cannot be found  |
| MULTIPLE_ROWS_FOUND   | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria<br><b>NOTE:</b> Only returned when the <code>odk="yes"</code> attribute is supplied, and duplicate candidate rows to update are found |

#### Examples

##### Request 1

A request is made to create a data row in the PoolQuotaEntity (PoolQuota) data. The data row identifier field is name, and the value is Q1. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="name"/><value val="Q1"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
```

```

</set>
</req>

```

### Response 1

The request is successful, and the data row Q1 was created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

### Request 2

A request is made to create a data row in the PoolQuotaEntity (PoolQuota) data. PoolQuota is a valid opaque data type, but the pool does not have this opaque data type. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="300000"/></expr>
    <expr><attr name="name"/><value val="Q2"/></expr>
    <expr><attr name="cid"/><value val="9223372036854999999"/></expr>
    <expr><attr name="time"/><value val="10:10"/></expr>
    <expr><attr name="totalVolume"/><value val="55000"/></expr>
    <expr><attr name="inputVolume"/><value val="50000"/></expr>
    <expr><attr name="outputVolume"/><value val="5000"/></expr>
    <expr><attr name="serviceSpecific"/><value val="serviceSpecific"/></expr>
    <expr><attr name="nextResetTime"/>
      <value val="1961-12-15T09:04:03"/></expr>
  </set>
</req>

```

### Response 2

The request is successful, and the data row as well as the PoolQuota entity is created. The original request is not included.

```

<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>

```

### Request 3

A request is made to create a data row in the PoolDynamicQuotaEntity (PoolDynamicQuota) data. The PoolID key is supplied, which references the pool. The data row identifier field is name, and the value is PDQ1. The request is not required in the response.

```

<req name="insert" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="400000"/></expr>
    <expr><attr name="name"/><value val="PDQ1"/></expr>
    <expr><attr name="Type"/><value val="top-up"/></expr>
    <expr><attr name="InstanceId"/><value val="15678"/></expr>
    <expr><attr name="Priority"/><value val="4"/></expr>
    <expr><attr name="InitialTime"/><value val="135"/></expr>
    <expr><attr name="InitialTotalVolume"/><value val="2000"/></expr>
    <expr><attr name="InitialInputVolume"/><value val="1500"/></expr>
    <expr><attr name="InitialOutputVolume"/><value val="500"/></expr>
    <expr><attr name="InitialServiceSpecific"/><value val="4"/></expr>
    <expr><attr name="activationdatettime"/><value val="2015-05-22T00:00:00-05:00"/></expr>
    <expr><attr name="expirationdatettime"/><value val="2015-05-29T00:00:00-05:00"/></expr>
    <expr><attr name="InterimReportingInterval"/><value val="100"/></expr>
    <expr><attr name="Duration"/><value val="10"/></expr>
  </set>
</req>

```

**Response 3**

The request is successful, and the data row was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**7.4.2 Get Row****Description**

This operation retrieves a data rows for the pool identified by the poolId.

The data row identifier field is specified in rowIdName, and the row identifier value is specified in rowIdValue. An additional field can be specified to indicate a unique row in instanceFieldName/instanceFieldValue.

All data rows that match the requested rowIdName/rowIdValue and instanceFieldName/instanceFieldValue (if specified) are returned.

**NOTES**

- The *rowIdValue* is case-sensitive. If a row existed called DayPass, then an attempt to retrieve a row called DayPass is successful, but an attempt to retrieve a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to retrieve a row with a field with the value Data is successful, but an attempt to retrieve a row with a field with the value DATA fails.

**Prerequisites**

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- The transparent entity must exist for the pool.

**Request**

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
    <expr><attr name="rowIdName"/><op value="="/>
      <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
  - o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response  
values: 1 to 4294967295
  - *entityName*: A user defined entity type/name for the transparent data
    - o Value is PoolQuotaEntity for the PoolQuota transparent data

- o Value is `PoolDynamicQuotaEntity` for the `PoolDynamicQuota` transparent data
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length  
Values: 1 to 99999999999999999999
- *rowIdName*: Name of the XML attribute that identifies the row in the data blob
  - o Value is name for `PoolQuota` transparent data
  - o Value is name for `PoolDynamicQuota` 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
  - o Value is `cid` or `Type` for the `PoolQuota` transparent data
  - o Value is `InstanceId` or `Type` for the `PoolDynamicQuota` transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

## Response

```

<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
<
      <rv>
        <![CDATA[cdataRowValue1]]>
      </rv>
|
      <rv null="y"/>
>
    </row>
[
    <row>
      <rv>
        <![CDATA[cdataRowValue2]]>
      </rv>
    </row>
    :
    <row>
      <rv>
        <![CDATA[cdataRowValueN]]>
      </rv>
    </row>
]
  </rset>
]
</req>

```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 or more is expected for success, whether or not a row was found
- *cdataRowValueN*: Contents of the XML data blob containing one requested/matching data row

**NOTES**

- The <rset> (row set) element is optional. It is only present if the request was successful. One <row> element is returned per matching row, with a single <rv> (row value) element containing an XML CDATA construct containing a single requested data row instance.
- If the transparent entity exists, but the row value was not found, then the <rv> (row value) indicates the row does not exist by containing the value <rv null="y"/>.
- If the PSO feature is enabled, and the UDR is the non pool host UDR for the specified pool, then empty entity data returns, as pool entity data is only stored on the pool host UDR.

**Error Codes 368: Get Row**

| Error Code          | Description  |
|---------------------|--|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found   |
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found |
| REG_DATA_NOT_FOUND  | Register Data Not Found  |

**Get Row Examples****Request 1**

A request is made to get the Q1 data row from the PoolQuota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the PoolQuota data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?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>]]>
      </rv>
    </row>
  </rset>
</req>
```

**Request 2**

A request is made to get the Weekend data row from the PoolQuota data. The PoolQuota data contains two rows called Weekend. One with cid of 11223344, the other with a cid of 99887766. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, and 2 PoolQuota data rows are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="Weekend">
              <cid>11223344</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>]]>
      </rv>
    </row>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="Weekend">
              <cid>99887766</cid>
              <time>1232</time>
              <totalVolume>2000</totalVolume>
              <inputVolume>440</inputVolume>
              <outputVolume>8220</outputVolume>
              <serviceSpecific>99</serviceSpecific>
              <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
            </quota>
          </usage>]]>
      </rv>
    </row>
  </rset>
</req>
```

**Request 3**

A request is made to get the Weekend data row from the PoolQuota data, with the cid value of 11223344. The PoolQuota data contains two rows called Weekend. One with cid of 11223344, the other with a cid of 99887766. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```



```

</where>
</req>

```

### Response 3

The request is successful, and the PoolQuota data with a cid of 11223344 is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="Weekend">
              <cid>11223344</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>]]>
      </rv>
    </row>
  </rset>
</req>

```

### Request 4

A request is made to get the LateNight data row from the PoolQuota data, with the cid value of 11223344. The PoolQuota data contains four rows called LateNight. Two with cid of 11223344, one with a cid of 99887766, and one with a cid of 55556666. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="LateNight"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>

```

### Response 4

The request is successful, and the 2 PoolQuota data rows with a cid of 11223344 are returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <usage>
            <version>3</version>
            <quota name="LateNight">
              <cid>11223344</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>]]>
      </rv>
    </row>
  </rset>
</req>

```

```

</row>
<row>
  <rv>
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
      <usage>
        <version>3</version>
        <quota name="LateNight">
          <cid>11223344</cid>
          <time>1232</time>
          <totalVolume>2000</totalVolume>
          <inputVolume>440</inputVolume>
          <outputVolume>8220</outputVolume>
          <serviceSpecific>99</serviceSpecific>
          <nextResetTime>2011-04-22T00:00:00-05:00</nextResetTime>
        </quota>
      </usage>]]>
  </rv>
</row>
</rset>
</req>

```

**Request 5**

A request is made to get the Weekday data row in the PoolQuota data. The Weekday data row does not exist in the PoolQuota data. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>

```

**Response 5**

The request is successful, and indicates that the requested row does not exist. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
  </rset>
</req>

```

**Request 6**

A request is made to get the Weekday data row in the PoolQuota data. PoolQuota is a valid opaque data type, but the pool does not have this opaque data type. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>

```

**Response 6**

The request fails. The error value indicates the opaque data type is not found, and the affected rows are 0. The original request is not included.

```

<req name="select" resonly="y">
  <res error="70027" affected="0"/>
</req>

```

**Request 7**

A request is made to get the PDQ1 data row from the PoolDynamicQuota data, with the InstanceId value of 11223344. The PoolDynamicQuota data contains four rows called PDQ1. Two with InstanceId of 11223344, one with an InstanceId of 99887766, and one with an InstanceId of 55556677. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
      <value val="4000000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="PDQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

**Response 7**

The request is successful, and the 2 PoolDynamicQuota data rows with an InstanceId of 11223344 are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <definition>
            <version>1</version>
            <DynamicQuota name="PDQ1">
              <Type>top-up</Type>
              <InstanceId>11223344</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>]]>
      </rv>
    </row>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <definition>
            <version>1</version>
            <DynamicQuota name="DQ1">
              <Type>pass</Type>
              <InstanceId>11223344</InstanceId>
              <Priority>2</Priority>
              <InitialTime>135</InitialTime>
              <InitialTotalVolume>1000</InitialTotalVolume>
              <InitialInputVolume>500</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>]]>
      </rv>
    </row>
  </rset>
```

```
</req>
```

### Request 8

A request is made to get the Weekday data row in the PoolQuota data for a PSO pool, where the UDR instance receiving the request is not the pool host UDR for the pool in the request. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekday"/></expr>
  </where>
</req>
```

### Response 8

The request is successful, and empty data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

## 7.4.3 Delete Row

### Description

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

The data row identifier field is specified in rowIdName, and the row identifier value is specified in rowIdValue. An additional field can be specified to indicate a unique row in instanceFieldName/instanceFieldValue.

If more than one row matches the requested rowIdName/rowIdValue and instanceFieldName/instanceFieldValue (if specified), then all matching rows are deleted.

### NOTES

- The *rowIdValue* is case-sensitive. If a row existed called DayPass, then an attempt to delete a row called DayPass is successful, but an attempt to delete a row called DAYPASS fails.
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to delete a row with a field with the value Data is successful, but an attempt to delete a row with a field with the value DATA fails.
- The deletion of a non-existent data row is not considered an error.
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- The transparent entity must exist for the pool.

### Request

```
<req name="delete" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
```

```

<where>
  <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  <expr><attr name="rowIdName"/><op value="="/>
    <value val="rowIdValue"/></expr>
[
  <expr><attr name="instanceFieldName"/><op value="="/>
    <value val="instanceFieldValue"/></expr>
]
</where>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- **id:** (Optional) Transaction ID value in the request, and passed back in the response  
Values: 1 to 4294967295
- **entityName:** A user defined entity type/name for the transparent data
  - o Value is PoolQuotaEntity for the PoolQuota transparent data
  - o Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data
- **poolId:** PoolID value of the pool. Numeric value, 1 to 22 digits in length  
Values: 1 to 9999999999999999999999
- **rowIdName:** Name of the XML attribute that identifies the row in the data blob
  - o Value is name for PoolQuota transparent data
  - o Value is name for PoolDynamicQuota 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
  - o Value is cid or type for the PoolQuota transparent data
  - o Value is InstanceId or Type for the PoolDynamicQuota transparent data
- **instanceFieldValue:** Corresponding field value assigned to *instanceFieldName*

## Response

```

<req name="delete" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the resonly="y" attribute is set in the original request  
Values: A string with 1 to 4096 characters
- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** A value of 1 indicates the rows existed, or that the row did not exist

**Error Codes 379: Delete Row**

| Error Code          | Description  |
|---------------------|--|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found   |
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found |
| REG_DATA_NOT_FOUND  | Register Data Not Found  |

**Delete Row Examples****Request 1**

A request is made to delete the Q1 data row in the PoolQuota data. The Q1 data row exists in the PoolQuota data, and is there is only one row called Q1. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

**Response 1**

The request is successful, and the data row in the PoolQuota data was deleted. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to delete the Weekend data row in the PoolQuota data. The Weekend data row does not exist in the PoolQuota data. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, because an error is not returned if the data row is not present. The original request is not included.

```
<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 3**

A request is made to delete the Q3 data row in the PoolQuota data. The PoolQuota data contains two rows called Q3. The request is not required in the response.

```
<req name="delete" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
```

```

    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
  </where>
</req>

```

**Response 3**

The request is successful, and the data row in the PoolQuota data was deleted. The original request is not included.

```

<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 4**

A request is made to delete the Q4 data row from the PoolQuota data, with the cid value of 11223344. The PoolQuota data contains two rows called Q4. One with cid of 11223344, the other with a cid of 99887766. The request is not required in the response.

```

<req name="delete" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q4"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>

```

**Response 4**

The request is successful, and the data row in the PoolQuota data was deleted. The original request is not included.

```

<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 5**

A request is made to delete the PDQ1 data row from the PoolDynamicQuota data, with the Type value of pass. The PoolDynamicQuota data contains two rows called PDQ1. One with a Type of pass the other with a Type of quota. The request is not required in the response.

```

<req name="delete" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
      <value val="500000"/></expr>
    <expr><attr name="name"/><op value="="/>
      <value val="PDQ1"/></expr>
    <expr><attr name="Type"/><op value="="/>
      <value val="pass"/></expr>
  </where>
</req>

```

**Response 5**

The request is successful, and the data row in the PoolDynamicQuota data was deleted that matched the Type. The original request is not included.

```

<req name="delete" resonly="y">
  <res error="0" affected="1"/>
</req>

```

## 7.5 Pool Data Row Field Commands

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

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

**NOTE:** Pool data row field commands may only be performed on entities defined as transparent in the SEC. Attempting to perform a command on an entity defined as opaque results in an OPER\_NOT\_ALLOWED error being returned.

**Table 24: Summary of Pool Data Row Field Commands**

| Command          | Description                                | Keys  | Command Syntax  |
|------------------|--|---|---|
| Get Row Field    | Retrieve values for the specified fields   | (PoolID and Row Identifier and Field name)<br>Or<br>(PoolID and Row Identifier, Instance Identifier and Field name) | <pre>&lt;req name="select"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...</pre>   |
| Update Row Field | Update fields to the specified values      |   | <pre>&lt;req name="update"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;       &lt;/expr&gt; ...   ]</pre> |
| Delete Row Field | Delete all values for the specified fields |   | <pre>&lt;req name="update"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="rowIdName"&gt;       &lt;value val="rowIdValue"&gt;     &lt;/expr&gt; ...   [     &lt;expr&gt;       &lt;attr name="instanceFieldName"&gt;         &lt;value val="instanceFieldValue"&gt;       &lt;/expr&gt; ...   ]</pre> |

### 7.5.1 Get Row Field

#### Description

This operation retrieves a fields in a data row for the pool identified by the poolId.

The data row identifier field is specified in rowIdName, and the row identifier value is specified in rowIdValue. An additional field can be specified to indicate a unique row in instanceFieldName/instanceFieldValue. The field names are specified in fieldNameX.

All data rows that match the requested rowIdName/rowIdValue and instanceFieldName/instanceFieldValue (if specified) are returned.

#### NOTES

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



- The *rowIdValue* is case-sensitive. If a row existed 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.
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to delete a row with a field with the value Data is successful, but an attempt to delete a row with a field with the value DATA fails.
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier/instance in the transparent data must exist for the subscriber.
- The field names specified must be valid fields for the Entity as defined in the SEC.

### Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <select>
    <expr><attr name="fieldName1"/><value val="[fieldName1]"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="[fieldName2]"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="[fieldNameN]"/></expr>
  ]
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
    <expr><attr name="rowIdName"/><op value="="/>
      <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *entityName*: A user defined entity type/name for the transparent data
  - o Value is PoolQuotaEntity for the PoolQuota transparent data
  - o Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data

- *fieldNameX*: A user defined field in the data row
- *fieldValueX*: (Optional) Corresponding field value assigned to *fieldNameX*

**NOTE:** For multi-value fields, the value can contain a comma separated list of values

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

Values: 1 to 9999999999999999999999

- *rowIdName*: Name of the XML attribute that identifies the row in the data blob
  - o Value is name for PoolQuota transparent data
  - o Value is name for PoolDynamicQuota 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
  - o Value is cid or type for the PoolQuota transparent data
  - o Value is InstanceId or type for the PoolDynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

## Response

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv> >
    [
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv> >
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv> >
    ]
  </row>
  [
    <row>
      <rv>rowValue1</rv> | <rv null="y"> | <rv></rv> >
    [
      <rv>rowValue2</rv> | <rv null="y"> | <rv></rv> >
      :
      <rv>rowValueN</rv> | <rv null="y"> | <rv></rv> >
    ]
  </row>
  :
  <row>
    <rv>rowValue1</rv> | <rv null="y"> | <rv></rv> >
  [
    <rv>rowValue2</rv> | <rv null="y"> | <rv></rv> >
    :
    <rv>rowValueN</rv> | <rv null="y"> | <rv></rv> >
  ]
  </row>
]
</rset>
]
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 is expected if the specified row exists (whether or not the field was found). A value of 0 is expected if the row does not exist
- *rowValue*: The value of the requested field

**NOTE:** for multi-value fields, the value contains a comma separated list of values on a single line. For example, a,b,c

## NOTES

- The <rset> (row set) element is optional. It is only present if the request was successful. One <row> element is returned per matching row. One <rv> (row value) element exists for every fieldNameX supplied in the original request. The <rv> elements are ordered the same as the fieldNameX fields were specified in the original request. If the field is valid, but not present in the entity, this is indicated with <rv null="y">. If the field is present, but has an empty value, this is indicated with <rv></rv>.
- If the PSO feature is enabled, and the UDR is the non pool host UDR for the specified pool, then empty entity data returns, as pool entity data is only stored on the pool host UDR.

## Error Codes 40: Get Row Field

| Error Code          | Description   |
|---------------------|---|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found  |
| FIELD_UNDEFINED     | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found                      |
| REG_DATA_NOT_FOUND  | Register Data Not Found   |
| ROW_NOT_FOUND       | Data row specified is not found   |

## Examples

### Request 1

A request is made to get the inputVolume field in the Q1 data row of the PoolQuota data. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <select>
    <expr><attr name="inputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

### Response 1

The request is successful, and the requested field value 980 is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>980</rv>
    </row>
  </rset>
</req>
```

**Request 2**

A request is made to get the outputVolume and cid fields in the Q2 data row of the PoolQuota data. The PoolQuota data contains two rows called Q2. One with cid of 11223344, the other with a cid of 99887766. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></expr>
    <expr><attr name="cid"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q2"/></expr>
  </where>
</req>
```

**Response 2**

The request is successful, and the requested field values are returned from each row. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>220</rv>
      <rv>11223344</rv>
    </row>
    <row>
      <rv>1050</rv>
      <rv>99887766</rv>
    </row>
  </rset>
</req>
```

**Request 3**

A request is made to get the outputVolume field in the Q3 data row of the PoolQuota data, with the cid value of 11223344. The PoolQuota data contains two rows called Q3. One with cid of 11223344, the other with a cid of 99887766. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <select>
    <expr><attr name="outputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

**Response 3**

The request is successful, and the requested field value 4000 is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>4000</rv>
    </row>
  </rset>
</req>
```

**Request 4**

A request is made to get the InstanceId, InitialTotalVolume and InitialInputVolume fields in the PDQ3 data row of the PoolDynamicQuota data, with the InstanceId value of 15678. The PoolDynamicQuota data contains two rows called PDQ3. One with InstanceId of 15570, the other with an InstanceId of 15678. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <select>
    <expr><attr name="InstanceId"/></expr>
    <expr><attr name="InitialTotalVolume"/></expr>
    <expr><attr name="InitialInputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
      <value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/>
      <value val="PDQ3"/></expr>
    <expr><attr name="InstanceId"/><op value="="/>
      <value val="15678"/></expr>
  </where>
</req>
```

**Response 4**

The request is successful, and the requested field values are returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>15678</rv>
      <rv>2000</rv>
      <rv>1500</rv>
    </row>
  </rset>
</req>
```

**Request 5**

A request is made to get the inputVolume field in the Q1 data row of the PoolQuota data for a PSO pool, where the UDR instance receiving the request is not the pool host UDR for the pool in the request. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <select>
    <expr><attr name="inputVolume"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

**Response 5**

The request is successful, and empty data is returned. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

## 7.5.2 Update Row Field

### Description

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

The data row identifier field is specified in rowIdName, and the row identifier value is specified in rowIdValue. An additional field can be specified to indicate a unique row in instanceFieldName/instanceFieldValue. The field names are specified in fieldNameX. If the specified fields are valid, but do not exist, they are created.

If more than one row matches the requested rowIdName/rowIdValue and instanceFieldName/instanceFieldValue (if specified), then the update request fails.

### NOTES

- If the specified row does not exist, the request fails.
- If the requested fields are valid, but not present, they are created.
- The *rowIdValue* is case-sensitive. If a row existed 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.
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to delete a row with a field with the value Data is successful, but an attempt to delete a row with a field with the value DATA fails.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- A data row with the given identifier/instance in the transparent data must exist for the pool.
- The field names specified must be valid fields for the Entity as defined in the SEC.

### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
    <expr><attr name="rowIdName"/><op value="="/>
      <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request

- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response  
Values: 1 to 4294967295
- *entityName*: A user defined entity type/name for the transparent data
  - o Value is PoolQuotaEntity for the PoolQuota transparent data
  - o Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data
- *fieldNameX*: A user defined field in the data row
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*  
**NOTE:** For multi-value fields, the value can contain a comma separated list of values on a single line. For example, a,b,c
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length  
Values: 1 to 9999999999999999999999
- *rowIdName*: Name of the XML attribute that identifies the row in the data blob
  - o Value is name for PoolQuota transparent data
  - o Value is name for PoolDynamicQuota 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
  - o Value is cid or type for the PoolQuota transparent data
  - o Value is InstanceId or Type for the PoolDynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the resonly="y" attribute is set in the original request  
Values:A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 is expected for success

**Error Codes 381: Update Row Field**

| Error Code          | Description  |
|---------------------|--|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found   |
| FIELD_VAL_INVALID   | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC |

| Error Code          | Description   |
|---------------------|---|
| FIELD_UNDEFINED     | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC                           |
| FIELD_NOT_UPDATABLE | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND  | Register Data Not Found   |
| ROW_NOT_FOUND       | Data row specified is not found   |
| MULTIPLE_ROWS_FOUND | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria |

### Update Row Field Examples

#### Request 1

A request is made to update the inputVolume field in the Q1 data row of the PoolQuota data. The Q1 data row exists in the PoolQuota data, and is there is only one row called Q1. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

#### Response 1

The request is successful, and the field in the data row in the PoolQuota data was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### Request 2

A request is made to update the cid field in the Q1 data row in the PoolQuota data. The Q1 data row exists in the PoolQuota data, and is there is only one row called Q1. The cid field is not allowed to be updated. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="cid"/><value val="11223344"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Weekend"/></expr>
  </where>
</req>
```



**Response 2**

The request fails. The error value indicates the cid field cannot be updated, and the affected rows are 0. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70016" affected="0"/>
</req>
```

**Request 3**

A request is made to update the outputVolume field in the Q6 data row of the PoolQuota data. The Q6 data row exists in the PoolQuota data, but there are two rows called Q6. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q6"/></expr>
  </where>
</req>
```

**Response 3**

The request fails because there was more than one row called Q6. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

**Request 4**

A request is made to update the InitialTotalVolume and InitialInputVolume fields in the PDQ1 data row of the PoolDynamicQuota data where InstanceId is 15678. The PDQ1 data row exists in the PoolDynamicQuota data, but there are two rows called PDQ1 one with InstanceId of 15570, the other with an InstanceId of 15678. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <set>
    <expr><attr name="InitialTotalVolume"/><value val="1000"/></expr>
    <expr><attr name="InitialInputVolume"/><value val="1000"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
      <value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/>
      <value val="PDQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/>
      <value val="15678"/></expr>
  </where>
</req>
```

**Response 4**

The request is successful, and the field in the data row in the PoolDynamicQuota data was updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

### 7.5.3 Delete Row Field

#### Description

This operation deletes a fields in a data row for the pool identified by the poolId.

The data row identifier field is specified in rowIdName, and the row identifier value is specified in rowIdValue. An additional field can be specified to indicate a unique row in instanceFieldName/instanceFieldValue. The field names are specified in fieldNameX.

If more than one row matches the requested rowIdName/rowIdValue and instanceFieldName/instanceFieldValue (if specified), then the delete request fails.

#### NOTES

- If the specified row does not exist, the request fails. If the specified row exists, but the field does not exist, this is not treated as an error, and row/field data is not deleted.
- The *rowIdValue* is case-sensitive. If a row existed 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.
- The *instanceFieldValue* is case-sensitive. If a field contained the value Data, then an attempt to delete a field in a row with a field with the value Data is successful, but an attempt to delete a field in a row with a field with the value DATA fails.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

#### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- At least one data row with the given identifier/instance in the transparent data must exist for the pool.
- The field names specified must be valid fields for the Entity as defined in the SEC.

#### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><op value="="/>
      <value val="" isnull="y"/></expr>
  [
    <expr><attr name="fieldName2"/><op value="="/>
      <value val="" isnull="y"/></expr>
    :
    <expr><attr name="fieldNameN"/><op value="="/>
      <value val="" isnull="y"/></expr>
  ]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
    <expr><attr name="rowIdName"/><op value="="/>
      <value val="rowIdValue"/></expr>
  [
    <expr><attr name="instanceFieldName"/><op value="="/>
      <value val="instanceFieldValue"/></expr>
  ]
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *entityName*: A user defined entity type/name for the transparent data
  - o Value is PoolQuotaEntity for the PoolQuota transparent data
  - o Value is PoolDynamicQuotaEntity for the PoolDynamicQuota transparent data
- *fieldNameX*: A user defined field in the data row
- *fieldValueX*: Corresponding field value assigned to *fieldNameX*

**NOTE:** For multi-value fields, the value can contain a comma separated list of values on a single line. For example, a,b,c

- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length  
Values: 1 to 99999999999999999999
- *rowIdName*: Name of the XML attribute that identifies the row in the data blob
  - o Value is name for PoolQuota transparent data
  - o Value is name for PoolDynamicQuota 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
  - o Value is cid or Type for the PoolQuota transparent data
  - o Value is InstanceId or Type for the PoolDynamicQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the resonly="y" attribute is set in the original request  
Values:A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 indicates the row existed and the field was deleted. A value of 0 indicates the field did not exist

**Error Codes 392: Delete Row Field**

| Error Code          | Description                |
|---------------------|----------------------------|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found |

| Error Code           | Description   |
|----------------------|---|
| OCC_CONSTR_VIOLATION | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_UNDEFINED      | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE  | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND        | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND   | Register Data Not Found   |
| ROW_NOT_FOUND        | Data row specified is not found   |
| MULTIPLE_ROWS_FOUND  | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria                   |

### Delete Row Field Examples

#### Request 1

A request is made to delete the inputVolume field in the Q1 data row of the PoolQuota data. The Q1 data row exists in the PoolQuota data, and is there is only one row called Q1. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q1"/></expr>
  </where>
</req>
```

#### Response 1

The request is successful, and the field in the data row was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### Request 2

A request is made to delete the outputVolume field in the Q3 data row of the PoolQuota data. The PoolQuota data contains two rows called Q3. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q3"/></expr>
  </where>
</req>
```

**Response 2**

The request fails, because there are two PoolQuota rows called Q3. The original request is not included.

```
<req name="update" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

**Request 3**

A request is made to update delete the outputVolume field in the Q4 data row of the PoolQuota data with the cid 11223344. The Q4 data row exists in the PoolQuota data, and is there are two rows called Q4, one with cid 11223344 and one with cid 99887766. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="outputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="300000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="Q4"/></expr>
    <expr><attr name="cid"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

**Response 3**

The request is successful, and the outputVolume field in the Q4 data row in the PoolQuota data was deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 4**

A request is made to delete the InitialInputVolume field in the PDQ1 data row of the PoolDynamicQuota data with the InstanceId 11223344. The PDQ1 data row exists in the PoolDynamicQuota data, and is there are two rows called PDQ1, one with InstanceId 11223344 and one with InstanceId 99887766. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolDynamicQuotaEntity"/>
  <set>
    <expr><attr name="InitialInputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
      <value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="PDQ1"/></expr>
    <expr><attr name="InstanceId"/><op value="="/><value val="11223344"/></expr>
  </where>
</req>
```

**Response 4**

The request is successful, and the InitialInputVolume field in the PDQ1 data row in the PoolDynamicQuota data was deleted for the row with the specified InstanceId. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 5**

A request is made to delete the inputVolume field in the PQ1 data row of the PoolQuota data. The PQ1 data row exists in the PoolQuota data, there is only one row called PQ1 and the inputVolume field does not exist. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolQuotaEntity"/>
  <set>
    <expr><attr name="inputVolume"/><op value="="/>
      <value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/>
      <value val="400000"/></expr>
    <expr><attr name="name"/><op value="="/><value val="PQ1"/></expr>
  </where>
</req>
```

**Response 5**

The request is successful and the original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**7.6 Pool Data Field Commands**

A transparent data entity may contain data that is organized in fields where each field is defined as a name value pair 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>fieldName</name>
  <value>fieldValue</value>
</property>
```

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

**NOTE:** Pool data 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 OPER\_NOT\_ALLOWED error being returned.

**Table 25: Summary of Pool Data Commands**

| Command           | Description   | Keys                  | Command Syntax  |
|-------------------|---|-----------------------|---|
| Create Data Field | Create/update data field in transparent data of the specified type. | PoolID and Field Name | <pre>&lt;req name="insert"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="fieldName"&gt;       &lt;value val="fieldValue"&gt;     &lt;/expr&gt; ...</pre> |
| Get Data Field    | Retrieve data field from transparent data of the specified type.    |                       | <pre>&lt;req name="select"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="fieldName"&gt;       &lt;value val="fieldValue"&gt;     &lt;/expr&gt; ...</pre> |
| Update Data Field | Update data field in transparent data of the specified type.        |                       | <pre>&lt;req name="update"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="fieldName"&gt;       &lt;value val="fieldValue"&gt;     &lt;/expr&gt; ...</pre> |

| Command           | Description  | Keys | Command Syntax  |
|-------------------|--|------|---|
| Delete Data Field | Delete data field in transparent data of the specified type. |      | <pre>&lt;req name="update"&gt;   &lt;ent name="entityName"/&gt;   ... &lt;expr&gt;     &lt;attr name="fieldName"&gt;       &lt;value val="fieldValue"&gt;     &lt;/expr&gt; ...</pre> |

## 7.6.1 Create 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 fieldNameX, and the field value is specified in fieldValueX.

If the specified field does not exist, it is created. If the field does exist, it is updated/replaced only if the optional odk flag is set to yes.

### NOTES

- The *fieldName* is not case-sensitive. If a field existed called mcc, then an attempt to create/update an existing field called MCC is successful.
- If the transparent entity specified in entityName does not exist for the pool, it is created.
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.

### Request

**NOTE:** This command allows 2 different formats. One with the *PoolID* in the <set> element, and another with the *PoolID* in a <where> element.

#### Format 1

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="PoolID"/><value val="poolId"/></expr>
  <expr><attr name="fieldName1"/><value val="fieldValue"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
</set>
</req>
```

#### Format 2

```
<req name="insert" [resonly="resonly"] [id="id"] [odk="yes"]>
  <ent name="entityName"/>
  <set>
  <expr><attr name="fieldName"/><value val="fieldValue"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
```

```
<where>
  <expr><attr name="PoolID"/><value val="poolId"/></expr>
</where>
</set>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *odk*: (Optional) Indicates that the insert request is converted to an update if the field for the specified entity exists

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

Value is PoolStateEntity for the PoolState transparent data

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

Values: 1 to 99999999999999999999

- *fieldNameX*: The requested user defined field in the pool profile.

For the PoolState entity, this corresponds to a property in the entity

- o The *fieldNameX* case is stored exactly as it was sent in the request (This means the original case stored changes if an update is received)

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

**Response**

```
<req name="insert" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values:A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 is expected for success

**Error Codes 403: Create Data Field**

| Error Code          | Description                |
|---------------------|----------------------------|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found |



| Error Code            | Description   |
|-----------------------|---|
| FIELD_VAL_INVALID     | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| OCC_CONSTR_VIOLATION  | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| INVAL_REPEATABLE_ELEM | Invalid Repeatable Element  |
| INVALID_SOAP_XML      | Invalid SOAP XML  |
| FIELD_UNDEFINED       | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| KEY_NOT_FOUND         | Key Not Found. A subscriber or pool with the given key cannot be found  |

### Create 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. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="PoolID"/><value val="100000"/></expr>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

#### Response 1

The request is successful, and the property mcc with value 315 was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### Request 2

A request is made to create a property in the PoolState transparent data for a pool, using the alternate request format. The property name is mcc and the property value is 315. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
  <where>
<expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

#### Response 2

The request is successful, and the property was created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
```

```
</req>
```

### **Request 3**

A request is made to create a property in the PoolState transparent data for a pool. PoolState is a valid data type, but the pool does not have this entity type. The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
<expr><attr name="PoolID"/><value val="100000"/></expr>
  <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

### **Response 3**

The request is successful, and the property as well as the PoolState entity is created. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

### **Request 4**

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 odk attribute is included requesting the data be updated if it exists. The request is not required in the response.

```
<req name="insert" resonly="y" odk="yes">
  <ent name="PoolStateEntity"/>
  <set>
<expr><attr name="PoolID"/><value val="100000"/></expr>
  <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

### **Response 4**

The request is successful, and the existing property was updated. The original request is not included.

```
<req name="insert" resonly="y">
  <res error="0" affected="1"/>
</req>
```

### **Request 5**

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 property exists) The odk attribute is not included (requests the data be updated if it exists.) The request is not required in the response.

```
<req name="insert" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
  <expr><attr name="PoolID"/><value val="100000"/></expr>
  <expr><attr name="mcc"/><value val="315"/></expr>
  </set>
</req>
```

### **Response 5**

The request fails. The error value indicates the mcc property exists, and the affected rows are 0. The original request is not included.

```
<req name="insert" resonly="y">
```

```
<res error="70028" affected="0"/>
</req>
```

## 7.6.2 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 fieldNameX are returned.

If more than one field matches the requested fieldNameX, then all matching fields returns.

The transparent data field is specified in fieldNameX.

### NOTES

- If the specified field does not exist, null="y" is returned.
- The *fieldNameX* is not case-sensitive. If a field existed called mcc, then an attempt to get a field name called MCC is successful.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.
- A field with the given identifier in the transparent data must exist for the pool.

### Request

```
<req name="select" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <select>
    <expr><attr name="fieldName1"/></expr>
  [
    <expr><attr name="fieldName2"/></expr>
    :
    <expr><attr name="fieldNameN"/></expr>
  ]
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Value is PoolStateEntity for the PoolState transparent data

- *fieldNameX*: The requested user defined field in the pool profile.

For the PoolState entity, this corresponds to a property in the entity

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

Values: 1 to 9999999999999999999999

**Response**

```
<req name="select" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>fieldValue1</rv> | <rv null="y"> | <rv></rv> >
    [
      <rv>fieldValue2</rv> | <rv null="y"> | <rv></rv> >
      :
      <rv>fieldValueN</rv> | <rv null="y"> | <rv></rv> >
    ]
  </row>
</rset>
]
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 is expected if the specified data exists (whether or not the field was found). A value of 0 is expected if the data does not exist
- *fieldValue*: The value of the requested field

**NOTES**

- The `<rset>` (row set) element is optional. It is only present if the request was successful. One `<row>` element is returned per matching data. One `<rv>` (field value) element exists for every `fieldNameX` supplied in the original request. The `<rv>` elements are ordered the same as the `fieldNameX` properties were specified in the original request. If the field is valid, but not present in the entity, this is indicated with `<rv null="y">`. If the field is present, but has an empty value, this is indicated with `<rv></rv>`.
- If the PSO feature is enabled, and the UDR is the non pool host UDR for the specified pool, then empty data returns, as pool entity data is only stored on the pool host UDR.

**Error Codes 414: Get Data Field**

| Error Code          | Description   |
|---------------------|---|
| INTF_ENTY_NOT_FOUND | Interface Entity Not Found  |
| FIELD_UNDEFINED     | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC |
| KEY_NOT_FOUND       | Key Not Found. A subscriber or pool with the given key cannot be found                      |
| REG_DATA_NOT_FOUND  | Register Data Not Found   |

## Get Data Field Examples

### Request 1

A request is made to get three PoolState properties from the PoolState transparent data for a pool. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolStateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
    <expr><attr name="expire"/></expr>
    <expr><attr name="duration"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="/"><value val="100000"/></expr>
  </where>
</req>
```

### Response 1

The request is successful, two values are returned and one was not set. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>315</rv>
      <rv>2015-02-09T11:20:32</rv>
      <rv null="y"/>
    </row>
  </rset>
</req>
```

### Request 2

A request is made to get the mcc property in the PoolState transparent data for a pool. The mcc property is not set. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolStateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="/"><value val="100000"/></expr>
  </where>
</req>
```

### Response 2

The request is successful, and the property is indicated to not be set. The original request is not included.

```
<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y">
    </row>
  </rset>
</req>
```

### Request 3

A request is made to get the mcc property in the PoolState transparent data for a pool. The PoolState Entity does not exist. The request is not required in the response.

```
<req name="select" resonly="y">
  <ent name="PoolStateEntity"/>
```

```

<select>
  <expr><attr name="mcc"/></expr>
</select>
<where>
  <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
</where>
</req>

```

**Response 3**

The request fails. The error value indicates the PoolState entity does not exist, and the affected subscribers are 0. The original request is not included.

```

<req name="select" resonly="y">
  <res error="70027" affected="0"/>
</req>

```

**Request 4**

A request is made to get three PoolState properties from the PoolState transparent data for a PSO pool, where the UDR instance receiving the request is not the pool host UDR for the pool in the request. The request is not required in the response.

```

<req name="select" resonly="y">
  <ent name="PoolStateEntity"/>
  <select>
    <expr><attr name="mcc"/></expr>
    <expr><attr name="expire"/></expr>
    <expr><attr name="duration"/></expr>
  </select>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>

```

**Response 4**

The request is successful, and empty data is returned. The original request is not included.

```

<req name="select" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv null="y"/>
    </row>
  </rset>
</req>

```

**7.6.3 Update Data Field****Description**

This operation updates an existing field in a transparent data for the pool identified by the poolId.

The field name is specified in fieldNameX, and the field value is specified in fieldValueX.

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

**NOTES**

- If the requested fields are valid, but not present, they are created.
- The *fieldNameX* is not case-sensitive. If a field name existed called mcc, then an attempt to update a field called MCC is successful.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied.
- If a field being updated is specified more than once in a request, the last value specified is used.

- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.

### Request

```
<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><value val="fieldValue1"/></expr>
  [
    <expr><attr name="fieldName2"/><value val="fieldValue2"/></expr>
    :
    <expr><attr name="fieldNameN"/><value val="fieldValueN"/></expr>
  ]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

- o Value is *PoolStateEntity* for the PoolState transparent data

- *fieldNameX*: The requested user defined field in the pool profile.

- o For the PoolState entity, this corresponds to a property in the entity
- o The *fieldNameX* case is stored exactly as it was sent in the request (This means the original case stored changes if an update is received)

- *fieldValueX* Corresponding field value assigned to *fieldNameX*
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length

Values: 1 to 99999999999999999999

### Response

```
<req name="update" [resonly="resonly"] [id="id"]>
  [
    originalXMLRequest
  ]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 is expected for success

#### Error Codes 425: Update Data Field

| Error Code           | Description   |
|----------------------|---|
| INTF_ENTY_NOT_FOUND  | Interface Entity Not Found  |
| OCC_CONSTR_VIOLATION | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_VAL_INVALID    | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC                                      |
| FIELD_UNDEFINED      | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE  | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND        | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND   | Register Data Not Found   |

#### Update Data Field Examples

##### Request 1

A request is made to update the mcc and approved properties of the PoolState transparent data for a pool. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="mcc"/><value val="302"/></expr>
    <expr><attr name="approved"/><value val="no"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

##### Response 1

The request is successful, and mcc and approved property values are updated. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
```

##### Request 2

A request is made to update the approved property in the PoolState transparent data for a pool. The PoolState Entity does not exist for the pool. The request is not required in the response.

```
<req name="update" resonly="y">
```



```

<ent name="PoolStateEntity"/>
<set>
  <expr><attr name="approved"/><value val="no"/></expr>
</set>
<where>
  <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
</where>
</req>

```

## Response 2

The request fails because the PoolState entity does not exist for the pool. The original request is not included.

```

<req name="update" resonly="y">
  <res error="70027" affected="0"/>
</req>

```

## 7.6.4 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 fieldNameX.

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

### NOTES

- The deletion of a non-existent field is not considered an error.
- The *fieldNameX* is not case-sensitive. If a field existed called mcc, then an attempt to delete a field called MCC is successful.
- If a request both updates and deletes the same field, then the update is applied first, followed by the delete, irrespective of the order in which they are supplied
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The entityName must reference a valid pooled transparent Entity in the Interface Entity Map table in the SEC.

### Request

```

<req name="update" [resonly="resonly"] [id="id"]>
  <ent name="entityName"/>
  <set>
    <expr><attr name="fieldName1"/><op value="="/>
      <value val="" isnull="y"/></expr>
    [
      <expr><attr name="fieldName2"/><op value="="/>
        <value val="" isnull="y"/></expr>
      :
      <expr><attr name="fieldNameN"/><op value="="/>
        <value val="" isnull="y"/></expr>
    ]
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </where>
</req>

```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Value is PoolStateEnti ty for the PoolState transparent data

- *fieldNameX*: The requested user defined field in the pool profile.

For the PoolState entity, this corresponds to a property in the entity

- *fieldValueX*: Corresponding field value assigned to *fieldNameX*
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length

Values: 1 to 9999999999999999999999

**Response**

```
<req name="update" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values:A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 indicates the field was deleted. A value of 0 indicates the field did not exist

**Error Codes 436: Delete Data Field**

| Error Code           | Description   |
|----------------------|---|
| INTF_ENTY_NOT_FOUND  | Interface Entity Not Found  |
| OCC_CONSTR_VIOLATION | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field |
| FIELD_UNDEFINED      | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |
| FIELD_NOT_UPDATABLE  | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| KEY_NOT_FOUND        | Key Not Found. A subscriber or pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND   | Register Data Not Found   |

## Delete Data Field Examples

### Request 1

A request is made to delete the `mcc` property of the `PoolState` transparent data for a pool. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

### Response 1

The request is successful and `mcc` property is deleted. The original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

### Request 2

A request is made to delete the `mcc` property in `PoolState` transparent data for a pool. The `mcc` property does not exist for the `PoolState` data. The request is not required in the response.

```
<req name="update" resonly="y">
  <ent name="PoolStateEntity"/>
  <set>
    <expr><attr name="mcc"/><op value="="/><value val="" isnull="y"/></expr>
  </set>
  <where>
    <expr><attr name="PoolID"/><op value="="/><value val="100000"/></expr>
  </where>
</req>
```

### Response 2

The request is successful and the original request is not included.

```
<req name="update" resonly="y">
  <res error="0" affected="1"/>
</req>
```

## 7.7 Additional Pool Commands

Table 26: 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) | <req name="operation"><br><oper name="AddPoolMember">  |
| Remove Member from Pool | Remove subscriber from a pool              |   | <req name="operation"><br><oper name="DelPoolMember">  |
| Get Pool Members        | Retrieve pool member subscribers by PoolID | PoolID                                      | <req name="operation"><br><oper name="GetPoolMembers"> |

| Command                  | Description   | Keys                           | Command Syntax  |
|--------------------------|---|--------------------------------|---|
| Get Pool by Member (Key) | Retrieve PoolID for specified member subscriber                                   | MSISDN, IMSI, NAI or AccountId | <req name="operation"><br><oper name="GetPoolID">         |
| Get All Pool Members     | Retrieve pool member subscribers from all local or local/remote systems by PoolID | PoolID                         | <req name="operation"><br><oper name="GetAllPoolMembers"> |

### 7.7.1 Add Member to Pool

#### Description

This operation adds one or more subscribers 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 full pool member until the pool is created or pool profile updated on the pool host UDR when the connection between the two is active. In this situation, the subscriber is treated as a pool member by SOAP commands, but not as a pool member by Diameter Sh commands.

#### Prerequisites

- A pool with the key of the poolId supplied must exist.
- Separate subscribers with the Keys of the keyNameX/keyValueX supplied must exist.
- Each subscriber must not be a member of a pool.
- The pool must have less than the maximum number of member subscribers allowed.

#### Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="AddPoolMember">
    <expr><param name="PoolID"/><op value="/"><value val="poolId"/></expr>
    <expr><param name="subKeyName1"/><op value="/">
      <value val="subKeyValue1"/></expr>
  [
    <expr><param name="subKeyName2"/><op value="/">
      <value val="subKeyValue2"/></expr>
    :
    <expr><param name="subKeyName25"/><op value="/">
      <value val="subKeyValue25"/></expr>
  ]
  </oper>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Values: 1 to 99999999999999999999

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

Value is either IMSI, MSISDN, NAI, or AccountId

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

**NOTES**

- Up to 25 subscribers can be added in one request.
- On a low capacity server configuration, the maximum number of AddPoolMember requests that can be included in a <tx> is 3, if each request adds 25 members. Although nothing in the software limits the number of requests to 3, this is the recommended value.
- The number of subscribers being added must not cause the number of members in the pool to exceed the maximum allowed value, else the request fails.
- If any subscriber specified is a member of a pool, the request fails.

**Response**

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values:A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 or more is expected for success

**Error Codes 47: Add Member to Pool**

| Error Code             | Description   |
|------------------------|---|
| KEY_NOT_FOUND          | Key Not Found. A subscriber or pool with the given key cannot be found                                  |
| ALREADY_POOL_MEMBER    | Pool member exists. The subscriber is a member of a pool  |
| POOL_NOT_FOUND         | Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist |
| MAX_MEMBERS_BASIC_POOL | The maximum number of subscribers in a basic pool has been exceeded.                                    |

**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. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>
```

**Response 1**

The request is successful, and the subscriber is added to the pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to add a subscriber to a pool. The pool exists, but the subscriber does not. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="200002"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="15141234567"/></expr>
  </oper>
</req>
```

**Response 2**

The request fails. The error value indicates that the subscriber does not exist, and the affected rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70019" affected="0"/>
</req>
```

**Request 3**

A request is made to add a subscriber to a pool. The subscriber exists, but the pool does not. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="300003"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>
```

**Response 3**

The request fails. The error value indicates that the pool does not exist, and the affected rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70036" affected="0"/>
</req>
```

**Request 4**

A request is made to add a subscriber to a pool (either local or remote). Both the pool and the subscriber exist. The subscriber is a member of a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>
```

**Response 4**

The request fails. The error value indicates the subscriber is a member of a pool, and the affected rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70023" affected="0"/>
</req>
```

**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 pool has the maximum number of members allowed. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>
```

**Response 5**

The request fails. The error value indicates the basic pool has the maximum number of members allowed, and the affected rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70051" affected="0"/>
</req>
```

**Request 6**

A request is made to add 3 subscribers to a pool. The pool and all subscribers exist. Subscribers are not a member of a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="800000"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
    <expr><param name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="14162221234"/></expr>
  </oper>
</req>
```

**Response 6**

The request is successful, and the 3 subscribers are added to the pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 7**

A request is made to add a subscriber to a PSO pool, where the UDR instance receiving the request is not the pool host UDR for the pool in the request. The non pool host UDR pool has 25 subscribers. The pool profile from the pool host UDR is not received on the non pool host UDR. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="AddPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="400000"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
```

```

        <value val="33123654862"/></expr>
    </oper>
</req>

```

### Response 7

The request is successful, and the subscriber is added to the pool as no limit is enforced on a non pool host UDR as long as the pool profile from the pool host UDR is not received. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>

```

## 7.7.2 Remove Member from Pool

### Description

This operation removes one or more subscribers from a pool.

### Prerequisites

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

Separate subscribers with the Keys of the keyNameX and keyValueX supplied must exist.

Each subscriber must be a member of the specified pool.

### Request

```

<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="DelPoolMember">
    <expr><param name="PoolID"/><op value=""/><value val="poolId"/></expr>
    <expr><param name="subKeyName1"/><op value=""/>
      <value val="subKeyValue1"/></expr>
  [
    <expr><param name="subKeyName2"/><op value=""/>
      <value val="subKeyValue2"/></expr>
    :
    <expr><param name="subKeyName25"/><op value=""/>
      <value val="subKeyValue25"/></expr>
  ]
  </oper>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Values: 1 to 99999999999999999999

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

Value is either IMSI, MSISDN, NAI, or AccountId

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



**NOTES**

- Up to 25 subscribers can be removed in one request.
- On a low capacity server configuration, the maximum number of DelPoolMember requests that can be included in a <tx> is 3, if each request deletes 25 members. Although nothing in the software limits the number of requests to 3, this is the recommended value.
- If any subscriber specified is not a member of the pool, the request fails.

**Response**

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
  <res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values: A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools updated. A value of 1 or more is expected for success

**Error Codes 448: Remove Member from Pool**

| Error Code      | Description   |
|-----------------|---|
| KEY_NOT_FOUND   | Key Not Found. A subscriber or pool with the given key cannot be found                                  |
| NOT_POOL_MEMBER | Not a pool member   |
| POOL_NOT_FOUND  | Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist |

**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. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="DelPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>
```

**Response 1**

The request is successful, and the subscriber is removed from the pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

**Request 2**

A request is made to remove a subscriber from a pool. Both the pool and the subscriber exist. The subscriber is not a member of the pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="DelPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="200000"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
  </oper>
</req>
```

**Response 2**

The request fails. The error value indicates the subscriber is not a member of the pool, and the affected rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70025" affected="0"/>
</req>
```

**Request 3**

A request is made to remove 3 subscribers from a pool. The pool and all subscribers exist. All subscribers are a member of the pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="DelPoolMember">
    <expr><param name="PoolID"/><op value="="/><value val="800000"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="15145551234"/></expr>
    <expr><param name="IMSI"/><op value="="/>
      <value val="302370123456789"/></expr>
    <expr><param name="MSISDN"/><op value="="/>
      <value val="14162221234"/></expr>
  </oper>
</req>
```

**Response 3**

The request is successful, and the 3 subscribers are removed from the pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="3"/>
</req>
```

**7.7.3 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**

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="GetPoolMembers">
    <expr><param name="PoolID"/><op value="="/><value val="poolId"/></expr>
  </oper>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)

- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Values: 1 to 99999999999999999999

## Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <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>]]>
      </rv>
    </row>
  </rset>
]
</req>
```

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values:A string with 1 to 4096 characters

- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of Pools returned. A value of 1 is expected for success
- *keyNameX*: A key field for the member subscriber

Value is either IMSI, MSISDN, NAI, or AccountId

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

**NOTES**

- The <rset> (row set) element is optional. It is only present if the request was successful. Only a single <row> element is returned, with one <rv> (row value) element.
- 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.
- The format of this response can be returned in a legacy SPR compatible mode if UDR is configured to do so. See 7.8.1Appendix C for more details.

**Error Codes 49: Get Pool Members**

| Error Code     | Description   |
|----------------|---|
| POOL_NOT_FOUND | Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist |

**Get Pool Members Examples**

**Request 1**

A request is made to get the list of subscribers for a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolMembers">
    <expr><param name="PoolID"/><op value="/"><value val="100000"/></expr>
  </oper>
</req>
```

**Response 1**

The request is successful, and the 3 member subscribers are returned. The original request is not included.

```
<req name="operation">
  <res error="0" affected="1"></res>
  <rset>
    <row>
      <rv>
        <![CDATA[<?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>
          </members>
        </CDATA>
      </rv>
    </row>
  </rset>
</req>
```

```

        <id><name>NAI</name><value>joe@wireless.com</value></id>
        <id><name>NAI</name><value>p12321@mynet.com</value></id>
      </member>
    </members>]]>
  </rv>
</row>
</rset>
</req>

```

**Request 2**

A request is made to get the list of subscribers for a pool. The pool exists, but has no member subscribers. The request is not required in the response.

```

<req name="operation" resonly="y">
  <oper name="GetPoolMembers">
    <expr><param name="PoolID"/><op value="="/><value val="200000"/></expr>
  </oper>
</req>

```

**Response 2**

The request is successful, and no member subscribers are returned. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 3**

A request is made to get the list of subscribers for a pool. The pool does not exist. The request is not required in the response.

```

<req name="operation" resonly="y">
  <oper name="GetPoolMembers">
    <expr><param name="PoolID"/><op value="="/><value val="300000"/></expr>
  </oper>
</req>

```

**Response 3**

The request fails. The error value indicates that the pool was not found, and the affected rows are 0. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="70036" affected="0"/>
</req>

```

**7.7.4 Get PoolID****Description**

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

**Prerequisites**

A subscriber with the Keys of the keyNameX/keyValueX values supplied must exist.

The subscriber must be a member of a pool.

**Request**

```

<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="GetPoolID">
    <expr><param name="keyName1"/><op value="="/>
      <value val="keyValue1"/></expr>
  [
    <expr><param name="keyName2"/><op value="="/>

```

```

        <value val="keyValue2"/></expr>
    :
    <expr><param name="keyNameN"/><op value="" />
        <value val="keyValueN"/></expr>
]
</oper>
</req>

```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response

Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)

- **id:** (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Value is either IMSI, MSISDN, NAI, or AccountId

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

**Response**

```

<req name="operation" [resonly="resonly"] [id="id"]>
[
    originalXMLRequest
]
<res error="error" affected="affected"/>
[
    <rset>
        <row>
            <rv>poolId</rv>
        </row>
    </rset>
]
</req>

```

- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of subscribers returned. A value of 1 is expected for success
- **poolId:** PoolID value of the pool the subscriber is a member of. Numeric value, 1 to 22 digits in length

Values: 1 to 99999999999999999999

**NOTE:** The *<rset>* (row set) element is optional. It is only present if the request was successful, and the subscriber is a member of a pool. Only a single *<row>* element is returned, with one *<rv>* (row value) element, which contains the PoolID of the subscriber.

**Error Codes 50: Get PoolID**

| Error Code    | Description  |
|---------------|--|
| KEY_NOT_FOUND | Key Not Found. A subscriber or pool with the given key cannot be found |

| Error Code      | Description   |
|-----------------|---|
| NOT_POOL_MEMBER | Not a pool member. The 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. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolID">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="33123654862"/></expr>
    </oper>
  </req>
```

#### Response 1

The request is successful, and the PoolID value was returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>100000</rv>
    </row>
  </rset>
</req>
```

#### Request 2

A request is made to get the PoolID for a subscriber. The subscriber exists, but is not a member of a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolID">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="15141234567"/></expr>
    </oper>
  </req>
```

#### Response 2

The request fails. The error value indicates that the subscriber is not a member of a pool. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70025" affected="0"/>
</req>
```

#### Request 3

A request is made to get the PoolID for a subscriber. The subscriber does not exist. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolID">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="15145556789"/></expr>
    </oper>
  </req>
```

**Response 3**

The request fails. The error value indicates that the subscriber does not exist, and the affected rows are 0. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70019" affected="0"/>
</req>
```

**Request 4**

A request is made to get the PoolID for a subscriber. Both MSISDN and AccountId keys are supplied, and reference the same subscriber. The subscriber is a member of a pool. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetPoolID">
    <expr><param name="MSISDN"/><op value="="/>
      <value val="15141234567"/></expr>
    <expr><param name="AccountId"/><op value="="/>
      <value val="111222333"/></expr>
  </oper>
</req>
```

**Response 4**

The request is successful, and the PoolID value was returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
  <rset>
    <row>
      <rv>100000</rv>
    </row>
  </rset>
</req>
```

**7.7.5 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.

**NOTE:** The addressList values (IMSI and ALL) are not case-sensitive.

**Prerequisites**

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

**Request**

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="GetAllPoolMembers">
    <expr><param name="PoolID"/><op value="="/>
      <value val="poolId"/></expr>
    [ <expr><param name="AddressList"/><op value="="/>
      <value val="addressList"/></expr> ]
    [ <expr><param name="PSO"/><op value="="/>
      <value val="pso"/></expr> ]
  </oper>
</req>
```

- **resonly:** (Optional) Indicates whether the response consists of the result only, without including the original request in the response



Values:

- o y—Provides the result only, does not include the original request
- o n—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

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

Values: 1 to 99999999999999999999

- *addressList*: (Optional) The requested address type of the subscriber for a pool is returned in the response

Values:

- o IMSI—Only pool members with an IMSI address type are returned in the response
- o All—All pool members are returned in the response (default)
- *pso*: (Optional) 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:

- o yes—Response includes pool members across all UDR instances in the pool network
- o no—Response includes only 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.

**Response**

```
<req name="operation" [resonly="resonly"] [id="id"]>
[ originalXMLRequest ]
  <res error="error" affected="affected"/>
[ <rset>
  <row>
    <rv>
      <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <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>]]>
</rv>
  </row>
</rset> ]
</req>
```

- **resonly:** (Optional) The *resonly* value from the original XML request, if supplied
- **originalXMLRequest:** (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request

Values: A string with 1 to 4096 characters

- **udrid:** (Optional) A subscriber key range that is hosted by each UDR in the remote pool network. The response only includes the *udrid* if the Pools Spanning UDRs feature is enabled.

Value: 1 to 10 digits

- **id:** (Optional) The *id* value from the original XML request, if supplied
- **error:** Error code indicating outcome of request. 0 means success, see below for other values
- **affected:** The number of Pools returned. A value of 1 is expected for success
- **keyNameX:** A key field for the member subscriber

Value is either IMSI, MSISDN, NAI, or AccountId

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

**NOTES**

- The *<rset>* (row set) element is optional. It is only present if the request was successful. Only a single *<row>* element is returned, with one *<rv>* (row value) element.
- 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.
- The response includes pool members across all UDR instances in the pool network if the *psd* 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 address types, then the pool member is not included in the response.
- If the PSO feature is enabled, then each pool member returned also includes the *udrid* corresponding to the UDR instance where they reside in the pool network.

**Error Codes 51: Get All Pool Members**

| Error Code      | Description   |
|-----------------|---|
| POOL_NOT_FOUND  | Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist |
| REQUEST_TIMEOUT | Provisioning Request Timeout, a response was not 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. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
    <expr><param name="PoolID"/><op value="/"><value val="100000"/></expr>
  </oper>
</req>
```

**Response 1**

The request is successful, and the 3 subscriber members are returned. Only pool members from the local UDR instance are returned. The original request is not included.

```
<req name="operation">
  <res error="0" affected="1"></res>
  <rset>
    <row>
      <rv>
        <![CDATA[<?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>]]>
        </rv>
      </row>
    </rset>
  </req>
```

**Request 2**

A request is made to get the list of subscriber members for a pool. The pool exists, but has no subscriber members. The request is not required in the response. The PSO feature is enabled.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
    <expr><param name="PoolID"/><op value="/"><value val="200000"/></expr>
  </oper>
</req>
```

**Response 2**

The request is successful, and subscriber members are not returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"></res>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <members>
            </members>]]>
        </rv>
      </row>
    </rset>
  </req>
```

```

    </rset>
</req>

```

### 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. The request is not required in the response.

```

<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="300000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>

```

### Response 3

The request fails. The error value indicates that the pool does not exist on all queried UDR instances (local or remote), and the affected rows are 0. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="70036" affected="0"/>
</req>

```

### 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. The request is not required in the response.

```

<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>

```

### 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. The original request is not included.

```

<req name="operation">
  <res error="0" affected="1"></res>
  <rset>
    <row>
      <rv>
        <![CDATA[<?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>2</udrId>
              <id><name>MSISDN</name><value>380561234777</value></id>
              <id><name>IMSI</name><value>311480100000999</value></id>
            </member>
            <member>
              <udrId>3</udrId>
              <id><name>NAI</name><value>joe@wireless.com</value></id>
              <id><name>NAI</name><value>p12321@mynet.com</value></id>
            </member>
          </members>]]>
      </rv>
    </row>
  </rset>
</req>

```

```

    </row>
  </rset>
</req>

```

### **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 request is not required in the response. The PSO feature is enabled. There is a connection issue between the local and remote UDRs.

```

<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>

```

### **Response 5**

The request fails. The error value indicates a provisioning request timeout, the request is not sent to the remote UDR due to the connection being down, and the affected rows are 0. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="70053" affected="0"/>
</req>

```

### **Request 6**

A request is made to get the list of subscribers for a pool on all UDRs. The PSO feature is enabled. The request is not required in the response. For this case, a response is not received from the remote UDR.

```

<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="100000"/></expr>
<expr><param name="PSO"/><op value="="/><value val="yes"/></expr>
  </oper>
</req>

```

### **Response 6**

The request fails. The error value indicates a provisioning request timeout, a response was not received from the remote UDR. The affected rows are 0 and the original request is not included.

```

<req name="operation" resonly="y">
  <res error="70053" affected="0"/>
</req>

```

### **Request 7**

A request is made to get the list of subscriber members for a pool. The requested addressList type is IMSI and none of the subscriber members have IMSIs. The request is not required in the response. The PSO feature is enabled.

```

<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
<expr><param name="PoolID"/><op value="="/><value val="200000"/></expr>
<expr><param name="AddressList"/><op value="="/><value val="IMSI"/></expr>
  </oper>
</req>

```

**Response 7**

The request is successful, and subscriber members are not returned. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/></res>
  <rset>
    <row>
      <rv>
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
          <members>
            </members>]]>
      </rv>
    </row>
  </rset>
</req>
```

**Request 8**

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. The request is not required in the response.

```
<req name="operation" resonly="y">
  <oper name="GetAllPoolMembers">
    <expr><param name="PoolID"/><op value="/"><value val="100000"/></expr>
    <expr><param name="PSO"/><op value="/"><value val="yes"/></expr>
  </oper>
</req>
```

**Response 8**

The request is successful, and the 3 member subscribers are returned. The response includes local pool members only. The original request is not included.

```
<req name="operation">
  <res error="0" affected="1"/></res>
  <rset>
    <row>
      <rv>
        <![CDATA[<?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>]]>
      </rv>
    </row>
  </rset>
</req>
```

**7.8 Pool Special Operation Commands**

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

The required row is identified in the request by the rowIdName.

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

**Table 27: Summary of Pool Special Operation Commands**

| Command          | Description                                 | Keys                       | Command Syntax  |
|------------------|---|----------------------------|---|
| Reset Pool Quota | Reset the fields in the specified PoolQuota | Pool ID and Row Identifier | <pre>&lt;req name="operation"&gt;   &lt;oper name="Reset" ent="PoolQuotaEntity"&gt;</pre> |

### 7.8.1 Reset PoolQuota

#### Description

This operation resets a particular quota row in the PoolQuota data associated with a pool.

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

If the pool has PoolQuota data, then the configured values in the specified PoolQuota row are reset to the configured default values.

#### NOTES

- The rowIdName is case-sensitive. If a row existed 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 Reset 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.
- This operation is ignored on an NPHO and a success is returned. Updates are not made to the database for these requests on NPHO.

#### Prerequisites

- A pool with the key of the poolId supplied must exist.
- The PoolQuota transparent data must exist for the pool.
- The specified PoolQuota row must exist in the PoolQuota transparent data.

#### Request

```
<req name="operation" [resonly="resonly"] [id="id"]>
  <oper name="Reset" ent="entityName">
    <expr><param name="PoolID"/><op value=""/>
      <value val="poolID"/></expr>
    <expr><param name="rowIdName"/><op value=""/>
      <value val="rowIdValue"/></expr>
    [
      <expr><param name="instanceFieldName"/><op value=""/>
        <value val="instanceFieldValue"/></expr>
    ]
  </oper>
</req>
```

- *resonly*: (Optional) Indicates whether the response consists of the result only, without including the original request in the response

#### Values:

- o *y*—Provides the result only, does not include the original request
- o *n*—Includes the original request in the response (default)
- *id*: (Optional) Transaction ID value in the request, and passed back in the response

Values: 1 to 4294967295

- *entityName*: A user defined entity type/name for the transparent data  
Value is PoolQuotaEntity for the PoolQuota transparent data
- *poolId*: PoolID value of the pool. Numeric value, 1 to 22 digits in length  
Values: 1 to 99999999999999999999
- *rowIdName*: Name of the XML attribute that identifies the row in the data blob  
Value is name for Quota 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 for the PoolQuota transparent data
- *instanceFieldValue*: Corresponding field value assigned to *instanceFieldName*

**Response**

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.  
**NOTE:** This is always present unless the *resonly="y"* attribute is set in the original request  
Values:A string with 1 to 4096 characters
- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools reset. A value of 1 is expected for success

**Error Codes 452: Reset PoolQuota**

| Error Code          | Description   |
|---------------------|---|
| KEY_NOT_FOUND       | Key Not Found. A pool with the given key cannot be found  |
| REG_DATA_NOT_FOUND  | Register Data Not Found   |
| ROW_NOT_FOUND       | Data row specified is not found   |
| MULTIPLE_ROWS_FOUND | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria |

**Reset PoolQuota Examples**

**Request 1**

A request is made to reset the PQ1 PoolQuota row for a pool. The pool has PoolQuota data, and the PoolQuota data contains a PoolQuota row called PQ1. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
```



```

    <expr><param name="PoolID"/><op value="="/>
      <value val="100000"/></expr>
    <expr><param name="name"/><op value="="/><value val="PQ1"/></expr>
  </oper>
</req>

```

**Response 1**

The request is successful, and the specified PoolQuota row was reset. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>

```

**Request 2**

A request is made to reset the Monthly PoolQuota row. The pool does not have PoolQuota data. The request is not required in the response.

```

<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
      <value val="200000"/></expr>
    <expr><param name="name"/><op value="="/><value val="Monthly"/></expr>
  </oper>
</req>

```

**Response 2**

The request fails. The error value indicates the pool does not have PoolQuota data, and the affected rows are 0. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="70027" affected="0"/>
</req>

```

**Request 3**

A request is made to reset the PQ6 PoolQuota row. The pool has PoolQuota data, but the PoolQuota data does not contain a PoolQuota row called PQ6. The request is not required in the response.

```

<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
      <value val="300000"/></expr>
    <expr><param name="name"/><op value="="/><value val="PQ6"/></expr>
  </oper>
</req>

```

**Response 3**

The request fails, because the PQ6 data row was not present. The original request is not included.

```

<req name="operation" resonly="y">
  <res error="70032" affected="0"/>
</req>

```

**Request 4**

A request is made to reset the PQ6 PoolQuota row. The pool has PoolQuota data with multiple rows called PQ6. The request is not required in the response.

```

<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
      <value val="400000"/></expr>
    <expr><param name="name"/><op value="="/><value val="PQ6"/></expr>
  </oper>

```

```
</req>
```

#### **Response 4**

The request fails, because multiple rows exist with the rows called PQ6. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="70035" affected="0"/>
</req>
```

#### **Request 5**

A request is made to reset the PQ1 PoolQuota row for a pool with cid field 1234. The pool has PoolQuota data, and the PoolQuota data contains a PoolQuota row called PQ1. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
      <value val="400000"/></expr>
    <expr><param name="name"/><op value="="/><value val="PQ1"/></expr>
    <expr><param name="cid" /><op value="="/><value val="1234"/></expr>
  </oper>
</req>
```

#### **Response 5**

The request is successful, and the specified PoolQuota row was reset. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

#### **Request 6**

A request is made to reset the PQ1 PoolQuota row for a pool with cid field 1234. The pool has PoolQuota data, the PoolQuota data contains a PoolQuota row called PQ1 and cid field. The request is not required in the response.

```
<req name="operation">
  <oper name="Reset" ent="PoolQuotaEntity">
    <expr><param name="PoolID"/><op value="="/>
      <value val="500000"/></expr>
    <expr><param name="name"/><op value="="/><value val="PQ1"/></expr>
    <expr><param name="cid" /><op value="="/><value val="1234"/></expr>
  </oper>
</req>
```

#### **Response 6**

The request is successful, and the specified PoolQuota row was reset. The original request is not included.

```
<req name="operation" resonly="y">
  <res error="0" affected="1"/>
</req>
```

## Appendix A. SOAP Interface Error Codes

SOAP error codes are returned by the SOAP interface in the error attribute parameter of the <res> message (see section 5.2.2). The error parameter of a response message indicates the success or failure of a request.

The complete set of response error codes and their associated values are defined in the following table.

The Type column indicates if an error is permanent (P) or temporary (T), or indicates success (S). A request that results in a permanent error is discarded and not sent again. A request that results in a temporary can be sent again at a different time, and may be successful.

Error codes that are marked with an \* (asterisk) are permanent errors that can be fixed by means of configuration, such as configuring the entities or fields in the SEC.

### Error Codes 53: SOAP Interface

| Error Code            | Value | Type | Description   |
|-----------------------|-------|------|---|
| NOT_PROCESSED         | 1     | P    | Not processed. The request was in a block transaction, and was not processed due to an error with another request in the same block transaction |
| INTF_ENTY_NOT_FOUND   | 70000 | P*   | Interface Entity Not Found  |
| ENTY_DEF_NOT_FOUND    | 70002 | P    | Entity Definition Not Found   |
| VER_BFS_NOT_FOUND     | 70003 | P    | Versioned Base Field Set for the Transparent Entity Not Found   |
| NON_VER_BFS_NOT_FOUND | 70004 | P    | Non Versioned Base Field Set for the Transparent Entity Not Found   |
| MULT_VER_TAGS_FOUND   | 70005 | P    | Multiple Version Tags Found   |
| FIELD_VAL_INVALID     | 70006 | p*   | Field Value Not Valid. The value for a given field is not valid based on the definition in the SEC  |
| OCC_CONSTR_VIOLATION  | 70007 | P*   | Occurrence Constraint Violation. There are too many instances of a given field. Likely more than one instance of a non-repeatable field         |
| INVAL_REPEATABLE_ELEM | 70008 | P    | Invalid Repeatable Element  |
| INVALID_XML           | 70009 | P    | Invalid Input XML   |
| FLD_SET_NOT_FOUND     | 70010 | P    | Field Set Not Found   |
| FLD_SET_EXISTS        | 70011 | P    | Field Set Exists  |
| FIELD_NOT_FOUND       | 70012 | P    | Field Not Found   |
| FIELD_EXISTS          | 70013 | P    | Field Exists  |
| FLD_SET_UNDEFINED     | 70014 | P    | Field Set Not Defined   |
| FIELD_UNDEFINED       | 70015 | P    | Field Not Defined. The given field is not a valid field in the entity as defined in the SEC   |

| Error Code          | Value | Type | Description   |
|---------------------|-------|------|---|
| FIELD_NOT_UPDATABLE | 70016 | P    | Field Cannot be Updated. The field is defined in the SEC as not be updatable  |
| ENT_CANNOT_RESET    | 70017 | P    | Entity Cannot be Reset. The reset command cannot be used on the requested entity  |
| DB_OPER_FAILED      | 70018 | P    | Database Operation Failed   |
| KEY_NOT_FOUND       | 70019 | P    | Key Not Found. A subscriber or pool with the given key cannot be found  |
| KEY_EXISTS          | 70020 | P    | Key Exists. A subscriber or pool exists with the given key  |
| SUB_IN_POOL         | 70021 | P    | Subscriber is pool member. The subscriber is a member of a pool. A subscriber cannot be deleted if they are a pool member   |
| HAS_POOL_MEMBERS    | 70022 | P    | Has pool members. A pool cannot be deleted when it has member subscribers   |
| ALREADY_POOL_MEMBER | 70023 | P    | Pool member exists. The subscriber is a member of a pool  |
| MAX_POOL_MEMBERS    | 70024 | P    | Pool member List Max Limit Reached  |
| NOT_POOL_MEMBER     | 70025 | P    | Not a pool member   |
| OPER_NOT_ALLOWED    | 70026 | P    | Operation Not Allowed   |
| REG_DATA_NOT_FOUND  | 70027 | P    | Register Data Not Found   |
| REG_EXISTS          | 70028 | P    | This error code is utilized for two scenarios:<br>1. Register exists<br>2. A property exists and the odk flag was not set to indicate the insert request be converted to an update. |
| UNEXPECTED_ERROR    | 70029 | P    | Un-Expected Error   |
| INV_SOAP_XML        | 70030 | P    | Invalid SOAP XML  |
| SERVICE_UNAVAILABLE | 70031 | T    | Service is unavailable. Provisioning has been disabled  |
| ROW_NOT_FOUND       | 70032 | P    | Data row specified is not found   |
| VALUE_EXISTS        | 70033 | P    | List value added exists   |
| FLD_NOT_MULTI       | 70034 | P    | Field is not a multi-value field. Add and remove from list operations can only be performed on a multi-value field, and the field supplied is not multi-value                       |
| MULTIPLE_ROWS_FOUND | 70035 | P    | Multiple rows match the given criteria. When updating a row, only one row can exist that match the given row criteria   |

| Error Code                      | Value | Type | Description   |
|---------------------------------|-------|------|---|
| POOL_NOT_FOUND                  | 70036 | P    | Pool does not exist. A subscriber cannot be added, retrieved or removed from a pool that does not exist   |
| INVALID_KEY_VALUE               | 70037 | P    | The key value supplied is invalid, due to invalid characters or format.   |
| DB_RETRY_EXHAUSTED              | 70038 | T    | Data is not committed to database as the total number of retries to commit database transactions exhausted.<br><b>NOTE:</b> The client retries the command again  |
| DURABILITY_DEGRADED             | 70039 | T    | Data is not committed as Durability is degraded.<br><b>NOTE:</b> The client retries the command again   |
| DURABILITY_TIMEOUT              | 70040 | T    | Data is not made durable in the configured Durability Timeout.<br><b>NOTE:</b> The client retries the command again to get the data sent in the failed request to verify that it was stored by last request |
| TOO_BIG_MESSAGE                 | 70041 | P    | The provisioning request size exceeded the maximum allowed size   |
| MEMORY_FULL                     | 70042 | P    | Free system memory is low. Request cannot be performed  |
| MULTIPLE_KEYS_NOT_MATCH         | 70043 | P    | Multiple keys supplied do not refer to the same subscriber  |
| ONE_KEY_REQUIRED                | 70044 | P    | At least one key is required for a subscriber   |
| SYSTEM_CONGESTION               | 70045 | T    | Request rejected due to system congestion   |
| CONNECTION_ERROR                | 70046 | P    | Request to send to different UDR instance is not sent because a valid connection is not available to send the request   |
| MAX_MEMBERS_BASIC_POOL          | 70051 | P    | The maximum number of subscribers in a basic pool has been exceeded   |
| ENTERPRISE_TO_BASIC_POOL_FAILED | 70052 | P    | Enterprise to Basic Pool Conversion failed because the Pool has more members than the maximum threshold for a basic pool.   |
| REQUEST_TIMEOUT                 | 70053 | P    | Provisioning Request Timeout, a response was not received from Remote UDR   |
| INVALID_USER_CREDENTIALS        | 70054 | P    | Authentication of the Username/Password received in the SOAP message header has failed validation.  |
| AE_KEY_EXISTS                   | 70055 | P    | An AE subscriber exists with the given key. Only applicable when option enableAEKeyAlreadyExistsErrCode is enabled  |

## Appendix B. SOAP Interface System Variables

The SOAP interface has a set of system variables that affect its operation as it runs. SOAP interface variables (Table 28) can be set via the UDR GUI and can be changed at runtime to effect dynamic server reconfiguration.

**Table 28: SOAP Interface System variables**

| Parameter                         | Description   |
|-----------------------------------|---|
| SOAP Interface Port               | SOAP 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 REST interface.<br><br>Default is 62001; Range is 0 to 65535   |
| SOAP Interface Idle Timeout       | The maximum time (in seconds) that an open SOAP connection remains active without a request being sent, before the connection is dropped.<br><br>Default is 1200; Range is 1 to 86400   |
| Maximum SOAP Connections          | Maximum number of simultaneous SOAP Interface client connections.<br><br><b>NOTE:</b> Changes to the Maximum SOAP Connections option do not take effect until the udrprov process is restarted.<br><br>Default is 100; Range is 1 to 100  |
| Allow SOAP Connections            | Whether or not to allow incoming provisioning connections on SOAP Interface.<br><br>Default is UNCHECKED  |
| Transaction Durability Timeout*   | The amount of time (in seconds) allowed between a transaction being committed and it becoming durable. If Transaction Durability Timeout lapse, DURABILITY_TIMEOUT response is sent to the originating client. The associated request is resent to ensure that the request was committed.<br><br>Default is 5; Range is 2 to 3600 |
| Compatibility Mode*               | Indicates whether backwards compatibility is enabled. NOTE: Change to Compatibility Mode may cause the existing provisioning connections to be dropped.<br><br>Default is R10.0+  |
| Maximum Requests in SOAP <tx> XML | The maximum number of requests in a single SOAP tx transaction.<br><br>Default is 12; RANGE=1 to 50   |

**NOTE:** Parameters labeled with an \* (asterisk) are existing system variables defined and used by other components of UDR.

## Appendix C. 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 B), which is configurable by the UDR GUI, is set to R10+ then UDR behaves as described in the main body of this document. When `Compatibility Mode` is set to R9.x, then the differences contained in this appendix apply.

Enabling this configuration option results in the format of some requests and responses being different to the default UDR behavior. This appendix lists the different requests and responses that enabling the option applies to.

### Get Pool Members Response Format

UDR returns the list of pool members in the format using a `<members>` XML element, as returned by the REST interface.

When configured in legacy SPR mode, UDR returns the list of pool members in the format as described below.

#### Response

```
<req name="operation" [resonly="resonly"] [id="id"]>
[
  originalXMLRequest
]
<res error="error" affected="affected"/>
[
  <rset>
    <row>
      <rv>publicIdentity</rv> | <rv/> >
      <rv>MSISDN1[,MSISDN2[,MSISDN3]]</rv> | <rv/> >
      <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <rv/> >
      <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv/> >
      <rv>accountId</rv> | <rv/> >
    </row>
  ]
  <row>
    <rv>publicIdentity</rv> | <rv/> >
    <rv>MSISDN1[,MSISDN2[,MSISDN3]]</rv> | <rv/> >
    <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <rv/> >
    <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv/> >
    <rv>accountId</rv> | <rv/> >
  </row>
  :
  <row>
    <rv>publicIdentity</rv> | <rv/> >
    <rv>MSISDN1[,MSISDN2[,MSISDN3]]</rv> | <rv/> >
    <rv>IMSI1[,IMSI2[,IMSI3]]</rv> | <rv/> >
    <rv>NAI1[,NAI2[,NAI3]]</rv> | <rv/> >
    <rv>accountId</rv> | <rv/> >
  </row>
]
</rset>
]
</req>
```

- *originalXMLRequest*: (Optional) The text of the original XML request that was sent.

**NOTE:** This is always present unless the `resonly="y"` attribute is set in the original request

Values: A string with 1 to 4096 characters

- *resonly*: (Optional) The *resonly* value from the original XML request, if supplied
- *id*: (Optional) The *id* value from the original XML request, if supplied
- *error*: Error code indicating outcome of request. 0 means success, see below for other values
- *affected*: The number of pools returned. A value of 1 or more is expected for success

- *publicIdentity*: The internal public identity value for the subscriber. This field is not used, and a value is not present
- *MSISDNX*: Comma separated list of (up to 3) MSISDN values corresponding to subscriber in the pool. Values are not present if an MSISDN is not provisioned for the subscriber  
Values: A string with 8 to 15 digits (if value is set)
- *IMSIX*: Comma separated list of (up to 3) IMSI values corresponding to subscriber in the pool. Values are not present if an IMSI is not provisioned for the subscriber  
Values: A string with 10 to 15 digits (if value is set)
- *NAIX*: Comma separated list of (up to 3) NAI values corresponding to subscriber in the pool. Values are not present if an NAI is not provisioned for the subscriber
- Refer to Table 8 for supported NAI formats and length
- *accountId*: AccountId corresponding to subscriber in the pool. This value is not present if an AccountId is not provisioned for the subscriber  
Values: A string with 1 to 255 characters (if value is set)

**NOTE:** The <rsset> (row set) element is optional. It is only present if the request was successful. One subscriber is returned per <row> element returned, each always containing five <rv> (row value) elements.

## C.1 LEGACY SPR SOAP REQUEST FORMAT

The legacy SPR uses a different SOAP request format as described below. Requests can be sent using the same format as done for the legacy SPR.

**NOTE:** The <soapenv:Header> element in the request is optional and can be omitted. It can be provided, but are ignored. Authentication is not performed using the Username/Passwd in UDR.

**Figure 6: Legacy SPR SOAP Request Format**

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <soapenv:Header>
    <ns1:UserName
      soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      xsi:type="soapenc:string"
      xmlns:ns1="blueslice.com"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">Username</ns1:UserName>

    <ns2:Passwd
      soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      xsi:type="soapenc:string"
      xmlns:ns2="blueslice.com"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">Password</ns2:Passwd>
    </soapenv:Header>

    <soapenv:Body>

      <processTransaction xmlns="http://webservice.blueslice.com">

        <![CDATA[REQUEST]]>

      </processTransaction>

    </soapenv:Body>
  </soapenv:Envelope>
```



## Legacy SPR SOAP Response Format

When UDR is configured in legacy SPR mode, even when SOAP requests are sent using the SOAP request format as specified in Figure 6: Legacy SPR SOAP Request Format, the SOAP response format is different to the format that the legacy SPR returns. Based on the implementation of UDR, this is unavoidable.

The response does not cause a provisioning client that uses a native SOAP interface any issues (for example, one that uses the WSDL file), as it is a valid SOAP response. But, if any provisioning clients have been implemented to look for specific XML elements (such as <soapenv:Envelope> instead of <SOAP-ENV:Envelope>), this problems may arise.

The format of the SOAP response returned by UDR is listed below in Figure 7.

**NOTE:** The <SOAP-ENV:Header> is only returned if a <SOAP-ENV:Header> is included in the request. In UDR, the <SOAP-ENV:Header> is optional in the request, and is ignored.

### Figure 7: Legacy SPR SOAP Response Format

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://webservice.blueslice.com">

  <SOAP-ENV:Header/>

<
  <SOAP-ENV:Body>

    <ns1:message error="ErrorCode">

      <![CDATA[RESPONSE]]>

    </ns1:message>

  </SOAP-ENV:Body>
|
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <SOAP-ENV:Fault>
      ...
    </SOAP-ENV:Fault>

  </SOAP-ENV:Body>
>

</SOAP-ENV:Envelope>
```

Example of a successful SOAP response in legacy SPR mode:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://webservice.blueslice.com">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:message error="0">
      <![CDATA[<req name="insert" resonly="y">
        <res affected="1" error="0"/>
      </req>]]>
    </ns1:message>
  </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

### soapAttributeOrderInResponse

In ProvOptions table, if soapAttributeOrderInResponse flag is set as TRUE then SOAP Response has error followed by attribute in res element, otherwise the order is reversed.

Example of a successful SOAP response when soapAttributeOrderInResponse flag is set as FALSE (Default case):

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.oracle.com/udr/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
  <ns1:message error="0">
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?><req name="insert" resonly="y">
      <res affected="1" error="0"/>
    </req>]]>
  </ns1:message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of a successful response when soapAttributeOrderInResponse flag is set as TRUE:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.oracle.com/udr/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
  <ns1:message error="0">
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?><req name="insert" resonly="y">
      <res error="0" affected="1"/>
    </req>]]>
  </ns1:message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### validateProvResponse

In ProvOptions table, if validateProvResponse is set as TRUE then the response for the provisioning Read/Get requests are validated and changed as per the SEC configuration.

Example of a successful SOAP response when validateProvResponse flag is set as FALSE (Default case):

BillingDay field is changed to billingday for the subscriber. Provisioning Read/Get request is not validated as per SEC configuration while retrieving subscriber. See the response below:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.oracle.com/udr/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
  <ns1:message error="0">
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
      <req name="select" resonly="y">
        <res affected="1" error="0">
          <rset>
            <row>
```

```

    <rv>
      <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
        <subscriber>
          <field name="MSISDN">9876543210</field>
          <field name="billingday">23</field>
          <field name="Entitlement">DayPass</field>
          <field name="Custom20">xyz</field>
          <field name="Entitlement">DayPassPlus</field>
        </subscriber>]]]]>
      <![CDATA[>
    </rv>
  </row>
</rset>
</req>]]>
</ns1:message>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

### Example of a successful SOAP response when “validateProvResponse” flag is set as TRUE:

BillingDay field is changed to billingday for subscriber. Provisioning Read/Get request is validated as per SEC configuration while retrieving subscriber. See the response below:

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.oracle.com/udr/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
  <ns1:message error="0">
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
      <req name="select" resonly="y">
        <res affected="1" error="0"/>
        <rset>
          <row>
            <rv>
              <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
                <subscriber>
                  <field name="MSISDN">9876543210</field>
                  <field name="BillingDay">23</field>
                  <field name="Entitlement">DayPass</field>
                  <field name="Custom20">xyz</field>
                  <field name="Entitlement">DayPassPlus</field>
                </subscriber>]]]]>
              <![CDATA[>
            </rv>
          </row>
        </rset>
      </req>]]>
    </ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

## C.2 ENABLE AE KEY ALREADY EXISTS ERROR

In ProvOptions table, if enableAEKeyAlreadyExistsErrCode is set as TRUE (default set as FALSE) then error AE\_KEY\_EXISTS (70055) returns when following criterias are met:

A create subscriber request encounters key exists failure

The key which exists belongs to an AE subscriber

**Example of a SOAP response when enableAEKeyAlreadyExistsErrCode flag set to TRUE and create subscriber request failed when key exists failure:**

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://www.oracle.com/udr/"
xmlns:ns2="http://www.3gpp.org/udc/notification">
<SOAP-ENV:Body>
  <ns1:message error="0">
    <![CDATA[<?xml version="1.0" encoding="UTF-8"?><req name="insert" resonly="y">
      <res error="70055" affected="0"/>
      </req>]]>
    </ns1:message>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## Appendix D. 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, select the options 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 following options:
  - o For Technical issues such as creating a Service Request (SR), Select **1**
  - o For Non-technical issues such as registration or assistance with My Oracle Support, Select **2**

You are connected to a live agent who can assist you with My Oracle Support registration and opening a support ticket.

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

## Appendix E. Locate Product Documentation on the Oracle Help Center Site

Oracle Communications customer documentation is available on the web at the Oracle Help Center (OHC) site, <http://docs.oracle.com>. You do not have to register to access these documents. Viewing these files requires Adobe Acrobat Reader, which can be downloaded at <http://www.adobe.com>.

1. Access the Oracle Help Center site at <http://docs.oracle.com>
2. Click **Industries**.
3. Under the Oracle Communications subheading, click the **Oracle Communications documentation** link.
4. The Communications Documentation page displays. Most products covered by these documentation sets are under the headings Network Session Delivery and Control Infrastructure or Platforms.
5. Click your Product and then the Release Number.
6. A list of the entire documentation set for the selected product and release displays.
7. To download a file to your location, right-click the **PDF** link, select **Save target as** (or similar command based on your browser), and save to a local folder.