Proteus

API Guide

Version 4.0.6

BLUECAT

Legal Notices

Important – Read this page to ascertain binding terms.

Copyright

Trademarks

BlueCat Networks, BlueCat, the BlueCat Logo, Proteus and the Proteus logo, Adonis and the Adonis logo, Triton and the Triton logo, BlueCat DNS/DHCP Server, BlueCat Automation Manager, BlueCat Address Manager, BlueCat Address Manager for Windows DNS/DHCP, BlueCat Device Registration Portal, and BlueCat Threat Protection are trademarks of BlueCat Networks, Inc. or BlueCat Networks (USA) Inc. iDRAC is a registered trademark of Dell Inc. Windows is a registered trademark of Microsoft Corporation. UNIX is a registered trademark of The Open Group. Linux is a registered trademark of Linus Torvalds. All other product and company names are registered trademarks or trademarks of their respective holders.

Export Warning

BlueCat equipment is a Class A product. In a domestic environment, this product may cause radio interference, in which case you may be required to take appropriate measures.

Canadian Regulatory Compliance

BlueCat equipment is a Class A digital device that complies with Canadian ICES-003.

FCC Compliance

BlueCat equipment generates, uses and may emit radio frequency energy. This equipment has been type tested and found to comply with the limits for a Class A digital device pursuant to part 15 of FCC rules that are designed to provide reasonable protection against such radio frequency interference.

Operation of this equipment in a residential area may cause interference that may require you to take reasonable measures to correct at your expense.

Any modifications to this device, unless expressly approved by the manufacturer, can void the user’s authority to operate this equipment under part 15 of the FCC rules.

Disclaimer

a) Read this document before installing or using the product. Failure to follow the prescribed instructions may void the BlueCat warranty.

b) BlueCat has granted you the right to use this document. BlueCat believes the information it furnishes to be accurate and reliable, but BlueCat assumes no responsibility for, or arising out of, your use of the information except to the extent expressly set out in the e*nd-user agreement* (‘EUA’) if any, binding you and BlueCat associated with the product. No license is granted by implication or otherwise under any patent, copyright or other intellectual property right of BlueCat, except as specifically described in the above noted EUA, if any.

c) BlueCat reserves the right to change specifications at any time without notice.

d) BlueCat assumes no responsibility for any inaccuracies in this document. BlueCat reserves the right to change, modify, transfer or otherwise revise this publication without notice.

Contents

Preface 15

What's New 17

Proteus version 4.0.6 API 17
New Methods 17
Changes 17
Proteus version 4.0.5 API 18
New Methods 18
Changes 18
Proteus version 4.0.0 P1 API 18
New Methods 18
New Constant 18
Proteus version 4.0.0 API 18
New Methods 19
New Constants 19
Changes 19

Chapter 1: Overview

Manipulating Proteus Objects 24
Implementation 24
Finding Objects 24
Proteus Object Hierarchy 24
Proteus Object Tree 25
Logging In and Out of Proteus 26
Session Management 27
Security 27
Enabling SSL in Perl Clients 27
Enabling SSL in Java Clients 27

Chapter 2: The Proteus API

Web Services API 30
SOAP Binding Address 30

SOAP Ports 30
Maintaining state with cookies 30
API Objects 30
APIEntity Class 31
Class 31
APIDeploymentRole Class 31
APIDeploymentOption Class 32
APIUserDefinedField Class 32
API Sessions 33
Connect to Proteus 33
Log in and Log out 33
System Information 34
Java API Examples 34
Install Apache Axis 34
Connecting to Proteus 34
Logging In 35
Adding Objects 35
Getting Objects 37
Deleting Objects 37
Logging Out 38
Sequence of Calls in the Client 38
Available Java Classes 39
EntityProperties 39
API Constants 40
ProteusAPIUtils 40
Generating Java Artifacts from the WSDL 41
Perl API Examples 41
Connecting to Proteus 42
Logging in 42
Getting, Adding, Deleting, and Updating Objects 43
Logging out 44
Change to Perl Syntax in Proteus v3.1 44

Chapter 3: API Object Methods

Generic Methods 48
Getting Objects 48
Get Entity by Name 48
Get Entity by ID 48
Get Entities 49
Get Parent 50
Searching for and Retrieving Entities 50
Search by Category 50
Search by Object Types 51

Get Entities by Name 51
Get Entities by Name Using Options 52
Get MAC Address 53
Updating Objects 54
Update 54
Update with Options 55
Deleting Objects 56
Delete 56
Delete with Options 56
Linked Entities 56
Get Linked Entities 56
Link Entities 57
Unlink Entities 58
User-defined Fields 59
Getting User-defined Fields 59
Get User-defined Field 59
Update Bulk User-defined Field 59
IPAM 61
IPv4 Blocks 61
Add IPv4 Block by CIDR 61
Add IPv4 Block by Range 62
Add Parent Block 62
Add Parent Block with Properties 63
Get IP Range by IP Address 63
Get IPv4 Block by CIDR 64
Get IPv4 Block by Range 65
Merge Blocks with Parent 66
Merge Selected Blocks or Networks 66
Update IPv4 Block 67
IPv4 Block Generic Methods 67
IPv4 Networks 67
Add IPv4 Network 68
Get IPv4 Range by IP Address 68
Get IPv4 Network by CIDR 69
Get IPv4 Network by Hint 70
Get IPv4 Network by Range 71
Get Next Available Network 72
Get Next Available IP Range 72
Get Next Available IP Ranges 73
Split IPv4 Network 75
Update IPv4 Network 75
IPv4 Network Generic Methods 75
Add IPv4 Reconciliation Policy 76

IPv4 Network Templates 78
Add IPv4 Network Template 78
Assign or Update Template 79
Re-apply Template 81
Update IPv4 Network Template Name 83
IPv4 Network Template Generic Methods 83
IPv4 Addresses 84
Assign IPv4 Address 84
Assign Next Available IPv4 Address 85
Get IPv4 Address 86
Get Next IPv4 Address 86
Check Allocation for IPv4 Address 87
Allocate Next Available Address 88
Get Dependent Records 88
Update IPv4 Address 89
IPv4 Address Generic Methods 89
Change IPv4 Address State 90
IPv4 Group 90
Add IPv4 IP Group by Range 90
Add IPv4 IP Group by Size 91
IPv4 Objects 91
Move IPv4 Object 91
Move IP Object 92
Resize Range 93
IPv6 Objects 93
Add IPv6 Address 93
Add IPv6 Block by MAC Address 94
Add IPv6 Block by Prefix 95
Add IPv6 Network by Prefix 95
Get IPv6 Range by IP Address 96
Assign IPv6 Address 97
Clear IPv6 Address 98
Get Entity by Prefix 98
Get IPv6 Address 99
Reassign IPv6 Address 100
Update IPv6 Objects 100
Provision Devices 100
Add Device Instance 100
De-provision Devices 102
Delete Device Instance 102
DHCP 102
IPv4 DHCP Ranges 103
Add IPv4 DHCP Range 103

Add IPv4 DHCP Range By Size 103
Get IPv4 Range by IP Address 104
Get IPv4 DHCP Range 105
Get IPv4 DHCP Ranges 105
Get Max Allowed Range 106
Update IPv4 DHCP Range 106
IPv4 DHCP Range Generic Methods 106
IPv6 DHCP Ranges 107
Add IPv6 DHCP Range 107
Get IPv6 Range by IP Address 107
Get IPv6 DHCP Range 108
Get Multiple IPv6 DHCP Ranges 108
Update IPv6 DHCP Range 109
IPv6 DHCP Range Generic Methods 109
DHCP Client Options 109
Add DHCP Client Option 110
Get DHCP Client Option 110
Update DHCP Client Option 111
Delete DHCP Client Option 111
DHCP6 Client Options 112
Add DHCP6 Client Option 112
Get DHCP6 Client Option 112
Update DHCP6 Client Option 113
Delete DHCP6 Client Option 113
DHCP Custom Options 114
Add Custom Deployment Option 114
DHCP Service Options 114
Add DHCP Service Option 115
Get DHCP Service Option 116
Update DHCP Service Option 116
Delete DHCP Service Option 116
DHCP6 Service Options 117
Add DHCP6 Service Option 117
Get DHCP6 Service Option 118
Update DHCP6 Service Option 118
Delete DHCP6 Service Option 118
DHCP Vendor Options 119
Add DHCP Vendor Deployment Option 119
Add Vendor Option Definition 120
Add Vendor Profile 121
Delete DHCP Vendor Deployment Option 121
Get DHCP Vendor Deployment Option 122
Update DHCP Vendor Deployment Option 123

DHCP Match Classes 124
Add DHCP Match Classes 124
Update DHCP Match Classes 125
Delete DHCP Match Classes 125
Add DHCP Sub Classes 125
Update DHCP Sub Classes 125
Delete DHCP Match Classes 125
DNS 126
DNS Views 126
Add DNS View 126
Update DNS View 126
DNS View Generic Methods 127
Add Access Control List (ACL) 127
Update Access Control List (ACL) 128
DNS Zones 128
Add Entity for DNS Zones 128
Add Zone 129
Get Zones by Hint 129
Update Zone 130
Zone Generic Methods 130
Get Key Signing Key 131
DNS Zone Templates 131
Add Zone Template 131
Assign or Update Template 132
Update Zone Template 133
Zone Template Generic Methods 133
Add Records to DNS Zone Template 134
ENUM Zones 134
Add ENUM Zone 134
Update ENUM Zone 134
ENUM Zone Generic Methods 135
ENUM Numbers 135
Add ENUM Number 135
Update ENUM Number 135
ENUM Number Generic Methods 136
DNS Resource Records 136
Reference: Handling dotted resource records 136
Generic Resource Records 137
NAPTR Records 139
External Host Records 140
Host Records 141
Alias Records 145
Text Records 146

HINFO Records ... 147
MX Records ... 148
SRV Records ... 150
Start of Authority Records ... 151
Generic Records ... 152
DNS Options ... 153
Add DNS Option ... 153
Get DNS Option ... 154
Update DNS Option ... 154
Delete DNS Option ... 154
DNS Response Policies ... 155
Add Response Policy ... 155
Update Response Policy ... 156
Response Policy Generic Methods ... 156
Upload Response Policy Item ... 156
Deployment options ... 157
Getting deployment options ... 157
Get Deployment Options ... 157
TFTP ... 158
TFTP Groups ... 158
Add TFTP Group ... 158
Update TFTP Group ... 158
TFTP Group Generic Methods ... 158
TFTP Folders ... 159
Add TFTP Folder ... 159
Update TFTP Folder ... 159
TFTP Folder Generic Methods ... 159
TFTP Files ... 159
Add TFTP File ... 160
Update TFTP File ... 160
TFTP File Generic Methods ... 160
Servers and Deployment ... 160
Servers ... 161
Add Server ... 161
Import Server ... 163
Replace Server ... 163
Deploy Server ... 164
Deploy Server Configuration ... 165
Deploy Server Services ... 165
Quick Deployment ... 166
Deployment Status ... 167
Server Generic Methods ... 167
Get Published Interface ... 168

- DNS and DHCP Deployment Roles 168
  - Get Servers Associated with a Deployment Role 168
  - Get Server’s Associated Deployment Roles 168
  - Get Deployment Roles for DNS and IP Address Space Objects 169
  - Move Deployment Roles 169
- DHCP Deployment Roles 170
  - Add DHCP Deployment Role 170
  - Get DHCP Deployment Role 171
  - Update DHCP Deployment Role 171
  - Delete DHCP Deployment Role 172
- DNS Deployment Roles 172
  - Add DNS Deployment Role 173
  - Get DNS Deployment Role 173
  - Get DNS Deployment Role for View 174
  - Update DNS Deployment Role 174
  - Delete DNS Deployment Role 175
  - Delete DNS Deployment Role for View 175
- TFTP Deployment Roles 176
  - Add TFTP Deployment Role 176
  - Update TFTP Deployment Role 176
  - TFTP Deployment Role Generic Methods 176

Crossover High Availability (XHA) 177
- Requirements for creating an XHA pair 177
- Creating an XHA 177
  - Create XHA 177
  - Edit XHA 178
- Breaking an XHA 180
  - Break XHA 180
- XHA Failover 181
  - Failover XHA 181

Proteus Objects 182
- Configurations 182
  - Add Configuration 182
  - Update Configuration 182
  - Configuration Generic Methods 182
- Groups and Users 183
  - Add Group 183
  - Update Group 183
  - Group Generic Methods 184
  - Add User 184
  - Update User 185
  - User Generic Methods 185
- Authenticators 186

Update Authenticator 186
Authenticator Generic Methods 186
Access Rights 187
Add Access Right 187
Get Access Right 188
Get Access Rights for Entity 189
Get Access Rights for User 189
Update Access Rights 190
Delete Access Rights 191
Object Tag Groups 191
Add Object Tag Group 191
Update Object Tag Group 192
Object Tag Group Generic Methods 192
Object Tags 192
Add Object Tag 192
Assign Object Tag 193
Remove Object Tag 193
Update Object Tag 194
Object Tag Generic Methods 194
Database Management 194
Configure Replication 194
Devices 195
Add Device 195
Add Device Type 196
Add Device Subtype 196
MAC Pools 197
Get MAC Addresses in Pool 197
Update MAC Pool 197
MAC Pool Generic Methods 198
MAC Addresses 198
Add MAC Address 198
Associate MAC Address 198
Deny MAC Address 199
Is Address Allocated? 199
Update MAC Address 200
MAC Address Generic Methods 200
Workflow Change Requests 200
Migration 202
Migrate a File 202
Migration Status 202

Chapter 4: API Constants

Access Right Values ... 203
DHCP Deployment Role Types ... 203
Deployment Services ... 203
Deployment Status ... 204
DHCP Service Options ... 204
DHCPServiceOptionConstants ... 205
DHCP6 Service Options ... 206
DHCP Client Options ... 206
DHCP6 Client Options ... 209
DHCP Class Match Criteria ... 209
DHCP Custom Option Types ... 210
DNS Options ... 210
DNS Option Values ... 211
DNS Deployment Role Types ... 212
DNS Zones Deployment Validation Check ... 212
DNSSEC Key Format ... 212
IP Assignment Action Values ... 213
Object Properties ... 213
Object Types ... 220
PositionRangeBy ... 222
Response Policy Type ... 222
Entity Categories ... 223
ENUM Services ... 223
User-defined Field Type ... 224
User-defined Field Validator Properties ... 224
User History Privileges ... 225
User Type ... 225
User Access Type ... 225
User Security Privileges ... 226
Workflow Levels ... 226
SNMPSecurityLevels ... 226
Server Capability Profiles ... 226
Service Types ... 227
Traversal Methodology ... 227
Vendor Profile Option Types ... 227

Chapter 5: API Method Reference

API Sessions ... 229
Generic Methods ... 229
Linked Entities ... 230
User-defined Fields ... 230
IPAM ... 231
IPv4 Blocks ... 231
IPv4 Networks ... 231
IPv4 Network Templates ... 232

- IPv4 Addresses 233
- IPv4 Objects 234
- IPv4 Group 234
- IPv6 Objects 234
- Provision Devices 235
- De-provision Devices 235

DHCP 236

- IPv4 DHCP Ranges 236
- IPv6 DHCP Ranges 236
- DHCP Client Options 236
- DHCP6 Client Options 237
- DHCP Custom Options 237
- DHCP Service Options 237
- DHCP6 Service Options 238
- DHCP Vendor Options 238
- DHCP Match Classes 238

DNS 240

- DNS Views 240
- DNS Zones 240
- DNS Zone Templates 240
- ENUM Zones 241
- ENUM Numbers 242
- Generic Resource Records 242
- NAPTR Records 242
- External Host Records 242
- Host Records 243
- Alias Records 243
- Text Records 243
- HINFO Records 244
- MX Records 244
- SRV Records 244
- Start of Authority Records 244
- Generic Records 245
- DNS Options 245
- DNS Response Policies 245

Deployment options 246

TFTP 246

- TFTP Groups 246
- TFTP Folders 246
- TFTP Files 246

Servers and Deployment 247

- Servers 247
- DNS and DHCP Deployment Roles 247
- DHCP Deployment Roles 247
- DNS Deployment Roles 248
- TFTP Deployment Roles 248

Crossover High Availability (XHA) 248
Proteus Objects 250
Configurations 250
Groups and Users 250
Authenticators 250
Access Rights 250
Devices 251
Object Tag Groups 251
Object Tags 252
Database Management 252
MAC Pools 252
MAC Addresses 252
Workflow Change Requests 253
Migration 253

Appendix A: Property Options Reference

Property options 255
Configuration 255
Views and Zones 255
Resource Records 256
Admin 258
Tags 258
Vendor Profiles 258
DNSSEC 259
TFTP Objects 259
MAC Pool Objects 259
Device 259
Kerberos Realms 260
Server 261
IPv4Objects 262
IPv6Objects 264
DeploymentRoles 265
Access right 265
IP address states 265
IPv4 265
IPv6 266

Preface

This guide describes the Proteus Application Programming Interface (API) for controlling the Proteus IPAM Appliance and offers instructions on its implementation and usage. IP Address Management (IPAM) includes management of DNS (Domain Name Services), DHCP (Dynamic Host Control Protocol), and IP Inventories.

Who should read this guide?

This book is intended for a highly technical audience. This audience may include developers; IT planners; and IPAM, DNS, and DHCP administrators.

How is this book organized?

Procedural information is organized in numbered points to help in rapid implementation. Examples that are longer than a couple of lines are separated from other information, and are clearly marked with an Example heading. All examples are in pseudocode unless otherwise indicated.

Chapter 1: *Overview*—describes the general functionality of the Proteus API. This chapter describes the common sequences of operations and lists the available objects and methods.

Chapter 2: *The Proteus API*—describes the SOAP, Java, and Perl API implementations.

Chapter 3: *API Object Methods*—describes the methods available for the Proteus object types.

Chapter 4: *API Constants*—lists the constants used by the Proteus API methods.

Chapter 5: *API Method Reference*—lists methods by Proteus object type. This list provides hyperlinks to the method descriptions in Chapters 3 and 4.

References

Working with an IPAM system requires in-depth knowledge of many subject areas, including DNS, DHCP, IP Inventory Management and General Networking. The following references are provided for readers who require more background knowledge before working with Proteus.

- *The DHCP Handbook, Second Edition* by Ralph Droms and Ted Lemon, SAMS Publishing, ISBN 0-672-32327-3
- *Pro DNS and BIND* by Ron Aitchison, Apress, ISBN 1-59059-494-0
- The Internet System Consortium: www.isc.org.
- The BIND FAQ: https://www.isc.org/software/bind/documentation
- The DHCP FAQ: https://www.isc.org/software/dhcp/documentation.

How do I contact BlueCat Networks Client Care?

For 24/7/365 support, visit the BlueCat Networks CARE Portal at http://www.bluecatnetworks.com/support/support_portal.

What's New

Proteus version 4.0.6 API

New Methods

Proteus v4.0.6 includes the following new methods:

- **public long addACL**

  An Access Control List (ACL) can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 52.

- **long DHCP4RangeBySize**

Changes

Proteus v4.0.6 API includes the following changes:

- Modified behavior for User-defined fields in the **update()** method:
  - **To remove existing UDF values**—commit the **update()** method with an empty UDF value. If the UDF parameter is set to *mandatory*, the method fails as the UDF parameter cannot be empty.
  - **To update UDF values**—commit the **update()** method with the new UDF value. If you do not want to update the existing value, leave the UDF parameter and its value unchanged.
  - If the UDF parameter is set to *mandatory* and has a default value, committing the **update()** method with an empty UDF value will take the default value.
- The following options have been added to the **splitIP4Network()** method:
  - **assignDefaultGateway**
  - **overwriteConflicts**
  - **template**
- The following options have been added to the **replaceServer()** method:
  - **servicesIPv4Address**
  - **servicesIPv4Netmask**
  - **servicesIPv6Address**
  - **servicesIPv6Subnet**
  - **xhaIPv4Address**
  - **xhaIPv4Netmask**

  - **redundancyScenario**
  - **resetServices**

Proteus version 4.0.5 API

New Methods

Proteus v4.0.5 includes the following new methods:

- **public long addParentBlockWithProperties**
- **public APIEntity getNextAvailableIPRanges**

Changes

Proteus v4.0.5 API includes the following changes:

- Support for the new ***reuseExisting*** property has been added to the **APIEntity getNextAvailableIPRange** method.

Proteus version 4.0.0 P1 API

New Methods

Proteus v4.0.0 P1 includes the following new method:

- **APIEntity getNextAvailableIPRange**
- **public String[] getMaxAllowedRange**

New Constant

- *Traversal Methodology* on page 227

Proteus version 4.0.0 API

New Methods

Proteus v4.0.0 API includes the following new methods:

- **public long addResponsePolicy**
- **public void uploadResponsePolicyItems**
- **public long addDHCPMatchClass**
- **public long addDHCPSubClass**
- **public long createXHAPair**
- **public void editXHAPair**
- **public void failoverXHA**
- **public void breakXHAPair**
- **public APIDeployment[] getDeploymentOptions**
- **public void quickDeploy**
- **public APIDeploymentRole[] getServerDeploymentRoles**
- **public void updateWithOptions**
- **void configureReplication**
- **public String[] getKSK**

New Constants

- *SNMPSecurityLevels* on page 226

Changes

Proteus v4.0 API includes the following changes:

- Proteus API now supports multi-port Adonis servers. Existing customers who have upgraded their Proteus API to v4.0.6 may need to update their API calls to add a server with dedicated management enabled. For details on how to add a server, refer to *Add Server* on page 161.
- Changes to **IPv4 Network Templates** when performing *Add*, *Update* and *Get* operations through API:
  - The pipe (|) separator in the sub-type value has been replaced with commas (,).
  - If the name properties contains commas (,) or back-slashes (\), it needs to be escaped with backward slash (\).
- **getEntity()** only retrieves xHA server with the following properties:
  - ***activeNodeId***
  - ***passiveNodeId***
  - ***activeNodePhysicalAddress***

  - ***passiveNodePhysicalAddress***
- **getDNSDeploymentRole()**, **getDNSDeploymentRoleForView()**, and **getDHCPDeploymentRole()** now retrieve a new property ***inherited*** to indicate whether the deployment role(s) is inherited or not.
- The new **getDeploymentOptions()** method has been added to support the generic getting deployment options.
- Support for the new ***defaultView***, ***defaultDomains***, ***dnsRestrictions***, ***allowDuplicateHost*** and ***pingBeforeAssign*** properties have been added to the following API methods:
  - **addIP4BlockByCIDR**
  - **addIP4BlockByRange**
  - **addIP4Network**
  - **update()**—when calling IPv4 block or network.
  - **getEntityById**
  - **getEntityByCIDR**
  - **getEntityByRange**
  - **getParent**
- Support for the new additional ***inheritdefaultView***, ***inheritdefaultDomains***, ***inheritdnsRestrictions***, ***inheritallowDuplicateHost*** and ***inheritpingBeforeAssign*** properties have been added to the following API methods:
  - **addIP4BlockByCIDR**
  - **addIP4BlockByRange**
  - **addIP4Network**
  - **update()**—when calling IPv4 block or network.
  - **getEntityById**
  - **getEntityByCIDR**
  - **getEntityByRange**
  - **getParent**
- Support for the new ***workflowLevel***, ***deploymentAllowed***, and ***quickDeploymentAllowed*** properties has been added to the following API methods:
  - **addAccessRight**
  - **updateAccessRight**

  For more information on the new properties added, please refer to *Property Options Reference* on page 255.
- The ***UserAccessType*** property for the **addUser()** method is now mandatory. You must specify the value for this property to avoid an error. For more information, refer to *Add User* on page 184.
- Property options reference table has been added. Refer to *Property Options Reference* on page 255 to find which property values will be returned and what values can be updated.
- API constants table matching property key to property value has been added. Refer to *API Object Methods* on page 47 to find the constants used in the API methods.
- **updateWithOptions()** now also supports MX and SRV records. For more information, refer to *Update with Options* on page 55.

Chapter 1

Overview

The Proteus API (Application Programming Interface) is a web service that make Proteus accessible to any system that has standard network or Internet access. Use SOAP (Simple Object Access Protocol) to access this web service. The web service has a WSDL (Web Service Description Language) file that can be viewed in a browser. Use the WSDL file to generate client artifacts, such as methods and serialized classes. Implementors can use this service directly. Java and Perl API implementations are also provided.

Manipulating Proteus Objects

The API user remotely manipulates objects through the Proteus API, using a combination of generic and type-specific methods. This manipulation involves adding objects, getting objects by name, updating them, adding new child objects, access rights or object tags to them, and deleting them. The Proteus API also includes various query functions, such as a check to see if an IP address is allocated. To access the Proteus API, the script or application must log in to Proteus as an API user.

Implementation

Sessions in the Proteus API are implemented as Perl or Java programs, or accessed directly through the web service. Generally, these sessions log in, perform a function, and then log out again. Provided the script is successful, the next script then logs in, performs its function, and then logs out.

API sessions follow a pattern:

1. Connect to the Proteus server.
2. Log in.
3. Get the initial configuration, user, group or tag group object and proceed to step 4, or retrieve a specific object by name or ID using **getEntity()** and proceed to step 7.
4. Use **getEntities()** to find the children of the initial object.
5. Use **getEntity()** or a less generic method to select a single entity.
6. Add a child object or affect the current object.
7. Log out.

Variations on this pattern are possible, provided the API implementation can provide sufficient information to retrieve the required objects.

Finding Objects

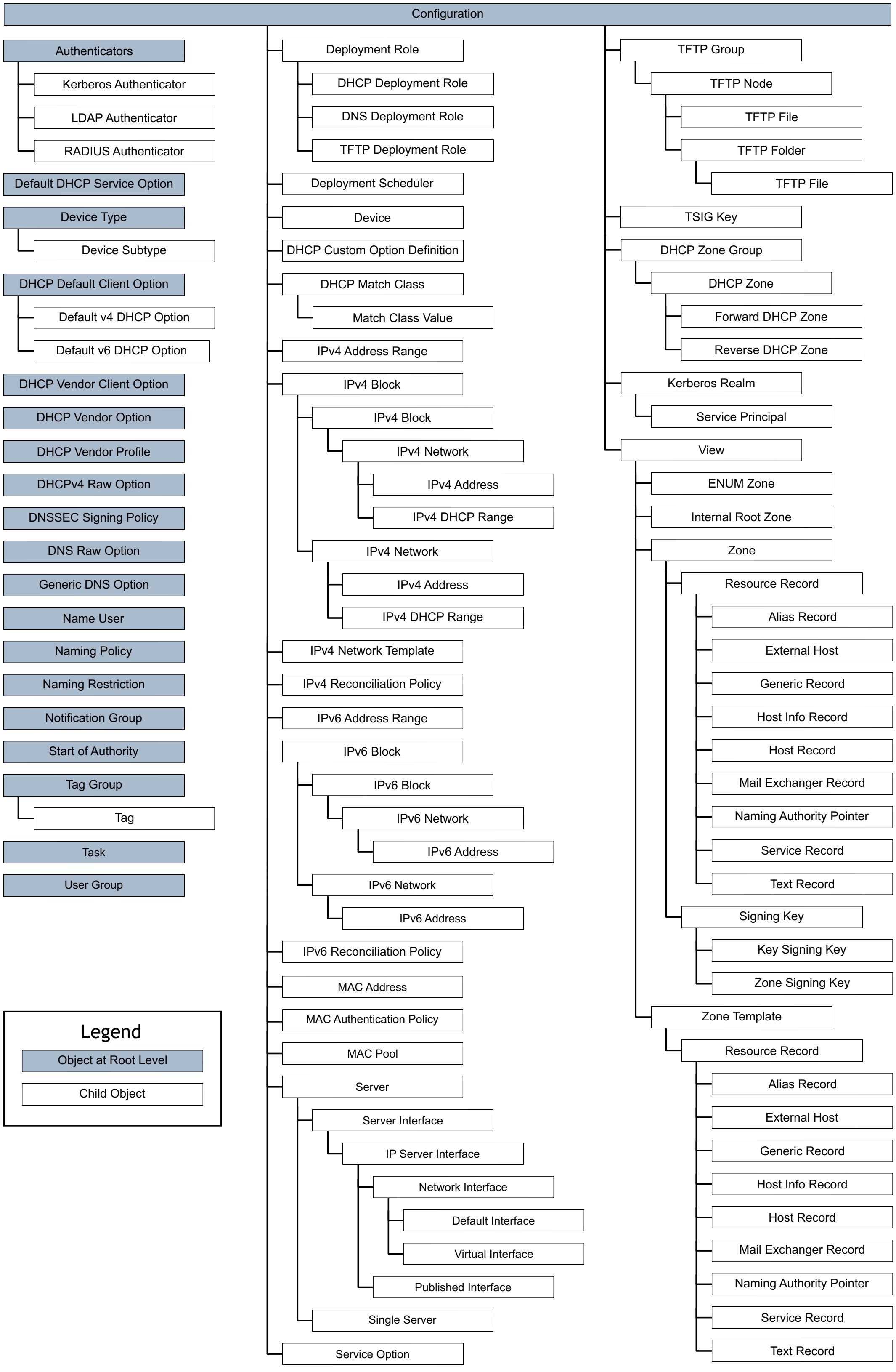
Two search functions provide quick access to Proteus objects, in most cases eliminating the need to walk the entire Proteus object tree:

- **searchByCategory()** returns an array of entities by searching on keywords associated with objects of a specified object category. For more information, see ***Search by Category*** on page 50.
- **searchByObjectTypes()** returns an array of entities by searching on keywords associated with objects of a specified object type. You can search for multiple object types with a single method call. For more information, see ***Search by Object Types*** on page 51.

Proteus Object Hierarchy

The web services API is designed to facilitate various types of development, and can be implemented in many different ways. Ultimately, client-side implementations can model the way that data is stored in Proteus in order to persist objects temporarily. Keeping this structure in mind will help you create caching or reference data structures within client implementations.

Proteus Object Tree



Logging In and Out of Proteus

To access the Proteus API, the script or application must log in to Proteus using an API user account. All API implementations must first connect to the Proteus API service, and then log in to start a session. None of the Proteus API functionality described in the following subsections is accessible unless an API user logs on to the system. Proteus API users have a specific access type (API) and cannot log in to Proteus through the GUI (Graphical User Interface). Similarly, non-API users cannot connect to Proteus through the API interface.

To connect to the service, you need the IP address for the Proteus server. Logging in creates a session for that user with a timeout limit corresponding to that set for all users. API users must also log out after the required operations have been completed.

For more information about API access, refer to the *Proteus Administration Guide* or the online Help.

To create an API user:

1. Using the Proteus web interface, log in to Proteus as an administrator.
2. On the *Administration* page, click **Users and Groups**. The *Users and Groups* page appears.
3. In the Users section, click **New** 👤. The *Add User* page appears.
4. In the User section, type a name in the **Username** field.
5. In the Authentication section, type and confirm the API user password in the **Password** and **Confirm Password** fields.

   If external authenticators are available, an **Other** checkbox and a list of authenticators appears in the Authentication section. To use an external authenticator for the API user, click the **Other** checkbox, and then select an authenticator from the list.
6. In the Extra Information section, set the following parameters:

   E-mail Address—type an email address for the API user. This information is required.

   Phone number—type a phone number for the API user. This information is optional.
7. In the **User Access** section you define the user type, security and history privileges, and access type:
   - **Type of User**—select the type of user, either **Non-Administrator** or **Administrator**. **Non-Administrator** users have access only to DNS and IPAM management functions. **Administrator** users have unlimited access to all Proteus functions.
   - **Security Privilege**—select a security privilege type from the drop-down list. This field is available only for **Non-Administrator** users with **GUI**, **API**, or **GUI and API** access.
   - **History Privilege**—select a history privilege type from the drop-down list. This field is available only for **Non-Administrator** users with **GUI**, or **GUI and API** access.
   - **Access Type**—select the type of access; **GUI**, **API**, or **GUI and API**. **GUI** (Graphical User Interface) users can access Proteus only through the Proteus web interface. **API** (Application Programming Interface) users can access Proteus only through the API. **GUI and API** users can access Proteus either through the Proteus web interface or the API.
8. In the **Assign to Group** section, you can assign the user to one or more existing user groups. In the text field, type the name of a user group. As you type, a list of user groups matching your text appears. Select a name from the list, and then click **Add** to the right of the text field.
9. In the **Change Control** section, add comments to describe the changes. This step is optional but may be set to be required.

10 Click **Add** at the bottom of the page.

Session Management

Web services do not define a standard for session management. Proteus maintains a session ID to associate it with the Proteus database session obtained from the initial log in attempt. This accounts for the stateless nature of the HTTP protocol.

Proteus uses cookies to maintain state. When using WSDL-generated classes, be sure to enable cookies on your system.

Security

The web service endpoint can be made secure by enabling SSL support using the Proteus Administration Console. This enables SSL for all Proteus services, including the web interface.

For instructions on enabling SSL on Proteus, refer to *HTTPS* in *Chapter 10: Appliance Settings* in the *Proteus Administration Guide*.

Enabling SSL in Perl Clients

To enable SSL in Perl scripts, you need to install the Crypt-SSLeay package. You can download the package from *http://search.cpan.org/dist/Crypt-SSLeay/*

Perl scripts can use SSL in their web service calls by using *https* instead of *http* in the proxy definition.

> To ensure that API scripts will continue to operate under the higher security implemented in Proteus v3.7.1 and later, the following conditions must be met:
>
> - Perl version must be v5.12 or later.
> - You must add the following line to the beginning of the script:
>   **$ENV{'PERL_LWP_SSL_VERIFY_HOSTNAME'} = 0;**

Enabling SSL in Java Clients

Follow these steps to enable SSL for Java clients. Before beginning, ensure that the Java Development Kit is installed on the client workstation.

To generate a keystore:

1 On Proteus, enable HTTPS support. For instructions on enabling SSL on Proteus, refer to *HTTPS* in *Chapter 10: Appliance Settings* in the *Proteus Administration Guide*.

2 On the client workstation, create a directory to hold the keystore.

3. On Proteus, locate the **/data/server/conf/server.cert** file, and then copy it to the keystore directory on your client workstation. If SSH is enabled on Proteus, use an SSH client to copy the file.
4. On the client workstation, navigate to the keystore directory. Ensure that the directory contains the **server.cert** file.
5. Execute the following command:

```
javaHomePath/bin/keytool -import -trustcacerts -alias ProteusAPI -file server.cert
-keystore client.ks -storepass bluecat
```

6. Ensure that a **client.ks** file has been generated and appears in the keystore directory.
7. Delete the **server.cert** file from the keystore directory on the client workstation.
8. When connecting to service, use the following command to call the **ProteusAPIUtils.connect()** method:

```
ProteusAPIUtils.connect(IPAddress, true, pathToClientKeystore\\client.ks);
```

where *IPAddress* is the IP address of the Proteus server, and *pathToClientKeystore* is the path to the keystore directory on the client workstation.

Chapter 2

The Proteus API

For integration into different types of network environments, Proteus includes the following implementations:

- The most open implementation is the Proteus service-oriented architecture implementing SOAP web services. Any client able to take advantage of web services can use this implementation.
- A Java API implementation is provided to integrate Proteus into *n*-tier Java architectures.
- A Perl API implementation is provided for environments where scripting is preferred.

This chapter provides information on the following topics:

- *Web Services API* on page 30 provides information on accessing the Proteus API as a SOAP web service.
- *API Sessions* on page 33 provides general information on connecting to the Proteus API and retrieving general system information.
- *Java API Examples* on page 34 discusses how to use the Proteus API through a Java application.
- *Perl API Examples* on page 41 discusses how to use the Proteus API through Perl scripts.

Web Services API

The Proteus API is a SOAP web service, so it has an accessible WSDL file. You can access this file and generate your own classes and methods to use when connecting to the service.

To view the WSDL file in a browser, navigate to the following address:

http://*ProteusAddress*/Services/API?wsdl

If HTTPS is enabled on Proteus, use the HTTPS protocol in the address.

SOAP Binding Address

The WSDL file uses the Proteus server’s host-name as the soap:address location. For a Proteus appliance with the factory default host-name, the Proteus API service looks like this example:

```
<service name='ProteusAPI'>
  <port binding='tns:ProteusAPIBinding' name='ProteusAPIPort'>
   <soap:address location='http://new.server/Services/API'/>
  </port>
 </service>
```

To configure the soap:address location attribute to your Proteus appliance, download a copy of the WSDL file to your workstation. Edit the local copy of the WSDL file to change the soap:address location to the required address. Configure your SOAP tools to load the WSDL file from your local copy of the file, rather than from the Proteus appliance.

SOAP Ports

To access the Proteus API, use the following ports:

- Port 80 when using HTTP
- Port 443 when using HTTPS

Maintaining state with cookies

Proteus uses cookies to maintain state. When using WSDL-generated classes, be sure to enable cookies on your system.

API Objects

The web service defines objects representing all Proteus object types supported in the service. These objects can be added, retrieved, manipulated, and deleted. For a list of objects and methods, refer to *API Method Reference* on page 229.

Four classes reference all objects within the web service:

- APIEntity
-
- APIDeploymentRole
- APIDeploymentOption
- APIUserDefinedField

APIEntity Class

This class represents all entities except options, roles, and access rights. It manages all other types by passing the values for the object as a delimited properties string of name–value pairs. The properties for each object are listed in *API Object Methods* on page 47.

Fields

- **id**—the database ID of the object in Proteus.
- **name**—the field name, which might be null.
- **type**—the class name of the object. For example, a configuration object has a type equal to **Configuration**. This field cannot be null. A list of types is part of the API client (Java and Perl).
- **properties**—a string that contains properties for the object in *attribute=value* format, with each separated by a **|** (pipe) character. For example, a host record object may have a properties field such as **ttl=123|comments=my comment|**. This field can be null.

Class

This class controls Access Rights objects.

Fields

- **entityId**—the database ID of the object to which the access right applies. This value must be greater than 0.
- **userId**—the database ID of the owner of the access right. This value must be greater than 0.
- **value**—the default access right (HIDE, VIEW, ADD, CHANGE, or FULL). This field cannot be null.
- **overrides**—indicates the types that are to be overridden in the access right in the format *objectType=accessRightValue* where *objectType* is the same object type used in APIEntity and *accessRightValue* is one of HIDE, VIEW, ADD, CHANGE or FULL. Multiple override elements are separated by a **|** (pipe) character.
- **properties**—a string containing extra properties for the object in the format *attribute=value.*

APIDeploymentRole Class

Manages the deployment roles that control the services provided by Proteus-managed servers. These objects support the standard object functions.

Fields

- **id**—the database ID of the deployment role in Proteus.
- **type**—the type of the role (NONE, MASTER, MASTER_HIDDEN, SLAVE, SLAVE_STEALTH, FORWARDER, STUB, RECURSION, PEER, or AD_MASTER.) This field cannot be null.
- **service**—DNS, DHCP, or TFTP. This field cannot be null.
- **entityId**—the database ID of entity. This value must be greater than 0.
- **serverInterfaceId**—the database ID of the server interface. This value must be greater than 0.
- **properties**—a string containing extra properties for the object in the format *attribute=value.* This field can be null if used for forward space. A ViewId property must be provided to assign DNS Roles to a Network or Block for a particular DNS View (reverse space). Multiple properties are separated by a **|** (pipe) character.

APIDeploymentOption Class

Deployment options configure both DHCP and DNS services on the network. They are available as DHCP client and service options, as well as standard DNS options. Deployment options support the standard object functions.

Fields

- **id**—the database ID of the option in Proteus.
- **type**—the option type (DNS, DHCPClient, or DHCPService). This field cannot be null.
- **name**—the name of the option.
- **value**—the single- or multiple-field value of the option; multiple values are comma-separated. This field cannot be null.

> When adding the DDNS hostname option, you need to specify the value in the following format: [**Type**],[**Position**],[**Data**] for IP and MAC type, and [**Type**],[**Data**] for FIXED type. Where:
>
> - **Type**—type of DDNS hostname. The possible values are DHCPServiceOptionConstants.DDNS_HOSTNAME_TYPE_IP, DHCPServiceOptionConstants.DDNS_HOSTNAME_TYPE_MAC, or DHCPServiceOptionConstants.DDNS_HOSTNAME_TYPE_FIXED.
> - **Position**—specify where you wish to add the data value to the IP or MAC address. The possible values are DHCPServiceOptionConstants.DDNS_HOSTNAME_POSITION_PREPEND, or DHCPServiceOptionConstants.DDNS_HOSTNAME_POSITION_APPEND.
>
>   This is only required for IP or MAC type with Data.
> - **Data**—For IP and MAC address, this value is used to be prepended or appended to the IP address or MAC address. For FIXED type, this value must be specified and will be used for the DDNS hostname. This is optional for IP and MAC type.

- **properties**—a string containing additional properties. This is used for user-defined fields on most objects, but also passes some values that do not have their own specific parameter.

APIUserDefinedField Class

User-defined fields can be added to each of the Proteus object types. This class allows API users to query and gather user-defined fields information for a specified object type.The properties for each object are listed in *API Object Methods* on page 47.

Fields

- **name**—the internal name of the user-defined field.
- **displayname**—the name of the user-defined field that appears to users in the Proteus interface.
- **type**—the type of the user-defined field. Types are available as constants in the `UserDefinedFieldType` class. For available constants, refer to *User-defined Field Type* on page 224.
- **defaultValue**—the default value for the user-defined field.
- **validatorProperties**—the validation properties for the user-defined field. Property names are available as constants in the `UserDefinedFieldValidatorProperties` class. For available

constants, refer to *User-defined Field Validator Properties* on page 224.

- **required**—the boolean value. If set to true, users must enter data in the field.
- **hideFromSearch**—the boolean value. If set to true, the user-defined field is hidden from the search.
- **renderAsRadioButton**—the boolean value. If set to true, the user-defined field is rendered as a radio button group.

API Sessions

Proteus API session methods control the connection, log in, and log out processes. There is also a method to return system information about the appliance.

Connect to Proteus

To begin an API session, you must first connect to the Proteus appliance using the following method:

```
ProteusAPI_PortType connect (
                           String address,
                           boolean enableSSL,
                           String keystoreLocation )
```

Parameters

address—the address of Proteus appliance.

enableSSL—if security is enabled on the Proteus appliance, set this flag to **true**.

keystoreLocation—the location of the certificate on the client.

If you are using HTTP, *address* is the only required parameter for this method.

Output/Response

This method returns a ProteusAPI_PortType reference containing the methods for the API.

Log in and Log out

Use the following methods to log in and log out of the Proteus system. You must use an API user account to access the Proteus API.

To log in, use the following method, passing the API user name and password:

```
login( String name, String password )
```

To log out, use the following method:

```
logout();
```

System Information

You can retrieve Proteus system information through the API. To retrieve system information, use the following method:

```
String getSystemInfo()
```

Output/Response

This method returns system information in the following format:

hostName=*value*|version=*value*|address=*value*|clusterRole=*value*|replicationRole=*value*|replicationStatus=*value*|entityCount=*value*|databaseSize=*value*|loggedInUsers=*value*

These attributes have the following values:

- **hostName**—the host name of the Proteus server.
- **version**—the version of the Proteus software.
- **address**—the IP address of the Proteus server.
- **clusterRole**—the role of the server in an XHA pair, either primary or secondary.
- **replicationRole**—the role of the server in database replication, either primary or secondary.
- **replicationStatus**—the status of the replication service on the Proteus server.
- **entityCount**—the number of entities within the Proteus database.
- **databaseSize**—the size, in megabytes, of the Proteus database.
- **loggedInUsers**—the number of users presently logged in to Proteus.

Java API Examples

You can implement the Proteus API in Java.

Install Apache Axis

To use the Proteus API with Java, you need to install the Apache Axis implementation of SOAP. Download Apache Axis from http://ws.apache.org/axis/

Connecting to Proteus

- Connect to the API service using the static method connect(*address*) from the *ProteusAPIUtils* class. This method has two versions:
    - An HTTP version that takes one argument:

```
// To connect to the API service using HTTP:

ProteusAPI_PortType service = ProteusAPIUtils.connect( "ProteusIPAddress" );
```

- An HTTPS version that takes three arguments:

```
// To connect to the API service using HTTPS:
```

```
ProteusAPI_PortType service = ProteusAPIUtils.connect(
                                "ProteusIPAddress",
                                true,
                                "c:\\client.ks" );
```

Logging In

- Log in as an API user. Use the **login()** method to log in:

```
service.login( "apiUserName", "password" );
```

Adding Objects

- Use the **addIP4NetworkTemplate()** methods to add an IPv4 network template to the specified configuration:

```
//Create a configuration if it doesn't already exist
APIEntity config =
tmpService.getEntityByName(0,"Configuration",ObjectTypes.Configuration);
if (config==null){
APIEntity newConfig = new APIEntity();
newConfig.setName("Configuration");
newConfig.setType(ObjectTypes.Configuration);
tmpService.addEntity(0, newConfig);
}
config = tmpService.getEntityByName(0,"Configuration",ObjectTypes.Configuration);
Long configID = config.getId();

EntityProperties props1 = new EntityProperties();
int gatewayOffset1=1;//1 would be x.x.x.1, -1 would be x.x.x.254
String dhcp = ObjectProperties.reservedDHCPRange;
String from1 ="FROM_START";
String name1 = "reservedDHCP";
int offset1 = 2;//Starts from x.x.x.2 going upward or x.x.x.253 going down
int size1 = 25;
String
reservedDHCP=String.format("{%s,%d,%d,%s,%s,0}",dhcp,offset1,size1,from1,name1);
props1.addProperty(ObjectProperties.gateway, gatewayOffset1 + "");
props1.addProperty(ObjectProperties.reservedAddresses, reservedDHCP);
String propString1 = props1.getPropertiesString();
tmpService.addIP4NetworkTemplate(configID, "dhcp reserved", propString1);
```

```
EntityProperties props2 = new EntityProperties();
int gatewayOffset2=(-1);//Gateway will be at x.x.x.254
String block= ObjectProperties.reservedBlock;
String from2 = "FROM_END";
String name2 = "reservedBlock";
int offset2 = 4;//Starts from x.x.x.251 going downward
int size2 = 10;
String
reservedBLOCK=String.format("{%s,%d,%d,%s,%s}",block,offset2,size2,from2,name2);
props2.addProperty(ObjectProperties.gateway, gatewayOffset2 + "");
props2.addProperty(ObjectProperties.reservedAddresses, reservedBLOCK);
String propString2 = props2.getPropertiesString();
tmpService.addIP4NetworkTemplate(configID, "block reserved", propString2);

//you can add multiple reserved ranges to a template.
EntityProperties props = new EntityProperties();
props.addProperty(ObjectProperties.gateway, gatewayOffset1 + "");
props.addProperty(ObjectProperties.reservedAddresses,
reservedDHCP+","+reservedBLOCK);
String propString = props.getPropertiesString();
tmpService.addIP4NetworkTemplate(configID, "NetworkTemplate", propString);
//You can now add a new network to any block using this template.
```

Getting Objects

- Use one of the **getEntity()** methods to get existing objects. The first object retrieved is almost always a configuration:

```
// Get an existing configuration using
// getEntityByName( long parentId, String name, String type )

APIEntity existingConfiguration = service.getEntityByName(
                            0,
                            "Configuration Name",
                            ObjectTypes.Configuration );
```

- Use the **getUserDefinedFields()** methods to get the user-defined fields information of an Object Type:

```
// Get all User Defined Fields of ObjectType ObjectTypes.Device
// server.getUserDefinedFields( String type, boolean requiredFieldsOnly );
APIUserDefinedField[] fields = service.getUserDefinedFields(ObjectTypes.Device,
false);
```

- Use the **getZonesByHint()** method to get the accessible zones information of an existing object:

```
// This will return the first 10 Zones where the logged-in user has VIEW access.
service.getZonesByHint( long parentId, 0, 10, null );


// This will return the first 10 Zones in which a user can ADD Host Records and the
zone name starts with “hewlett” and the parent of the zones is the specified
parent.
            String options = ObjectProperties.hint + "=hewlett|" +
ObjectProperties.accessRight + "=" + AccessRightValues.AddAccess + "|"+
ObjectProperties.overrideType + "=" + ObjectTypes.HostRecord;


service.getZonesByHint( long parentId, 0, 10, options );
```

For almost all object types, the **add()** and **get()** methods require *parentID*, which is the ID of the parent object. The following objects can take 0 (zero) as the *parentID*: Configuration, TagGroup, User, UserGroup, and Authenticator.

Deleting Objects

- To delete an object, call the object ID in a **delete()** method:

```
// First, add a new user:

EntityProperties userProperties = new EntityProperties();

userProperties.addProperty(
                            ObjectProperties.securityPrivilege,
                            UserSecurityPrivileges.VIEW_OTHERS_ACCESS_RIGHTS );
```

```
userProperties.addProperty(
                            ObjectProperties.historyPrivilege,
                            UserHistoryPrivileges.VIEW_HISTORY_LIST );

long userId = service.addUser(
                            "tempUser",
                            "tempPassword",
                            userProperties.getPropertiesString() );

// Now, delete the user:

service.delete( userId );
```

Logging Out

- After completing API tasks, the API user must log out:

```
service.logout();
```

> While a session will expire based on the **Session Timeout** value set on the *Configure Global Settings* page of the Proteus web interface, an explicit logout is strongly recommended to close the API user session.

Sequence of Calls in the Client

This example adds a host record to an existing zone. It demonstrates a complete session using the Proteus API from Java. This example implements the following steps:

1. Connect to the Proteus API service.
2. Log in.
3. Get the parent configuration object by name.
4. Get the parent view object by name.
5. Get the parent zone object by name (you can use the absolute name), and then retrieve its child. In this case, to retrieve example.net, we retrieve the parent **com** to find its child object **example.com**.
6. Define a host record object and add it to the parent zone object.
7. Log out.

```
import com.bluecatnetworks.proteus.api.client.java.*;

import java.rmi.RemoteException;

import com.bluecatnetworks.proteus.api.client.java.proxy.APIEntity;

import com.bluecatnetworks.proteus.api.client.java.proxy.ProteusAPI_PortType;

import com.bluecatnetworks.proteus.api.client.java.constants.*;

public class ProteusAddHostRecord {

    public static void main(String[] args) throws Exception {
        Proteusadduser test = new Proteusadduser();

        test.start();

    }

    private void start() throws Exception {

        ProteusAPI_PortType service = ProteusAPIUtils.connect( "ProteusIPAddress"
);

        service.login( "api_user", "password" );

        APIEntity existingConfiguration = service.getEntityByName( 0, "Existing
Config", ObjectTypes.Configuration );

        APIEntity existingView = service.getEntityByName(
existingConfiguration.getId(), "Existing View", ObjectTypes.View );

        long hostRecordId = service.addHostRecord( existingView.getId(),
"www.example.com", "10.0.0.6,10.0.0.8", 1, null );

        service.logout();

    }

}
```

Available Java Classes

The API includes a number of classes to facilitate the use of the methods (for example, generating the properties strings). These classes are discussed below, with the exception of UserSecurityPrivileges.java and UserHistoryPrivileges.java.

EntityProperties

Contains methods to create the properties string in many API methods:

Class	Description
EntityProperties()	Empty-argument constructor.
EntityProperties(String *propertiesString*)	Constructor that returns an instance with existing properties.

addProperty(String *attribute*, String *value*)
addProperty(String *attribute*, Long *value*)
	Two methods of adding properties.

addProperty(String attribute, String[] *values*)	Adds the property in the following format: *properties*="*value*1=<data>|*value2*<data>"
removeProperty(String *attribute*)	Removes the property.
getProperty(String *attribute*)	Returns a property or returns null if not found.

Class	Description
getPropertiesString()	Returns the API format used by API methods of properties in the following format: *attribute1=value1|attribute2=value2...*
updateProperty(String *attribute*)	Updates a property, taking a string or a number.

API Constants

A number of classes containing constants are provided as part of the API client:

Class	Description
ObjectTypes	Contains types supported by the Proteus API (used in APIEntity).
ObjectProperties	Contains all supported properties used in the *properties* argument in many API methods.
AccessRightValues	Contains access right values used by access right methods.
DNSDeploymentRoleTypes	Contains DNS deployment role types (FORWARDER, MASTER, MASTER_HIDDEN, NONE, RECURSION, SLAVE, SLAVE_STEALTH, STUB, AD_MASTER).
DHCPDeploymentRoleTypes	Contains DHCP deployment role types (NONE, MASTER, PEER).
DNSOptions DHCPClientOptions DHCP6ClientOptions DHCPServiceOptions DHCP6ServiceOptions	Contains supported DNS, DHCP client, and DHCP service option names.
DNSOptionValues	Contains constants used by some DNS options (currently used by the forwarding-policy and transfer-format DNS options.)
EnumServices	Contains ENUM number services.
IPAssignmentActionValues	Contains actions supported by the **assignIP4Address()** method.
ServiceTypes	Contains services supported by DNS and DHCP roles.
ServerCapabilityProfiles	Indicates which services can be deployed to a server.
UserSecurityPrivileges	The security privilege for a user.
UserHistoryPrivileges	The history privilege for a user.

ProteusAPIUtils

This class contains the following static methods:

Class	Description
ProteusAPI_PortType connect (*address*)	Returns a service object containing all Proteus API methods (ProteusAPI_PortType is generated from the WSDL and is provided in the client API).

Class	Description
getEnumNumberData(String *service*, String *uri*, String *comment*, long *ttl*)	Returns the properties string representing one ENUM number as used by **addEnumMethod()** method.
getEnumNumberDataLine(String *numberData*)	Returns the properties string representing many ENUM numbers as used by **addEnumMethod()** method.

Generating Java Artifacts from the WSDL

WSDL2Java can generate Java artifacts. These are stubs for service access points and the serialized class *APIEntity*. Call the WSDL with the following command:

```
java -classpath <required jars> org.apache.axis.wsdl.WSDL2Java http://<Proteus
Address>/Services/API?wsdl
```

In the above command, *required jars* includes the following JAR files:

- axis.jar
- commons-logging.jar
- commons-discovery.jar
- jaxrpc.jar
- saaj.jar
- activation.jar
- mail.jar
- wsdl4j-1.5.1.jar

Although this functionality is available as a web service, a pre-defined Java implementation is provided. The Java API included with Proteus provides a wrapper around the calls available in the WSDL to make the API easier to implement.

Perl API Examples

The Proteus API can be implemented in Perl.

The module containing the full Perl API implementation is called *API.pm*.

On the workstation running Perl, locate the **lib** directory of your Perl installation and create a directory named **Proteus**. Copy the *API.pm* file to the **lib/Proteus** directory.

Older versions of the SOAP:Lite module for Perl may create some warnings. If this is an issue, upgrade to the latest module.

All Proteus methods that take arguments need to use the SOAP::Data package to convert these argument into SOAP compatible arguments. For example, to use the **login()** method, the username argument should look like this:

```
SOAP::Data->name ('username')->
             value( $username )->
             type( 'string' )->
             attr({xmlns => ''})
```

Where *username* is the name of the argument (as described by the WSDL), *value* is the value to be passed, and *type* is the SOAP type (for example, string, int, long, or base64). The *attr* value is necessary to make the SOAP message compatible with the service.

Connecting to Proteus

- Connect to the service using the connect(*address*) function from the Service package:

```
## connect to Proteus

$service = Service->connect( "address" => 'ipAddress' );

# use "enableSSL" flag if using SSL

# $service = Service->connect("address" => 'ipAddress', "enableSSL" => 'true' );
```

Logging in

- Log in as an API user. Use the **login()** method to log in:

```
## log in and establish a session

$service->login(

SOAP::Data->name('username')->
             value('apiUserName')->
             type('string')->
             attr({xmlns => ''}),

SOAP::Data->name('password')->
             value('apiUserPassword')->
             type('string')->
             attr({xmlns => ''}) );
```

Getting, Adding, Deleting, and Updating Objects

- Use the **get()**, **add()**, **delete()**, and **update()** methods to manipulate Proteus entities. This example shows the addition of a new configuration with a shared network:

```
# Add a new configuration with a shared network

my $configuration = APIEntity->new( "id" => 0,
             "name" => "Test Configuration",
             "type" => ObjectTypes::Configuration,
             "properties" =>
             ObjectProperties::sharedNetwork."=".$existingSharedNetwork1->
             get_id()."|" );

my $configurationId = $service->addEntity( SOAP::Data->type( 'long' )->
             name( 'parentId' )->
             value( 0 )->
             attr({xmlns => ''}),

SOAP::Data->type( 'APIEntity' )->name( 'entity' )->
             value( $configuration )->
             attr({xmlns => ''}) )->
             result;

print "New Configuration id = ".$configurationId.->get_id()."\n";
```

- Use the **getUserDefinedFields()** method to find the user-defined fields with its settings and values in Proteus. For example:

```
my @udfs= $service->getUserDefinedFields( SOAP::Data->type( 'string' )->name(
'type' )->value( ObjectTypes::Device )->attr({xmlns => ''}),
     SOAP::Data->type( 'boolean' )->name( 'requiredFieldsOnly' )->value( 'false' )
     ->attr({xmlns => ''}) )
           ->valueof('//getUserDefinedFieldsResponse/return/item');

print"number of fields=".@udfs."\n";

     for my $eachUDF ( @udfs )
     {
           my $udf = Service->blessAPIUserDefinedField( "object" => $eachUDF );
           print"Object----------------------\n";
           print $udf->get_name()."\n";
           print "Name=".$udf->get_name()."\n";
           print "DisplayName=".$udf->get_displayName()."\n";
           print "Type=".$udf->get_type()."\n";
           print "defaultValue=".$udf->get_defaultValue()."\n";
           print "Validator Properties=".$udf->get_validatorProperties()."\n";
           print "PredefinedValues=".$udf->get_predefinedValues()."\n";
           print "Required=".$udf->get_required()."\n";
           print "Hide from search=".$udf->get_hideFromSearch()."\n";
           print "Radio=".$udf->get_renderAsRadioButton()."\n";
     }
```

For almost all object types, the **add()** and most **get()** methods require *parentID*, which is the ID of the parent object. The following objects can take 0 (zero) as the *parentID*: Configuration, TagGroup, User, UserGroup, and Authenticator.

Logging out

- After completing API tasks, the API user must log out.

```
$service->logout();
```

While a session will expire based on the **Session Timeout** value set on the *Configure Global Settings* page of the Proteus web interface, an explicit logout is strongly recommended to close the API user session.

Change to Perl Syntax in Proteus v3.1

The WSDL generated by Proteus 3.1 has changed, resulting in a change to the way in which you return values from several methods. The following methods are affected by this change:

- **getAccessRightsForEntity()**
- **getAccessRightsForUser()**
- **getDeploymentRoles()**
- **getEntities()**
- **getEntitiesByName()**
- **getLinkedEntities()**
- **searchByCategory()**
- **searchByObjectTypes()**

Previously, you could return values with the syntax shown in this example:

```
my @MACAddresses = $service->getLinkedEntities(
  SOAP::Data->type( 'long' )
            ->name( 'entityId' )
            ->value( $parentId )
            ->attr({xmlns => ''}),
  SOAP::Data->type( 'string' )
            ->name( 'type' )
            ->value( ObjectTypes::IP4Address )
            ->attr({xmlns => ''}),
  SOAP::Data->type( 'int' )
            ->name( 'start' )
            ->value( 0 )
            ->attr({xmlns => ''}),
  SOAP::Data->type( 'int' )
            ->name( 'count' )
            ->value( 100 )
            ->attr({xmlns => ''}) )
->valueof( '//getLinkedEntitiesResponse/result/value' );
```

The path in **`bold text`** has changed. The path in the `valueof` line should now be **`/return/item`**:

```
->valueof( '//getLinkedEntitiesResponse/return/item' );
```

> You need to update any existing Perl scripts to use the new **/return/item** path in these methods.

The following table lists the syntax for the `valueof` line of each affected method:

Method	valueof Path
getAccessRightsForEntity()	`->valueof( '//getAccessRightsForEntityResponse/return/item' );` For more information, see *Get Access Rights for Entity* on page 189.
getAccessRightsForUser()	`->valueof( '//getAccessRightsForUserResponse/return/item' );` For more information, see *Get Access Rights for User* on page 189.
getDeploymentRoles()	`->valueof( '//getDeploymentRolesResponse/return/item' );` For more information, see *Get Deployment Roles for DNS and IP Address Space Objects* on page 169.
getEntities()	`->valueof( '//getEntitiesResponse/return/item' );` For more information, see *Get Entities* on page 49.
getEntitiesByName()	`->valueof( '//getEntitiesByNameResponse/return/item' );` For more information, see *Get Entities by Name* on page 51.
getLinkedEntities()	`->valueof( '//getLinkedEntitiesResponse/return/item' );` For more information, see *Get Linked Entities* on page 56.
searchByCategory()	`->valueof( '//searchByCategoryResponse/return/item' );` For more information, see *Search by Category* on page 50.
searchByObjectTypes()	`->valueof( '//searchByObjectTypesResponse/return/item' );` For more information, see *Search by Object Types* on page 51.

Chapter 3

API Object Methods

This chapter lists the methods available in the Proteus API. Some of the generic methods include implementation examples in Java and Perl: the others are either described in pseudocode or are extended from generic methods through passing field values.

> ! Proteus API does not validate the user-defined fields with a pre-defined set of values when adding an object even though the Require Value property of the UDFs is set.

- *Generic Methods* on page 48
- *IPAM* on page 61
- *DHCP* on page 102
- *DNS* on page 126
- *TFTP* on page 158
- *Servers and Deployment* on page 160
- *Proteus Objects* on page 182

Generic Methods

Many of the object types listed below use the **update()**, **delete()**, and **get()** methods. While some objects may have specific **get()** methods, the generic methods described here are required in many Proteus API scripts.

Getting Objects

There are two generic methods for getting entity values:

- Get entities by name
- Get entities by ID

Get Entity by Name

getEntityByName() returns objects from the database referenced by their **name** field.

```
APIEntity getEntityByName( long parentId, String name, String type )
```

Parameters

parentId—the ID of the target object’s parent object.

name—the name of the target object.

type—the type of object returned by the method. This string must be one of the constants listed in *Object Types* on page 220.

Output/Response

Returns the requested object from the database.

Related Methods

- *Get Entity by ID* on page 48
- *Get Entities* on page 49
- *Get Entities by Name* on page 51
- *Get Linked Entities* on page 56
- *Get Entities by Name Using Options* on page 52

Get Entity by ID

getEntityById() returns objects from the database referenced by their database ID.

```
APIEntity getEntityById( long id )
```

Parameters

id—the object ID of the target object.

Output/Response

Returns the requested object from the database with its properties fields populated. For more information about the available options, please refer to *IPv4Objects* on page 262 in the *Property*

Options Reference section.

Related Methods

- *Get Entity by Name* on page 48
- *Get Entities* on page 49
- *Get Entities by Name* on page 51
- *Get Linked Entities* on page 56
- *Get Entities by Name Using Options* on page 52

Get Entities

getEntities() returns an array of child objects for a given *parentId* value. Objects returned in the array do not have their **properties** field set. They need to be called individually using the **getEntityById()** method to populate the **properties** field.

```
APIEntity[] getEntities( long parentId, String type, int start, int count )
```

Using **getEntities()** to search users will return all users existing in Proteus. Use **getLinkedEntities()** or **linkEntities()** to search users under a specific user group.

Parameters

parentId—the object ID of the target object's parent object.

type—the type of object returned. This must be one of the constants listed in *Object Types* on page 220.

start—indicates where in the list of objects to start returning objects. The list begins at an index of 0.

count—indicates the maximum number of child objects to return.

Output/Response

Returns an array of the requested objects from the database without their properties fields populated, or returns an empty array.

The Perl syntax for returning a value with the method has changed in Proteus version 3.1. For more information, see *Change to Perl Syntax in Proteus v3.1* on page 44.

Related Methods

- *Get Entity by Name* on page 48
- *Get Entity by ID* on page 48
- *Get Entities by Name* on page 51
- *Get Linked Entities* on page 56
- *Get Entities by Name Using Options* on page 52

Get Parent

getParent returns the parent entity of a given entity.

```
APIEntity[] getParent( long entityId )
```

Parameters

long entityId—the entity Id.

Output/Responses

Returns the APIEntity for the parent entity with its properties fields populated. For more information about the available options, please refer to *IPv4Objects* on page 262 in the *Property Options Reference* section.

Related Methods

- *Get Entity by Name* on page 48
- *Get Entity by ID* on page 48
- *Get Entities by Name* on page 51
- *Get Linked Entities* on page 56

Searching for and Retrieving Entities

Search by Category

searchByCategory() returns an array of entities by searching for keywords associated with objects of a specified object category.

```
APIEntity[] searchByCategory ( String keyword, String category, int start, int
count )
```

Parameters

keyword—the search keyword string. This value cannot be null or empty.

category—the entity category to be searched. This must be one of the entity categories listed in *Entity Categories* on page 223.

start—indicates where in the list of returned objects to start returning objects. The list begins at an index of 0. This value cannot be null or empty.

count—the maximum number of objects to return. The default value is 10. This value cannot be null or empty.

Output/Response

Returns an array of entities matching the keyword text and the category type, or returns an empty array.

The Perl syntax for returning a value with the method has changed in Proteus version 3.1. For more information, see *Change to Perl Syntax in Proteus v3.1* on page 44.

Related Methods

- *Search by Object Types* on page 51
- *Get Entities by Name* on page 51
- *Get MAC Address* on page 53

Search by Object Types

searchByObjectTypes() returns an array of entities by searching for keywords associated with objects of a specified object type. You can search for multiple object types with a single method call.

```
APIEntity[] searchByObjectTypes ( String keyword, String types, int start, int
count )
```

Parameters

keyword—the search keyword string. This value cannot be null or empty.

types—the object types for which to search, specified in the following format:

```
"type1[,type2…]"
```

The object type must be one of the types listed in *Object Types* on page 220.

start—indicates where in the list of returned objects to start returning objects. The list begins at an index of 0. This value cannot be null or empty.

count—the maximum number of objects to return. The default value is 10. This value cannot be null or empty.

Output/Response

Returns an array of entities matching the keyword text and the object type.

The Perl syntax for returning a value with the method has changed in Proteus v3.1. For more information, see *Change to Perl Syntax in Proteus v3.1* on page 44.

Related Methods

- *Search by Category* on page 50
- *Get Entities by Name* on page 51
- *Get MAC Address* on page 53

Get Entities by Name

getEntitiesByName() returns an array of entities that match the specified parent, name, and object type.

```
APIEntity[] getEntitiesByName ( long parentId, String name, String type, int start,
int count )
```

Parameters

parentId—the object ID of the parent object of the entities to be returned.

name—the name of the entity.

type—the type of object to be returned. This value must be one of the object types listed in *Object Types* on page 220.

start—indicates where in the list of returned objects to start returning objects. The list begins at an index of 0. This value cannot be null or empty.

count—the maximum number of objects to return.

Output/Response

Returns an array of entities. The array is empty if there are no matching entities.

The Perl syntax for returning a value with the method has changed in Proteus version 3.1. For more information, see *Change to Perl Syntax in Proteus v3.1* on page 44.

Related Methods

- *Get Entity by Name* on page 48
- *Get Entity by ID* on page 48
- *Get Entities* on page 49
- *Get Linked Entities* on page 56
- *Get Entities by Name Using Options* on page 52

Get Entities by Name Using Options

getEntitiesByNameUsingOptions() returns an array of entities that match the specified name and object type. Searching behavior can be changed by using the options.

```
APIEntity[] getEntitiesByNameUsingOptions ( long parentId, String name, String
type, int start, int count, String options )
```

Parameters

parentId—the object ID of the parent object of the entities to be returned.

name—the name of the entity.

type—the type of object to be returned. This value must be one of the object types listed in *Object Types* on page 220.

start—indicates where in the list of returned objects to start returning objects. The list begins at an index of 0. This value cannot be null or empty.

count—the maximum number of objects to return.

options—a string containing options. Currently the only available option is ***ObjectProperties.ignoreCase***. By default, the value is set to *false*. Setting this option to *true* will ignore the case-sensitivity used while searching entities by name.

```
ObjectProperties.ignoreCase = [true | false]
```

Output/Response

Returns an array of entities. The array is empty if there are no matching entities.

Related Methods

- *Get Entity by Name* on page 48
- *Get Entity by ID* on page 48
- *Get Entities* on page 49
- *Get Linked Entities* on page 56

Get MAC Address

getMACAddress() returns an APIEntity for a MAC address.

```
APIEntity getMACAddress ( long configurationId, String macAddress )
```

Parameters

configurationId—the object ID of the configuration in which the MAC address is located.

macAddress—the MAC address in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

Output/Response

Returns an APIEntity for the MAC address. Returns null if the MAC address does not exist. The property string of the returned entity should include the MAC address:

```
address=nn-nn-nn-nn-nn-nn|
```

If the MAC address is in a MAC pool, the property string includes the MAC pool information:

```
macPool=macPoolName|
```

Related Methods

- *Search by Category* on page 50
- *Search by Object Types* on page 51
- *Get Entities by Name* on page 51

Updating Objects

Updating an object involves two steps:

1 Building the object or parameter string used to update the object.

2 Performing the update.

Update

All entity update statements follow this format:

```
void update( APIEntity entity )
```

Parameters

entity—the actual API entity passed as an entire object that has its mutable values updated.

- Modified behavior for User-defined fields in the **update()** method:
  - **To remove existing UDF values**—commit the **update()** method with empty UDF value. If the UDF parameter is set to *mandatory*, the method fails as the UDF parameter cannot be empty.
  - **To update UDF values**—commit the **update()** method with the new UDF value. If you do not want to update the existing value, leave the UDF parameter and its value unchanged.
  - If the UDF parameter is set to *mandatory* and has a default value, committing the **update()** method with an empty UDF value will take the default value.

Related Methods

- *Delete with Options* on page 56

In this example, an existing shared network is passed to a configuration object as a parameter. After the values in the object or properties string have been set, the **update()** method modifies the value in the Proteus database. Property values can be a string, long, or integer value. Proteus uses the appropriate method to process the data for that property.

This example uses Java to return a managed server as an APIEntity, get the properties for the server, add a **connected** property with the value **true**, set the properties for the server, and then update the server.

```
APIEntity server = service.getEntityByName(config.getId(), serverName,
ObjectTypes.Server);

EntityProperties props = new EntityProperties(server.getProperties());

props.addProperty(ObjectProperties.connected, "true");

server.setProperties(props.getPropertiesString());

service.update(server);
```

This example uses Perl to update an external host record:

```
my $externalHostRecord = $service->getEntityById( SOAP::Data->type( 'long' )->
                            name( 'id' )->
                            value( $externalHostRecordId )->
                            attr({xmlns => ''}) ) ->result;

$externalHostRecord = Service->blessAPIEntity( "object" => $externalHostRecord );

$externalHostRecord->set_name( "external2.host2.com" );

$service->update( SOAP::Data->type( 'APIEntity' )->
                            name( 'entity' )->
                            value( $externalHostRecord )-
>                            attr({xmlns => ''}) );
```

Update with Options

This method is currently used for CName, MX and SRV records, and the option is only applicable to these types.

updateWithOptions() updates objects requiring a certain behavior that is not covered by the regular **update()** method.

```
void updateWithOptions ( APIEntity entity, String options )
```

Parameters

entity—the actual API entity to be updated.

options—a string containing the update options. Currently, only one option is supported:

```
linkToExternalHostRecord=boolean|
```

- If *true*, update will search for the external host record specified in *linkedRecordName* even if a host record with the same exists under the same DNS View. If the external host record is not present, it will throw an exception.
- If *false*, update will search for the host record specified in *linkedRecordName*.

Output/Response

None.

Related Methods

- *Delete* on page 56

Deleting Objects

Delete

To delete an object using the generic **delete()** method, pass the entity ID from the database identifying the object to be deleted:

```
void delete( long ObjectId )
```

Parameters

ObjectId—the ID for the object to be deleted.

Output/Response

None.

Related Methods

- *Delete with Options* on page 56

Delete with Options

deleteWithOptions() deletes objects that have options associated with their removal. This method currently works only with the deletion of dynamic records from the Proteus database. When deleted, dynamic records present the option of not dynamically deploying to Adonis.

```
void deleteWithOptions ( long objectId, String options )
```

Parameters

objectId—the object ID of the object to delete. This must be the object ID of a resource record.

options—a string containing the delete options. Currently, only one delete option is supported:

```
noServerUpdate=boolean|
```

Output/Response

None.

Related Methods

- *Deleting Objects* on page 56

Linked Entities

Get Linked Entities

getLinkedEntities() returns an array containing the entities linked to a specified entity.

```
APIEntity[] getLinkedEntities ( long entityId, String type, int start, int count )
```

Parameters

entityId—the object ID of the entity for which to return linked entities.

type—the type of entity for which to return linked entities. This value must be one of the types listed in *Object Types* on page 220.

- When specifying a host record as the **entityId**, you must use **RecordWithLink** for the type. This returns any CNAME, MX, or SRV records linked to the specified host record.
- When specifying a MAC address as the **entityId**, this method returns the IPv4 address associated with the MAC address. When appropriate, **leaseTime** and **expiryTime** information also appears in the returned properties string.

start—indicates where in the list of returned objects to start returning objects. The list begins at an index of 0. This value cannot be null or empty.

count—the maximum number of objects to return.

Output/Response

Returns an array of entities. The array is empty if there are no linked entities.

The Perl syntax for returning a value with the method has changed in Proteus version 3.1. For more information, see *Change to Perl Syntax in Proteus v3.1* on page 44.

Related Methods

- *Link Entities* on page 57
- *Unlink Entities* on page 58
- *Get Entity by Name* on page 48
- *Get Entity by ID* on page 48
- *Get Entities* on page 49
- *Get Entities by Name* on page 51

Link Entities

linkEntities() establishes a link between two specified Proteus entities. This method works on the following types of objects and links:

Type of entity for entity1Id	Type of entity for entity2Id	Result	Supported Properties
Any entity	Tag	Links the tag to the entity.	None
Tag	Any entity	Links the entity to the tag.	None
MACPool	MACAddress	Links the MAC address to the MAC pool.	None
MACAddress	MACPool	Links the MAC pool to the MAC address.	None
User	UserGroup	Links the user group to the user.	None
UserGroup	User	Links the user to the user group.	None

```
void linkEntities ( long entity1Id, long entity2Id, String properties )
```

Parameters

entity1Id—the object ID of the first entity in the pair of linked entities.

entity2Id—the object ID of the second entity in the pair of linked entities.

properties—adds object properties, including user-defined fields.

Output/Response

None.

Related Methods

- *Unlink Entities* on page 58
- *Get Linked Entities* on page 56

Unlink Entities

unlinkEntities() removes the link between two specified Proteus entities. This method works on the following types of objects and links:

Type of entity for entity1Id	Type of entity for entity2Id	Result
Any entity	Tag	Removes the tag linked to the entity.
Tag	Any entity	Removes the entities from the object tag.
MACPool	MACAddress	Removes the MAC address from the MAC pool.
MACAddress	MACPool	Removes the MAC pool from the MAC address.
User	UserGroup	Removes the user group from the user.
UserGroup	User	Removes the user from the user group.

```
void unlinkEntities ( long entity1Id, long entity2Id, String properties )
```

Parameters

entity1Id—the object ID of the first entity in the pair of linked entities.

entity2Id—the object ID of the second entity in the pair of linked entities.

properties—adds object properties, including user-defined fields.

Output/Response

None.

Related Methods

- *Link Entities* on page 57
- *Get Linked Entities* on page 56

User-defined Fields

You can add user-defined fields to any Proteus object type; these fields are available on all of the object adding and editing forms. A user-defined field can include several enforced data types and can be validated against a complex set of criteria. Any reasonable number of user-defined fields can be added to an object type to track data according to your data schema requirements.

Existing user-defined fields for objects can be set and updated through the API calls. Values for these fields can be set in the **properties** parameter where they are noted the same as any other object field. If a value is set or updated for a non-existent user-defined field, an exception is thrown. These are passed as name-value pairs, and multiple user-defined fields are separated by a **|** (pipe) character.

Getting User-defined Fields

Get User-defined Field

getUserDefinedFields() returns the user-defined fields information.

```
public APIUserDefinedField[] getUserDefinedFields ( String type, boolean
requiredFieldsOnly )
```

Parameters

type—the type of the user-defined fields. This must be one of the constants listed in ***Object Types*** on page 220.

requiredFieldsOnly—specifies whether all user-defined fields of the object type will be returned or not. If set to true, only required fields will be returned.

Output/Response

Returns the user-defined fields information.

Update Bulk User-defined Field

updateBulkUdf() updates values of various UDFs for different objects.

```
public byte[] updateBulkUdf ( byte[] data, String properties )
```

Parameters

data—the file to be used to update UDFs. The file is passed to Proteus as a byte array that is the stream of the CSV file contents. The file must follow the following pattern:

- **EntityId**—the object ID of the entity on which the UDF needs to be updated. This must be entered into the first column.
- **UDFName**—the actual name of the UDF that needs to be updated. This must be entered into the second column.

- **newUDFValue**—the new value of the UDF which needs to be updated on the entity. This must be entered into the third column.

> - The file format should be CSV.
> - The file should not contain any header.
> - The data should start from the first line.
> - To include any special characters, users need to escape the data.

properties—reserved for future use.

Output/Response

Returns a CSV file containing the following information:

- **LineNumber**—the respective line number in the input CSV file. This appears in the first column in the output CSV file.

FailureMessage—the reason for the failure identified by the system. This appears in the second column in the output CSV file.

> An empty CSV file will be returned when all the rows in the input CSV file were processed successfully.

IPAM

The IP core contains information about network structures or allocation blocks and static and dynamic allocations. This information is integrated with the DNS core to keep the DNS space current with the IP networks that it represents. DHCP configuration is modeled on the allocation blocks in the Proteus IP core and is kept current by real-time feedback from managed servers. Dynamic DNS changes, such as address allocations, from Active Directory and other updating systems are sent to Proteus in real time, showing administrators that an automated process made a configuration change.

IPv4 Blocks

An IPv4 block is a group of IPv4 addresses that is separated from a larger network by subnetting. Addresses within a block cannot be routed until they have been allocated into a network. Blocks can be added and returned by IP range or CIDR notation. You can specify default DNS domains for IPv4 blocks with the **defaultDomains** property. To add a single default domain, specify the object ID for the required domain. To add multiple default domains, specify the object IDs for multiple domains as a comma-delimited list of domain object IDs.

Add IPv4 Block by CIDR

addIP4BlockByCIDR() adds a new IPv4 Block using CIDR notation.

```
long addIP4BlockByCIDR( long parentId, String CIDR, String properties )
```

Parameters

parentId—the object ID of the target object’s parent object.

CIDR—the CIDR notation defining the block (for example, 10.10/16).

properties—a string containing options. For more information about the available options, please refer to *IPv4Objects* on page 262 in the *Property Options Reference* section.

Output/Response

Returns the object ID for the new IPv4 block.

Related Methods

- *Add IPv4 Block by Range* on page 62
- *Add Parent Block* on page 62
- *Get IPv4 Block by CIDR* on page 64
- *Get IPv4 Block by Range* on page 65
- *Merge Blocks with Parent* on page 66
- *Merge Selected Blocks or Networks* on page 66
- *Update IPv4 Block* on page 67
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

Add IPv4 Block by Range

addIP4BlockByRange() adds a new IPv4 block defined by an address range.

```
long addIP4BlockByRange( long parentId, String start, String end, String properties
)
```

Parameters

parentId—the object ID of the target object’s parent object.

start—an IP address defining the lowest address or start of the block.

end—an IP address defining the highest address or end of the block.

properties—a string containing options. For more information about the available options, please refer to *IPv4Objects* on page 262 in the *Property Options Reference* section.

Output/Response

Returns the object ID for the new IPv4 block.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add Parent Block* on page 62
- *Get IPv4 Block by CIDR* on page 64
- *Get IPv4 Block by Range* on page 65
- *Merge Blocks with Parent* on page 66
- *Merge Selected Blocks or Networks* on page 66
- *Update IPv4 Block* on page 67
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

Add Parent Block

addParentBlock() creates an IPv4 block from a list of IPv4 blocks or networks. All blocks and networks must have the same parent but it does not need to be contiguous.

```
void addParentBlock ( long[] blockOrNetworkIDs )
```

Parameters

blockOrNetworkIDs—an array containing the object IDs of IPv4 blocks or networks.

Output/Response

Returns the object ID for the new IPv4 parent block.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add IPv4 Block by Range* on page 62
- *Add Parent Block with Properties* on page 63
- *Get IPv4 Block by Range* on page 65
- *Merge Blocks with Parent* on page 66
- *Merge Selected Blocks or Networks* on page 66
- *Update IPv4 Block* on page 67
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

Add Parent Block with Properties

addParentBlockWithProperties() creates an IPv4 block with a name from a list of IPv4 blocks or networks. All blocks and networks must have the same parent but it does not need to be contiguous.

```
public long addParentBlockWithProperties ( long[] blockOrNetworkIDs, String
properties )
```

Parameters

blockOrNetworkIDs—an array containing the object IDs of IPv4 blocks or networks.

properties—a string containing the following option:

- **name**—the name of the new IPv4 block to be created.

Output/Response

Returns the object ID for the new IPv4 parent block.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add IPv4 Block by Range* on page 62
- *Add Parent Block* on page 62
- *Get IPv4 Block by Range* on page 65
- *Merge Blocks with Parent* on page 66
- *Merge Selected Blocks or Networks* on page 66
- *Update IPv4 Block* on page 67
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

Get IP Range by IP Address

getIPRangedByIP() returns the IPv4 Block containing the specified IPv4 address. Use this method to find the Configuration, IPv4 Block, IPv4 Network, or DHCP Range containing a specified address. You can specify the type of object to be returned, or you can leave the type of object null to find the most direct container for the object.

```
APIEntity getIPRangedByIP ( long containerId, String type, String address )
```

Parameters

containerId—the object ID of the container in which the IPv4 address is located. This can be a Configuration, IPv4 Block, IPv4 Network, or DHCP Range. When you do not know the block, network, or range in which the address is located, specify the configuration.

type—the type of object containing the IPv4 or IPv6 address. Specify `ObjectTypes.IP4Block`, `ObjectTypes.IP4Network`, or `ObjectTypes.DHCP4Range` to find the block, network, or range containing the IPv4 address. Specify `null` to return the most direct container for the IPv4 address.

address—an IPv4 address.

Output/Response

Returns an APIEntity for the object containing the specified address. If no object is found, returns `null`. If `ObjectTypes.IP4Block`, `ObjectTypes.IP4Network`, or `ObjectTypes.DHCP4Range` is specified as the **type** parameter, returns an object of the specified type. If `null` is specified as the **type** parameter, returns the most direct container for the IPv4 address.

Related Methods

- *Get IPv4 Block by Range* on page 65
- *Get Entity by Name* on page 48
- *Get Entity by ID* on page 48
- *Get Entities* on page 49

Get IPv4 Block by CIDR

getEntityByCIDR() returns an IPv4 Block by calling the block using CIDR notation.

```
APIEntity getEntityByCIDR( long parentId, String cidr, String type )
```

Parameters

parentId—the object ID of the target object’s parent object.

CIDR—CIDR notation defining the block to be returned (for example, 10.10/16).

type—the type of object returned: IP4Block. This must be one of the constants listed in *Object Types* on page 220.

Output/Response

Returns the specified IPv4 block object from the database.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add IPv4 Block by Range* on page 62
- *Add Parent Block* on page 62
- *Get IPv4 Block by Range* on page 65
- *Merge Blocks with Parent* on page 66
- *Merge Selected Blocks or Networks* on page 66
- *Update IPv4 Block* on page 67
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

Get IPv4 Block by Range

getEntityByRange() returns an IPv4 Block by calling the block using its address range.

```
APIEntity getEntityByRange( long parentId, String address1, String address2, String
type )
```

Parameters

parentId—the object ID of the target object’s parent object.

address1—an IP address defining the lowest address or start of the block.

address2—an IP address defining the highest address or end of the block.

type—the type of object returned: IP4Block. This must be one of the constants listed in *Object Types* on page 220.

Output/Response

Returns the requested IPv4 block object from the database.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add IPv4 Block by Range* on page 62
- *Add Parent Block* on page 62
- *Get IPv4 Block by CIDR* on page 64
- *Merge Blocks with Parent* on page 66
- *Merge Selected Blocks or Networks* on page 66
- *Update IPv4 Block* on page 67
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

Merge Blocks with Parent

mergeBlocksWithParent() merges specified IPv4 blocks into a single block. The blocks must all have the same parent and must be contiguous. Blocks whose parent object is the configuration cannot contain networks.

```
void mergeBlocksWithParent ( long[] blockIDs )
```

Parameters

blockIDs—an array containing a list of IPv4 block IDs.

Output/Response

None.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add IPv4 Block by Range* on page 62
- *Add Parent Block* on page 62
- *Get IPv4 Block by CIDR* on page 64
- *Get IPv4 Block by Range* on page 65
- *Merge Selected Blocks or Networks* on page 66
- *Update IPv4 Block* on page 67
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

Merge Selected Blocks or Networks

mergeSelectedBlocksOrNetworks() merges specified IPv4 blocks or IPv4 networks into a single IPv4 block or IPv4 network. The list of objects to be merged must all be of the same type (for example, all blocks or all networks). The objects must all have the same parent and must be contiguous.

```
void mergeSelectedBlocksOrNetworks ( long[] blockOrNetworkIds, long
blockOrNetworkToKeep )
```

Parameters

blockOrNetworkIds—an array containing a list of IPv4 block or network IDs.

blockOrNetworkToKeep—the ID of the IPv4 block or IPv4 network that will retain its identity after the merge.

Output/Response

None.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add IPv4 Block by Range* on page 62
- *Add Parent Block* on page 62
- *Get IPv4 Block by CIDR* on page 64
- *Get IPv4 Block by Range* on page 65
- *Merge Blocks with Parent* on page 66
- *Update IPv4 Block* on page 67
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

Update IPv4 Block

An IPv4 block's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add IPv4 Block by Range* on page 62
- *Get IPv4 Block by CIDR* on page 64
- *Get IPv4 Block by Range* on page 65
- *Merge Blocks with Parent* on page 66
- *Merge Selected Blocks or Networks* on page 66
- *IPv4 Block Generic Methods* on page 67
- *Resize Range* on page 93

IPv4 Block Generic Methods

IPv4 blocks can be deleted using the generic **delete()** method. For more information, see *Deleting Objects* on page 56.

Related Methods

- *Add IPv4 Block by CIDR* on page 61
- *Add IPv4 Block by Range* on page 62
- *Get IPv4 Block by CIDR* on page 64
- *Get IPv4 Block by Range* on page 65
- *Merge Blocks with Parent* on page 66
- *Merge Blocks with Parent* on page 66
- *Update IPv4 Block* on page 67
- *Resize Range* on page 93

IPv4 Networks

An IPv4 network is an object that attaches to a router interface that routes directly to individual IP addresses. An IPv4 network is therefore a group of IPv4 addresses that can be routed. An IPv4 network always has a network or a block as its parent object in Proteus. Networks can be added and returned both by IP range and CIDR notation. The next available network can also be returned and allocated.

You can specify default DNS domains for IPv4 networks with the **defaultDomains** property. To add a single default domain, specify the object ID for the required domain. To add multiple default domains, specify the object IDs for multiple domains as a comma-delimited list of domain object IDs.

Add IPv4 Network

addIP4Network() adds an IPv4 network using CIDR notation.

```
long addIP4Network( long blockId, String CIDR, String properties )
```

Parameters

blockId—the object ID of the new network's parent IPv4 block.

CIDR—the CIDR notation defining the network (for example, 10.10.10/24).

properties—a string containing options. For more information about the available options, please refer to *IPv4Objects* on page 262 in the *Property Options Reference* section.

Output/Response

Returns the object ID for the new IPv4 network.

Related Methods

- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Get Next Available Network* on page 72
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Reconciliation Policy* on page 76
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

Get IPv4 Range by IP Address

getIPRangedByIP() returns the IPv4 Network containing the specified IPv4 address. Use this method to find the Configuration, IPv4 Block, IPv4 Network, or DHCP Range containing a specified address. You can specify the type of object to be returned, or you can leave the type of object null to find the most direct container for the object.

```
APIEntity getIPRangedByIP( long containerId, String type, String address )
```

Parameters

containerId—the object ID of the container in which the IPv4 address is located. This can be a Configuration, IPv4 Block, IPv4 Network, or DHCP Range. When you do not know the block, network, or range in which the address is located, specify the configuration.

type—the type of object containing the IPv4 address. Specify `ObjectTypes.IP4Block`, `ObjectTypes.IP4Network`, or `ObjectTypes.DHCP4Range` to find the block, network, or range containing the IPv4 address. Specify `null` to return the most direct container for the IPv4 address.

address—an IPv4 address.

Output/Response

Returns an APIEntity for the object containing the specified address. If no object is found, returns `null`. If `ObjectTypes.IP4Block`, `ObjectTypes.IP4Network`, or `ObjectTypes.DHCP4Range` is specified as the **type** parameter, returns an object of the specified type. If `null` is specified as the **type** parameter, returns the most direct container for the IPv4 address.

Related Methods

- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Get Entity by Name* on page 48
- *Get Entity by ID* on page 48
- *Get Entities* on page 49

Get IPv4 Network by CIDR

getEntityByCIDR() returns an IPv4 Network by calling it using CIDR notation.

```
APIEntity getEntityByCIDR( long parentId, String cidr, String type )
```

Parameters

parentId—the object ID of the network's parent object.

CIDR—CIDR notation defining the network (for example, 10.10.10/24).

type—the type of object returned: IP4Network. This must be one of the constants listed in *Object Types* on page 220.

Output/Response

Returns the specified IPv4 network object from the database.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by Range* on page 71
- *Get Next Available Network* on page 72
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Reconciliation Policy* on page 76
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

Get IPv4 Network by Hint

getIP4NetworksByHint() returns an array of IPv4 networks found under a given container object. The networks can be filtered by using ObjectProperties.hint, ObjectProperties.accessRight, and ObjectProperties.overrideType options.

```
public APIEntity[] getIP4NetworksByHint( long containerId, int start, int count,
String options )
```

Parameters

containerId—the object ID for the container object. It can be the object ID of any object in the parent object hierarchy. The highest parent object is the configuration level.

start—indicates where in the list of objects to start returning objects. The list begins at an index of 0.

count—indicates the maximum number of child objects that this method will return.

options—a string containing options. The Option names available in the ObjectProperties are *ObjectProperties.hint*, *ObjectProperties.accessRight*, and *ObjectProperties.overrideType*. Multiple options can be separated by a | (pipe) character. For example:

```
hint=ab|overrideType=HostRecord|accessRight=ADD
```

The values for the *ObjectProperties.hint* option can be the prefix of the IP address for a network or the name of a network.

- **Example 1**—will match networks that have the network ID starting with 192.168. For example, 192.168.0.0/24 or 192.168.1.0/24.

```
String options = ObjectProperties.hint + “=198.168|”
```

- **Example 2**—will match networks that have a name starting with “abc”. For example, “abc”, “abc123” or “abcdef”.

```
String options = ObjectProperties.hint + “=abc|”
```

Matching networks to a network ID (Example 1) will take precedence over matching networks to a name (Example 2).

The values for the *ObjectProperties.accessRight* and *ObjectProperties.overrideType* options must be one of the constants listed in *Access Right Values* on page 203 and *Object Types* on page 220. For example:

```
String options = ObjectProperties.accessRight + "=" + AccessRightValues.AddAccess +
"|"+ ObjectProperties.overrideType + "=" + ObjectTypes.HostRecord;
```

Output/Response

Returns an array of IPv4 networks based on the input argument without their properties fields populated, or returns an empty array if containerId is invalid. If no access right option is specified, the View access level will be used by default.

Get IPv4 Network by Range

getEntityRange() returns an IPv4 Network by calling it using its address range.

```
APIEntity getEntityByRange( long parentId, String address1, String address2, String
type )
```

Parameters

parentId—the object ID of the network’s parent object.

address1—an IP address defining the lowest address or start of the network.

address2—an IP address defining the highest address or end of the network.

type—the type of object returned: IP4Network. This must be one of the constants listed in *Object Types* on page 220.

Output/Response

Returns the specified IPv4 network object from the database.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get Next Available Network* on page 72
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Reconciliation Policy* on page 76
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

Get Next Available Network

getNextAvailableIP4Network() returns the object ID for the next available (unused) network within a configuration or block. If the next available network does not exist, you can create it.

```
long getNextAvailableIP4Network( long parentId, long size, boolean isLargerAllowed,
boolean autoCreate )
```

Parameters

parentId—the object ID of the network’s parent object.

size—the size of the network, expressed as a power of 2.

isLargerAllowed—this Boolean value indicates whether to return larger networks than those specified with the **size** parameter.

autoCreate—this Boolean value indicates whether the next available network should be created if it does not exist.

Output/Response

Returns the object ID for the existing next available IPv4 network or, if the next available network did not exist and **autoCreate** was set to true, the newly created IPv4 network.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Reconciliation Policy* on page 76
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

Get Next Available IP Range

getNextAvailableIPRange() returns the object ID for the next available (unused) block or network within a configuration or block. If the next available IP range does not exist, you can create it.

```
APIEntity getNextAvailableIPRange( long parentId, long size, String type, String
properties )
```

Parameters

parentId—the object ID of the parent object under which the next available range resides (Configuration or Block).

size—the size of the range, expressed as a power of 2.

type—the type of the range object to be fetched. Currently IPv4 block and network are supported.

Properties—the string containing the following properties and values:

- **reuseExisting—***True* or *False*. This Boolean value indicates whether to search existing empty networks to find the available IP range of specified size.
- **isLargerAllowed—***True* or *False*. This Boolean value indicates whether to return larger networks than those specified with the **size** parameter.
- **autoCreate**—*True* or *False*. This Boolean value indicates whether the next available IP range should be created in the parent object if it does not exist.
- **traversalMethod**—this parameter identifies the appropriate search algorithm to find the suitable object. The possible values are:
    - **TraversalMethodology.NO_TRAVERSAL (NO_TRAVERSAL)**—will attempt to find the next range directly under the specified parent object. It will not search through to the lower level objects.
    - **TraversalMethodology.DEPTH_FIRST (DEPTH_FIRST)**—will attempt to find the next range under the specified object by iteratively through its children one by one. After exploring the object recursively for its child ranges, it will move to the next child object.
    - **TraversalMethodology.BREADTH_FIRST (BREADTH_FIRST)**—will attempt to find the next range under the specified object by iterative levels. It will first find the range immediately below the specified parent object. If not found, then it will attempt to find the range under all the first child objects.

Output/Response

Returns the object ID for the existing next available IPv4 range or, if the next available range did not exist and **autoCreate** was set to true, the newly created IPv4 range.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Reconciliation Policy* on page 76
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

Get Next Available IP Ranges

getNextAvailableIPRanges() returns the object IDs for the next available (unused) networks within a configuration or block. If the next available IP range does not exist, you can create it.

```
public APIEntity getNextAvailableIPRanges( long parentId, long size, String type,
int count, String properties )
```

Parameters

parentId—the object ID of the parent object under which the next available range resides (Configuration or Block).

size—the size of the range, expressed as a power of 2.

type—the type of the range object to be fetched. Currently only IPv4 network is supported.

count—the number of networks to be found.

> If the number of networks count is greater than 1:
> - isLargerAllowed and traveralmethod properties will not be applicable.
> - The DEPTH_FIRST methodology will be used to search objects.

Properties—the string containing the following properties and values:

- **isLargerAllowed—***True* or *False*. This Boolean value indicates whether to return larger networks than those specified with the **size** parameter.
- **autoCreate**—*True* or *False*. This Boolean value indicates whether the next available IP range should be created in the parent object if it does not exist.
- **traversalMethod**—this parameter identifies the appropriate search algorithm to find the suitable object. The possible values are:
  - **TraversalMethodology.NO_TRAVERSAL (NO_TRAVERSAL)**—will attempt to find the next range directly under the specified parent object. It will not search through to the lower level objects.
  - **TraversalMethodology.DEPTH_FIRST (DEPTH_FIRST)**—will attempt to find the next range under the specified object by iteratively through its children one by one. After exploring the object recursively for its child ranges, it will move to the next child object.
  - **TraversalMethodology.BREADTH_FIRST (BREADTH_FIRST)**—will attempt to find the next range under the specified object by iterative levels. It will first find the range immediately below the specified parent object. If not found, then it will attempt to find the range under all the first child objects.

Output/Response

Returns the object IDs for the existing next available IPv4 range or, if the next available range did not exist and **autoCreate** was set to true, the newly created IPv4 range.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Reconciliation Policy* on page 76
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

Split IPv4 Network

splitIP4Network() splits an IPv4 network into the specified number of networks.

```
public APIEntity[] splitIP4Network ( long networkId, int numberOfParts, string
options )
```

Parameters

networkId—the database object ID of the network that is being split.

numberOfParts—the number of the networks into which the network will be split. Valid values are *2 to the power of 2 up to 1024*.

options—a string containing the following options:

- **assignDefaultGateway**—a Boolean value. If set to *true*, a gateway will be created by using the default gateway value which is the first IP address in the network. If set to *false*, no gateway will be created. The default value is *true*.
- **overwriteConflicts**—a Boolean value. If set to *true*, any conflicts within the split IPv4 network will be removed. The default value is *false*.
- **template**—a network template ID. The default value is 0 which means no network template will be used. Specify a network template ID if you wish to apply one.

Output/Response

Returns an array of networks created after splitting the network.

Update IPv4 Network

An IPv4 network's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Get Next Available Network* on page 72
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Reconciliation Policy* on page 76
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

IPv4 Network Generic Methods

IPv4 networks can be deleted using the generic **delete()** method. For more information, see *Deleting Objects* on page 56.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Get Next Available Network* on page 72
- *Update IPv4 Network* on page 75
- *Add IPv4 Reconciliation Policy* on page 76
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

Add IPv4 Reconciliation Policy

addIP4ReconciliationPolicy adds an IPv4 reconciliation policy.

```
long addIP4ReconciliationPolicy( long parentId, string name, string properties )
```

Parameters

long parentId—the object ID of the network's parent of the policy.

string name—the name of the policy.

string properties—the property string contains the following properties and values:

Property	Value
seedRouterAddress	IPv4 address - used seed router is not Default Gateway Address
snmpVersion	Constants defined in SNMPVersion which includes [v1|v2c|v3]
snmpPortNumber	An integer greater than 0
snmpCommunityString	Strings separated by comma e.g. community10,community12,community13
securityLevel	This must be one of the constants listed in *SNMPSecurityLevels* on page 226.
context	A string; This is required only when snmpVersion is v3
authenticationType	Constants defined in SNMPAuthType which includes [MD5|SHA] This is required only when securityLevel is AUTH_NOPRIV or AUTH_PRIV
authPassphrase	A string; This is required only when securityLevel is AUTH_NOPRIV or AUTH_PRIV
privacyType	A string containing the privacy encryption types AES-128 and DES (by default, DES); This is required only when securityLevel is AUTH_PRIV
privacyPassphrase	A string containing the privacy authentication password; This is required only when securityLevel is AUTH_PRIV

Property	Value
networkBoundaries	IPv4 ranges separated by comma(,). support CIDR format and IPv4 range format e.g. 10.0/8,13.0.0.1-13.0.0.126
enableLayer2Discovery	True or false
schedule	Schedule setting Format: hh:mma,dd MMM yyyy frequencyType frequency: frequencyPeriod frequencyType: EVERY or ONCE frequency: an integer greater than 0 frequencyPeriod: constant defined in TimeUnits which includes [MINUTES|HOURS|DAYS] e.g. 03:37am,31 May 2011,EVERY,6,Days
activeSataus	True or false
accpetanceCriteriaReclaim	This is used if user wants to enable automated acceptance. acceptanceCriteriaReclaim, acceptanceCriteriaUnknown, acceptanceCriteriaMismatch and view should be used together as a complete configuration. Format: timeValue, timeUnit, actionType timeValue: a integer greater than 0 timeUnit: constant defined in timeUnits which includes [MINUTES|HOURS|DAYS] actionType: constant defined in AcceptanceActionType which includes [RECONCILE, NOACTION] e.g. 10,MINUTES,RECONCILE
acceptanceCriteriaUnknown	This is used if user want to enable automated acceptance. Format: timeValue, timeUnit, actionType timeValue: a integer greater than 0 timeUnit: constant defined in timeUnits which includes [MINUTES|HOURS|DAYS] actionType: constant defined in AcceptanceActionType which includes [RECONCILE, NOACTION] e.g. 20,HOURS,NOACTION
acceptanceCriteriaMismatch	This is used if user want to enable automated acceptance. Format: timeValue, timeUnit,actionType timeValue: a integer greater than 0 timeUnit: constant defined in timeUnits which includes [MINUTES|HOURS|DAYS] actionType: constant defined in AcceptanceActionType which includes [RECONCILE, NOACTION] e.g. 30,MINUTES,RECONCILE
view	This is used if user wants to enable automated acceptance an existing view's name

Property	Value
overriddenList	IPv4 ranges separated by comma(,). support CIDR format and IP4 range format e.g. 10/16,172/16,172.25.0.2-172.25.0.18

Output/Response

Adds an IPv4 reconciliation policy.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Resize Range* on page 93

IPv4 Network Templates

IPv4 network templates allow you to create standard settings that can be applied when you create new networks. Whenever you change template settings, all networks based on the template are updated accordingly. Use network templates to standardize address assignments and DHCP options.

> Proteus API v4.0 includes the following changes to IPv4 network templates when performing Add, Update and Get operations for IPv4 Network Templates through API:
>
> - The pipe (|) separator in the sub-type value has been replaced with commas (,).
> - If the name properties contains commas (,) or back-slashes (\), it needs to be escaped with backward slash (\).

Add IPv4 Network Template

addIP4NetworkTemplate() add an IPv4 network template to the specified configuration.

```
long addIP4NetworkTemplate ( long configurationId, String name, String properties )
```

Parameters

configurationId—the object ID of the configuration in which the IPv4 template is located.

name—the name of the IPv4 network template. This value cannot be empty or null.

properties—a string defining the IPv4 network template properties. For example:

```
gateway=[gateway_offset]|reservedAddresses={type,offset,size,dirction,name}
```

Where the possible values for each parameters are:

gateway_offset—this is to specify which address to assign an IPv4 gateway. When there is a negative sign in front of the gateway offset, then the gateway is at the end of the range. For example, if the value of gateway offset is -*n*, the n^{th} IP address from the end of range will be the gateway.

type—can be either *RESERVED_BLOCK* or *RESERVED_DHCP_RANGE.*

offset—this is to specify from which address to start to assign IPv4 addresses.

size—the size of the network.

direction—can be either *FROM_START* or *FROM_END*.

name—the name of the network.

Output/Response

Returns the object ID of the new IPv4 network template.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Get Next Available Network* on page 72
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Assign or Update Template* on page 79
- *Re-apply Template* on page 81
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83

Assign or Update Template

assignOrUpdateTemplate() assigns, updates, or removes DNS zone and IPv4 network templates.

```
void assignOrUpdateTemplate ( long entityId, long templateId, String properties )
```

Parameters

entityId—the object ID of the IPv4 network to which the network template is to be assigned or updated, or the object ID of the zone to which the zone template is to be assigned or updated.

templateId—the object ID of the DNS zone template or IPv4 network template. To remove a template, set this value to 0 (zero).

properties—a string containing the following settings:

- **ObjectProperties.templateType**—specifies the type of template on which this operation is being performed. The possible values are **ObjectProperties.IP4NetworkTemplateType** (Assigning or updating IP4NetowrkTemplate on an IP4Network) and **ObjectProperties.zoneTemplateType** (Assigning or updating zoneTemplate on a DNS zone). This is mandatory.

Along with **ObjectProperties.templateType**, user can also specify the reapply mode for various properties of the template.

- For Network template, the following additional parameters can also be specified:

  - **ObjectProperties.gatewayReapplyMode**
  - **ObjectProperties.reservedAddressesReapplyMode**
  - **ObjectProperties.dhcpRangesReapplyMode**
  - **ObjectProperties.ipGroupsReapplyMode**
  - **ObjectProperties.optionsReapplyMode**
- For Zone Template, the following additional parameter can also be specified:
  - **ObjectProperties.zoneTemplateReapplyMode**

  The possible values for re-apply mode properties are:
  - **ObjectProperties.templateReapplyModeUpdate**
  - **ObjectProperties.templateReapplyModeIgnore**
  - **ObjectProperties.templateReapplyModeOverwrite**

  If the re-apply mode is not specified in the properties, the default **ObjectProperties.templateReapplyModeIgnore** mode is used.

> If you are not using Java or Perl, refer to ***Object Properties*** on page 213 for the actual values.

Java client example:

```
EntityProperties ntProp = new EntityProperties();

ntProp.addProperty( ObjectProperties.templateType,
ObjectProperties.IP4NetworkTemplateType );

ntProp.addProperty( ObjectProperties.gatewayReapplyMode,
ObjectProperties.templateReapplyModeUpdate );

ntProp.addProperty( ObjectProperties.reservedAddressesReapplyMode,
ObjectProperties.templateReapplyModeUpdate );

service.assignOrUpdateTemplate( ip4N20_26Id, networkTemplateId,
ntProp.getPropertiesString() );
```

Perl client example:

```
SOAP::Data->type( 'string' )->name( 'properties' )->

value( ObjectProperties::templateType."=".ObjectProperties::
IP4NetworkTemplateType."|".

ObjectProperties:: gatewayReapplyMode."=".ObjectProperties::
templateReapplyModeUpdate."|" )

->attr({xmlns => ''}) )->result;
```

Output/Response

None.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Get Next Available Network* on page 72
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Network Template* on page 78
- *Re-apply Template* on page 81
- *Update IPv4 Network Template Name* on page 83
- *IPv4 Network Template Generic Methods* on page 83
- *Re-apply Template* on page 81

Re-apply Template

reapplyTemplate() reapplies DNS zone and IPv4 network templates. The template must already be applied to an object before you can re-apply or remove it.

```
void reapplyTemplate ( long templateId, String properties )
```

Parameters

templateId—the object ID of the DNS zone template or IPv4 network template to be reapplied.

properties—a string containing the following settings:

1. The properties value must include **ObjectProperties.templateType** with the value of **ObjectProperties.IP4NetworkTemplateType**.
2. To re-apply the network gateway, include **ObjectProperties.gatewayReapplyMode**. This is optional.
3. If the re-apply mode is not specified in the properties, the default **ObjectProperties.templateReapplyModeIgnore** mode is used.
4. The available re-apply modes include:

   ObjectProperties.templateReapplyModeUpdate

   ObjectProperties.templateReapplyModeIgnore

   ObjectProperties.templateReapplyModeOverwrite

Java client example:

```
EntityProperties ntProp = new EntityProperties();

ntProp.addProperty( ObjectProperties.templateType,
ObjectProperties.IP4NetworkTemplateType );

ntProp.addProperty( ObjectProperties.gatewayReapplyMode,
ObjectProperties.templateReapplyModeUpdate );

ntProp.addProperty( ObjectProperties.reservedAddressesReapplyMode,
ObjectProperties.templateReapplyModeUpdate );
```

```
service.reapplyTemplate( networkTemplateId3, ntProp.getPropertiesString() );
```

Perl client example:

```
SOAP::Data->type( 'string' )->name( 'properties' )->

value( ObjectProperties::templateType."=".ObjectProperties::
IP4NetworkTemplateType."|".

ObjectProperties:: gatewayReapplyMode."=".ObjectProperties::
templateReapplyModeUpdate."|" )

->attr({xmlns => ''}) )->result;
```

Output/Response

None.

Update IPv4 Network Template Name

An IPv4 network template’s **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Get Next Available Network* on page 72
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Network Template* on page 78
- *Assign or Update Template* on page 79
- *IPv4 Network Template Generic Methods* on page 83
- *Re-apply Template* on page 81

IPv4 Network Template Generic Methods

IPv4 network templates use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add IPv4 Network* on page 68
- *Get IPv4 Network by CIDR* on page 69
- *Get IPv4 Network by Range* on page 71
- *Get Next Available Network* on page 72
- *Update IPv4 Network* on page 75
- *IPv4 Network Generic Methods* on page 75
- *Add IPv4 Network Template* on page 78
- *Assign or Update Template* on page 79
- *Re-apply Template* on page 81
- *Update IPv4 Network Template Name* on page 83

IPv4 Addresses

An address is the actual IP address leased or assigned to a member of a network. IPv4 addresses need to be assigned a particular allocation rather than simply be added. The address allocation can be checked, along with any host records that are dependent on it. The next available address can also be returned. Only addresses within an existing network can be assigned.

Assign IPv4 Address

assignIP4Address() assigns a MAC address and other properties to an IPv4 address.

```
long assignIP4Address( long configurationId, String ip4Address, String macAddress,
String hostInfo, String action, String properties )
```

Parameters

configurationId—the object ID of the configuration in which the IPv4 address is located.

ip4Address—the IPv4 address.

macAddress—the MAC address to assign to the IPv4 address. The MAC address can be specified in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

hostInfo—a string containing host information for the address, in the following format:

```
hostname,viewId,reverseFlag,sameAsZoneFlag[,
hostname,viewId,reverseFlag,sameAsZoneFlag,...]
```

Where:

hostname—the Fully Qualified Domain Name (FQDN) for the host record to be added.

viewId—the object ID of the view under which this host should be created.

reverseFlag—the flag indicating if a reverse record should be created. The possible values are *true* or *false*.

sameAsZoneFlag—the flag indicating if record should be created as same as zone record. The possible values are *true* or *false*.

The comma-separated parameters may be repeated in the order shown above. The string must *not* end with a comma.

action—this parameter must be set to one of the constants shown in *IP Assignment Action Values* on page 213.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the newly assigned IPv4 address.

Related Methods

- *Assign Next Available IPv4 Address* on page 85
- *Get IPv4 Address* on page 86
- *Check Allocation for IPv4 Address* on page 87
- *Allocate Next Available Address* on page 88
- *Get Dependent Records* on page 88
- *Update IPv4 Address* on page 89
- *IPv4 Address Generic Methods* on page 89

Assign Next Available IPv4 Address

APIEntity assignNextAvailableIP4Address() assigns a MAC address and other properties to the next available and unallocated IPv4 address within a configuration, block, or network.

```
public APIEntity assignNextAvailableIP4Address( long configurationId, long
parentId, String macAddress, String hostInfo, String action, String properties )
```

Parameters

configurationId—the object ID of the configuration in which the IPv4 address is located.

parentId—the object ID of the configuration, block, or network in which to look for the next available address.

macAddress—the MAC address to be assigned to the IPv4 address. The MAC address can be specified in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

hostInfo—a string containing host information for the address, in the following format:

```
hostname,viewId,reverseFlag,sameAsZoneFlag[,
hostname,viewId,reverseFlag,sameAsZoneFlag,...]
```

Where:

hostname—the Fully Qualified Domain Name (FQDN) for the host record to be added.

viewId—the object ID of the view under which this host should be created.

reverseFlag—the flag indicating if a reverse record should be created. The possible values are *true* or *false*.

sameAsZoneFlag—the flag indicating if record should be created as same as zone record. The possible values are *true* or *false*.

The comma-separated parameters may be repeated in the order shown above. The string must *not* end with a comma.

action—this parameter must be set to one of the constants shown in *IP Assignment Action Values* on page 213.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the newly assigned IPv4 address.

Related Methods

- *Assign IPv4 Address* on page 84
- *Get IPv4 Address* on page 86
- *Check Allocation for IPv4 Address* on page 87
- *Allocate Next Available Address* on page 88
- *Get Dependent Records* on page 88
- *Update IPv4 Address* on page 89
- *IPv4 Address Generic Methods* on page 89

Get IPv4 Address

Returns the details for an IPv4 address object.

```
APIEntity getIP4Address( long containerId, String address )
```

Parameters

containerId—the object ID for the configuration, block, or network in which this address is located.

address—the IPv4 address.

Output/Response

Returns the requested IPv4 address object from the database.

Related Methods

- *Assign IPv4 Address* on page 84
- *Assign Next Available IPv4 Address* on page 85
- *Check Allocation for IPv4 Address* on page 87
- *Allocate Next Available Address* on page 88
- *Get Dependent Records* on page 88
- *Update IPv4 Address* on page 89
- *IPv4 Address Generic Methods* on page 89

Get Next IPv4 Address

getNextIP4Address() returns the next available IP addresses under specified circumstances.

```
APIEntity getNextIP4Address( long parentId, String properties )
```

Parameters

long parentId—the network or configuration Id.

String properties—the property string contains three properties, **skip**, **offset** and **excludeDHCPRange**. The values for skip and offset must be IPv4 addresses and must appear in dotted octet notation.

- **skip**—this is optional. It is used to specify the IP address ranges or IP addresses to skip, separated by comma. A hyphen(-), not a dash is used to separate the start and end addresses.

> Do not use the skip property if the parentId is a configuration Id. If you do, an error message appears, ‘Skip is not allowed for configuration level’.

- **offset**—this is optional. This is to specify from which address to start to assign IPv4 Address.
- **excludeDHCPRange**—this specifies whether IP addresses in DHCP ranges should be excluded from assignment. The value is either **true** or **false**, default value is false.

```
skip=10.10.10.128-10.10.11.200,10.10.11.210|offset=10.10.10.100|excludeDHCPRange=true|
```

Output/Responses

Returns the IPv4 address in octet notation.

Related Methods

- *Assign Next Available IPv4 Address* on page 85
- *Get IPv4 Address* on page 86
- *Check Allocation for IPv4 Address* on page 87
- *Allocate Next Available Address* on page 88

Check Allocation for IPv4 Address

isAddressAllocated() returns the allocation information for an IPv4 DHCP allocated address:

```
boolean isAddressAllocated( long configurationId, String ipAddress, String
macAddress )
```

Parameters

configurationId—the object ID for the configuration in which the IPv4 DHCP allocated address is located.

ip4Address—the IPv4 DHCP allocated address.

macAddress—the MAC address associated with the IPv4 DHCP allocated address. The MAC address can be specified in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

Output/Response

Returns a Boolean value indicating whether the address is allocated.

Related Methods

- *Assign IPv4 Address* on page 84
- *Assign Next Available IPv4 Address* on page 85
- *Get IPv4 Address* on page 86
- *Allocate Next Available Address* on page 88
- *Get Dependent Records* on page 88
- *Update IPv4 Address* on page 89
- *IPv4 Address Generic Methods* on page 89

Allocate Next Available Address

getNextAvailableIP4Address() returns the IPv4 address for the next available (unallocated) address within a configuration, block, or network.

```
String getNextAvailableIP4Address( long parentId )
```

Parameters

parentId—the object ID for configuration, block, or network in which to look for the next available address.

Output/Response

Returns the next available IPv4 address in an existing network as a string.

Related Methods

- *Assign IPv4 Address* on page 84
- *Assign Next Available IPv4 Address* on page 85
- *Get IPv4 Address* on page 86
- *Check Allocation for IPv4 Address* on page 87
- *Get Dependent Records* on page 88
- *Update IPv4 Address* on page 89
- *IPv4 Address Generic Methods* on page 89

Get Dependent Records

> This method is deprecated. Using this method now returns an error message. Use the **getLinkedEntities()** method instead. For more information, see *Get Linked Entities* on page 56.

Returns any host resource records that have been assigned to an IP address.

```
APIEntity[] getDependentRecords( long entityId, int start, int count )
```

Parameters

entityId—the object ID for the IP address.

start—indicates where in the list of dependent records to begin returning objects. The list begins at an index of 0.

count—the maximum number of dependent records to return.

Output/Response

Returns an array of APIEntity objects representing the host records associated with the IP address. After the host records are returned, this method should be run on the returned host records to discover any dependent CNAME, MX, and SRV records.

Related Methods

- *Assign IPv4 Address* on page 84
- *Assign Next Available IPv4 Address* on page 85
- *Get IPv4 Address* on page 86
- *Check Allocation for IPv4 Address* on page 87
- *Allocate Next Available Address* on page 88
- *Update IPv4 Address* on page 89
- *IPv4 Address Generic Methods* on page 89

Update IPv4 Address

An IPv4 address's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Assign IPv4 Address* on page 84
- *Assign Next Available IPv4 Address* on page 85
- *Get IPv4 Address* on page 86
- *Check Allocation for IPv4 Address* on page 87
- *Allocate Next Available Address* on page 88
- *Get Dependent Records* on page 88
- *IPv4 Address Generic Methods* on page 89

IPv4 Address Generic Methods

IPv4 addresses can be deleted using the generic **delete()** method. For more information, see *Deleting Objects* on page 56.

Related Methods

- *Assign IPv4 Address* on page 84
- *Assign Next Available IPv4 Address* on page 85
- *Get IPv4 Address* on page 86
- *Check Allocation for IPv4 Address* on page 87
- *Allocate Next Available Address* on page 88
- *Get Dependent Records* on page 88
- *Update IPv4 Address* on page 89

Change IPv4 Address State

public long changeStateIP4Address converts the state of an address from and between Reserved, DHCP Reserved, and Static, or DHCP Allocated to DHCP Reserved.

```
public void changeStateIP4Address( long addressId, String targetState, String
macAddress )
```

Parameters

addressId—the database ID of the address object.

targetState—one of MAKE_STATIC, MAKE_RESERVED, MAKE_DHCP_RESERVED. All of these constants are defined in the Java Class IPAssignmentActionValues or in the API.pm file for Perl.

macAddress—optional and only needed, if the target requires it. (e.g. MAKE_DHCP_RESERVED)

Output/Response

Converts an IP address from its current state to a target state; statically assigned, DHCP reserved, or logically reserved (non-DHCP). For example, this method can convert an IP address from a logical reservation to a static assignment or vise versa. Successful change of state results in null output.

Related Methods

- *Assign IPv4 Address* on page 84
- *Assign Next Available IPv4 Address* on page 85
- *Get IPv4 Address* on page 86
- *Check Allocation for IPv4 Address* on page 87
- *Allocate Next Available Address* on page 88
- *Get Dependent Records* on page 88
- *Update IPv4 Address* on page 89

IPv4 Group

Add IPv4 IP Group by Range

public long addIP4IPGroupByRange adds an IPv4 IP group by range bounds; start address and end address.

```
public long addIP4IPGroupByRange( long parentId, String name, String start, String
end, String properties )
```

Parameters

parentId—the object ID for the network in which this IP group is located.

name—the name of the IP group.

start—a start IP address of the IP group range.

end—an end IP address of the IP group range.

properties—adds object properties, including the user-defined fields.

Output/Response

Returns the object ID for the new IPv4 IP group range.

Related Methods

- *Add IPv4 IP Group by Size* on page 91.

Add IPv4 IP Group by Size

public long addIP4IPGroupBySize adds an IPv4 IP group by size.

```
public long addIP4IPGroupBySize( long parentId, String name, int size, String
positionRangeBy, String positionValue, String properties )
```

Parameters

parentId—the object ID for the network in which this IP group is located.

name—the name of the IP group.

size—the number of addresses in the IP group.

positionRangeBy—a string specifying the position of the IP group range in the parent network. The value must be one of the constants listed in *PositionRangeBy* on page 222. This is optional. If specified, *positionValue* must be provided.

positionValue—the offset value when using *positionRangeBy.START_OFFSET* or *positionRangeBy.END_OFFSET*. The start address of the IP group in the network when using *positionRangeBy.START_ADDRESS*. This is required only if *positionRangeBy* is specified.

properties—adds object properties, including the user-defined fields.

Output/Response

Returns the object ID for the new IPv4 IP group range.

Related Methods

- *Add IPv4 IP Group by Range* on page 90.

IPv4 Objects

Move IPv4 Object

This method will be deprecated in a future release in favor of the more extensive Move IP Object method. For more details, refer to *Move IP Object* on page 92.

moveIP4Object() moves an IPv4 block, an IPv4 network, or an IPv4 address to a new IPv4 address.

The block or network being moved must fit fully within the new parent object and must also not overlap its sibling objects. A network object cannot be moved directly beneath a configuration as it must be a child of a block object.

```
void moveIP4Object ( long objectId, String address )
```

Parameters

objectID—the object ID of the IPv4 block, network, or IP address to be moved.

address—the new address for the IPv4 block, network, or IP address.

Output/Response

None.

Related Methods

- *IPv4 Blocks* on page 61
- *IPv4 Networks* on page 67
- *IPv4 Addresses* on page 84
- *Move IP Object* on page 92

Move IP Object

> This method is more extensive version of the *Move IPv4 Object* on page 91, that this method replaces. Use this method to move IPv4 object.

moveIPObject() provides the option to update servers with instant dynamic host record changes on Adonis as part of the move action if the moved IP address is linked to a dynamic host record.

```
void moveIPObject ( long objectId, String address, String options )
```

Parameters

- **objectID**—the object ID of the IPv4 block, network, or IP address to be moved.
- **address**—the new address for the IPv4 block, network, or IP address.
- **options**—a string containing the following option:
  - **noServerUpdate**—boolean value. If set to true, instant dynamic host record changes will not be performed on Adonis when moving an IPv4 address object.

> This parameter will be ignored if added for block or network. It only works with IPv4 address.

Output/Response

None.

Related Methods

- *IPv4 Blocks* on page 61
- *IPv4 Networks* on page 67
- *IPv4 Addresses* on page 84
- *Move IPv4 Object* on page 91

Resize Range

resizeRange() changes the size of on IPv4 block or IPv4 network. The new size can be specified using CIDR notation or as an address range.

```
void resizeRange ( long objectId, String range, String options )
```

Parameters

objectID—the object ID of the IPv4 block or network to be resized.

range—the new size for the IPv4 block or network. Specific the size in CIDR notation or as an address range in the format *ipAddressStart-ipAddressEnd*. The address range must fit within a network boundary.

options—set as *null*. This parameter is reserved for future use.

Output/Response

None.

Related Methods

- *IPv4 Blocks* on page 61
- *IPv4 Networks* on page 67

IPv6 Objects

Add IPv6 Address

addIP6Address() adds an IPv6 address to a specified IPv6 network.

```
long addIP6Address ( long containerId, String address, String type, String name,
String properties )
```

Parameters

containerId—the object ID of the container in which the IPv6 address is being added. This can be the object ID of a Configuration, IPv6 block or IPv6 network.

> The parent IPv6 network object must exist before adding an IPv6 address, otherwise an error will occur.

address—the IPv6 address to be added. This value cannot be empty or null.

type—the type of IPv6 address. This value must be one of the following: *macAddress*, *IP6Address*, or *InterfaceID*.

> **address** and **type** must be consistent. For example, if the type is *ObjectTypes.IP6Address*, the address must be a string representing an IPv6 address.

name—descriptive name for the IPv6 address. This value can be null.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID of the new IPv6 address.

Related Methods

- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Block by Prefix* on page 95
- *Add IPv6 Network by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Clear IPv6 Address* on page 98
- *Get Entity by Prefix* on page 98
- *Get IPv6 Address* on page 99
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Add IPv6 Block by MAC Address

addIP6BlockByMACAddress() adds a IPv6 block by specifying the MAC address of the server.

```
long addIP6BlockByMACAddress ( long parentId, String macAddress, String name,
String properties )
```

Parameters

parentId—the object ID of the parent object of the new IPv6 block. The parent object must be another IPv6 block.

macAddress—the MAC address of the server in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

name—descriptive name for the IPv6 block. This value can be null.

properties—adds object properties, including user-defined fields. This value can be null.

Output/Response

Returns the object ID of the new IPv6 block.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by Prefix* on page 95
- *Add IPv6 Network by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Clear IPv6 Address* on page 98
- *Get Entity by Prefix* on page 98
- *Get IPv6 Address* on page 99
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Add IPv6 Block by Prefix

addIP6BlockByPrefix() adds an IPv6 block be specifying the prefix for the block.

```
long addIP6BlockByPrefix ( long parentId, String prefix, String name, String
properties )
```

Parameters

parentId—the object ID of the parent object of the new IPv6 block. The parent object may be a configuration or another IPv6 block.

prefix—the IPv6 prefix for the new block. This value cannot be empty or null.

name—a descriptive name for the IPv6 block. This value can be null.

properties—adds object properties, including user-defined fields. This value can be null.

Output/Response

Returns the object ID of the new IPv6 block.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Network by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Clear IPv6 Address* on page 98
- *Get Entity by Prefix* on page 98
- *Get IPv6 Address* on page 99
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Add IPv6 Network by Prefix

addIP6NetworkByPrefix() adds an IPv6 network by specifying the prefix for the network.

```
long addIP6NetworkByPrefix ( long parentId, String prefix, String name, String
properties )
```

Parameters

parentId—the object ID of the IPv6 block in which the new IPv6 network will be located.

prefix—the IPv6 prefix for the new network. This value cannot be empty or null.

name—descriptive name for the IPv6 network. This value can be null.

properties—adds object properties, including user-defined fields. This value can be null.

Output/Response

Returns the object ID of the new IPv6 network.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Block by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Clear IPv6 Address* on page 98
- *Get Entity by Prefix* on page 98
- *Get IPv6 Address* on page 99
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Get IPv6 Range by IP Address

getIPRangedByIP() returns the DHCPv6 Range containing the specified IPv6 address. Use this method to find the Configuration, IPv6 Block, IPv6 Network, or DHCPv6 Range containing a specified address. You can specify the type of object to be returned, or you can leave the type of object null to find the most direct container for the object.

```
APIEntity getIPRangedByIP( long containerId, String type, String address )
```

Parameters

containerId—the object ID of the container in which the IPv6 address is located. This can be a Configuration, IPv6 Block, IPv6 Network, or DHCPv6 Range. When you do not know the block, network, or range in which the address is located, specify the configuration.

type—the type of object containing the IPv6 address. Specify *ObjectTypes.IP6Block*, *ObjectTypes.IP6Network*, or *ObjectTypes.DHCP6Range* to find the block, network, or range containing the IPv6 address. Specify *null* to return the most direct container for the IPv6 address.

address—an IPv6 address.

Output/Response

Returns an APIEntity for the object containing the specified address. If no object is found, returns *null*. If *ObjectTypes.IP6Block*, *ObjectTypes.IP6Network*, or *ObjectTypes.DHCP6Range* is specified as the **type** parameter, returns an object of the specified type. If *null* is specified as the **type** parameter, returns the most direct container for the IPv6 address.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Block by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Clear IPv6 Address* on page 98
- *Get Entity by Prefix* on page 98
- *Get IPv6 Address* on page 99
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Assign IPv6 Address

assignIP6Address() assigns an IPv6 address to a MAC address and host.

```
boolean assignIP6Address ( long containerId, String address, String action, String
macAddress, String hostInfo, String properties )
```

Parameters

containerId—the object ID of the container in which the IPv6 address is being assigned. This can be the object ID of a Configuration, IPv6 block or IPv6 network.

The parent IPv6 network object must exist before adding an IPv6 address, otherwise an error will occur.

address—the IPv6 address to be assigned. This value cannot be empty or null.

- The address must be created with **addIP6Address()** before it can be assigned. For more information, see *Add IPv6 Address* on page 93.
- The address must be a string representing an IPv6 address.

action—determines how to assign the address. Valid values are **MAKE_STATIC** or **MAKE_DHCP_RESERVED**.

macAddress—the MAC address in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

hostInfo—the host information for the IPv6 address. This value can be empty or null. The hostInfo string uses the following format:

```
viewId, hostname, ifSameAsZone, ifReverseMapping
```

Where **viewId** is the object ID of the DNS view; **hostname** is the name of DNS zone for the address; **ifSameAsZone** is a Boolean value, with **true** indicating that the name of the resource record should be the same as the host; **ifReverseMapping** is a Boolean value, with **true** indicating that a reverse record should be created.

Shown here is an example of a **hostInfo** string:

```
2030445,www.example.com,false,true
```

properties—adds object properties, including user-defined fields. This value can be null.

Output/Response

Returns true if the IPv6 address is successfully assigned; returns false if the address is not successfully assigned.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Block by Prefix* on page 95
- *Add IPv6 Network by Prefix* on page 95
- *Clear IPv6 Address* on page 98
- *Get Entity by Prefix* on page 98
- *Get IPv6 Address* on page 99
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Clear IPv6 Address

clearIP6Address() clears a specified IPv6 address assignment.

```
boolean clearIP6Address ( long addressId )
```

addressId—the object ID of the IPv6 address to be deleted.

Output/Response

Returns **true** to indicate that the IPv6 address has been cleared, or returns **false** if the operation was unsuccessful.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Block by Prefix* on page 95
- *Add IPv6 Network by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Get Entity by Prefix* on page 98
- *Get IPv6 Address* on page 99
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Get Entity by Prefix

getEntityByPrefix() returns an APIEntity for an IPv6 block or network.

```
APIEntity getEntityByPrefix ( long containerId, String prefix, String type )
```

Parameters

containerId—the object ID of higher-level parent object (IPv6 block or configuration) in which the IPv6 block or network is located.

prefix—the prefix value for the IPv6 block or network. This value cannot be null or empty.

type—the type of object to be returned. This value must be either IP6Block or IP6Network.

Output/Response

Returns an APIEntity for the specified IPv6 block or network. The APIEntity is null if the block or network does not exist.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Block by Prefix* on page 95
- *Add IPv6 Network by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Clear IPv6 Address* on page 98
- *Get IPv6 Address* on page 99
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Get IPv6 Address

getIP6Address() returns an APIEntity for an IPv6 address.

```
APIEntity getIP6Address ( long containerId, String address )
```

Parameters

containerId—the object ID of the container in which the IPv6 address is located. The container can be a configuration, an IPv6 block, or an IPv6 network.

address—the IPv6 address.

Output/Response

Returns an APIEntity for the specified IPv6 address. The APIEntity is null of the IPv6 address does not exist.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Block by Prefix* on page 95
- *Add IPv6 Network by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Clear IPv6 Address* on page 98
- *Get Entity by Prefix* on page 98
- *Reassign IPv6 Address* on page 100
- *Update IPv6 Objects* on page 100

Reassign IPv6 Address

reassignIP6Address() reassigns an existing IPv6 address to a new IPv6 address. The destination address can be specified as an IPv6 address or as a MAC address from which an IPv6 address can be calculated.

```
long reassignIP6Address ( long oldAddressId, String destination, String properties
)
```

Parameters

oldAddressId—the object ID of the current IPv6 address.

destination—the destination of the reassigned address. This can be specified as an IPv6 address string (NetworkID and InterfaceID), or as a MAC address from which the new IPv6 address can be calculated. The MAC address can be specified in the format *nnnnnnnnnnnn* or *nn-nn-nn-nn-nn-nn*, where *nn* is a hexadecimal value.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID of the reassigned IPv6 address.

Related Methods

- *Add IPv6 Address* on page 93
- *Add IPv6 Block by MAC Address* on page 94
- *Add IPv6 Block by Prefix* on page 95
- *Add IPv6 Network by Prefix* on page 95
- *Assign IPv6 Address* on page 97
- *Clear IPv6 Address* on page 98
- *Get Entity by Prefix* on page 98
- *Get IPv6 Address* on page 99
- *Update IPv6 Objects* on page 100

Update IPv6 Objects

An IPv6 block, network or address’s **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Provision Devices

Add Device Instance

addDeviceInstance() is used to provision new devices for the network and combines a number of existing API methods into one. This method assigns the next available, or manually defined, IP address and optionally adds a DNS host record and MAC address that are linked to the IP address. When configured with a DNS host record, **addDeviceInstance()** will update the DNS server to immediately deploy the host record.

> Proteus adds the DNS host record directly to the Adonis server so that the individual host record is made live instantly. This is done through the Proteus to Adonis communication service (Command Server) and does NOT require a standard Proteus deployment.

```
String addDeviceInstance ( String configName, String deviceName, String
ipAddressMode, String ipEntity, String viewName, String zoneName, String
recordName, String macAddressMode, String macEntity, String options )
```

Parameters

- **configName**—name of parent configuration. If the value is null or cannot be found, an exception will be thrown.
- **deviceName**—IP/device name of the new instance. Reserved for future use.
- **ipAddressMode**—accepted values are REQUEST_STATIC, REQUEST_DHCP_RESERVED and PASS_VALUE. REQUEST_STATIC or REQUEST_DHCP_RESERVED is used to get the next available IP address. PASS_VALUE is used to pass an existing IP address. Metadata values will be updated to the newly assigned IP address only.

> REQUEST_DHCP_RESERVED is reserved for future use.

- **ipEntity**—if ipAddressMode is REQUEST_STATIC or REQUEST_DHCP_RESERVED, this needs to be the network where the IP address will be provisioned from in the format of an IP address range in CIDR format or range. If ipAddressMode is PASS_VALUE, this needs to be an IP address.
- **viewName**—name of parent view.
- **domainName**—parent zone of the record. This must be specified and existing if the viewName parameter is not null and existing.
- **recordName**—name of the host record to add. This cannot be null if both viewName and zoneName are specified and existing.

> The **viewName**, **domainName** and **recordName** parameters need to be used together: the values must all be *null*, or they must all be populated with specific values, that is, a non-empty string. If the values are *null*, DNS objects will not be created but an IP address will be assigned from a network and linked to a MAC address.

- **macAddressMode**—accepted values are REQUEST_VALUE or PASS_VALUE. If null, the MACEntity parameter will be ignored. Use PASS_VALUE to manually provide the MAC address linked to IP address. REQUEST_VALUE is reserved for future use.
- **macEntity**—if MACAddressMode is PASS_VALUE, this must be a MAC address. If MACAddressMode is REQUEST_VALUE, this is a MAC mask.
- **options**—the options string contains four properties, **skip**, **offset**, **excludeDHCPRange** and **allowDuplicateHosts**. The values for skip and offset must be IPv4 addresses and must appear in dotted octet notation.
  - **skip**—this is optional. It is applied to REQUEST_STATIC and REQUEST_DHCP_RESERVED for ipAddressMode. It is used to specify the IP address ranges or IP addresses to skip, separated by comma. A hyphen(-), not a dash is used to separate the start and end addresses.
  - **offset**—this is optional. It is applied to REQUEST_VALUE for ipAddressMode. This is to specify from which address to start to assign IPv4 Address.

- **excludeDHCPRange**—this specifies whether IP addresses in DHCP ranges should be excluded from assignment or not. It is applied to REQUEST_STATIC only for ipAddressMode. The value is either true or false, default value is false. The value will always be set to true if the ipAddressMode is REQUEST_DHCP_RESERVED.
- **allowDuplicateHosts**—this specifies whether the IP address can be added to an existing host record or not. The value is either true or false, default value is false.

Output/Response

Returns the property string containing IP address, netmask and gateway.

Related Methods

- *Delete Device Instance* on page 102

De-provision Devices

Delete Device Instance

deleteDeviceInstance() deletes either the IP address or MAC address (and all related DNS entries including host records, PTR records, or DHCP reserved addresses) on both the Proteus and Adonis server based on the IPv4 address or a MAC address supplied.

> Proteus deletes or updates the DNS host record directly to the Adonis server so that the individual host record is made live instantly. This is done through the Proteus to Adonis communication service (Command Server) and does NOT require a standard Proteus deployment.

```
deleteDeviceInstance ( String configName, String identifier, String options )
```

Parameters

- **configName**—name of parent configuration. If the value is null or cannot be found, an exception will be thrown.
- **identifier**—IP address or MAC address. If the value is null or cannot be found, an exception will be thrown. Relevant IP addresses, host records and MAC addresses liked to multiple entities will be updated. Relevant IP addresses, host records and MAC addresses liked to a single entity will be deleted.
- **options**—currently null. This parameter is reserved for future use.

Output/Response

None.

Related Methods

- *Add Device Instance* on page 100

DHCP

DHCP is an essential part of IPAM. DHCP manages the dynamic allocation of IP addresses on an IP network using the concept of address leases. In Proteus, DHCP is integrated into the IP core and defined using range objects. At most levels of the IP core, deployment options control the behavior of the DHCP service for an object and its descendant objects within the Proteus-managed network. A deployment role can also be associated with a server to provide DHCP services for a specific subnet.

IPv4 DHCP Ranges

DHCP ranges indicate the portion of a network that is dedicated to DHCP. Ranges can have deployment options assigned to them to control the exact settings that clients receive. Proteus then manages the deployment of the DHCP ranges to the managed server and activates the configuration.

Add IPv4 DHCP Range

addDHCP4Range() adds IPv4 DHCP ranges.

```
long addDHCP4Range( long networkId, String start, String end, String properties )
```

Parameters

networkId—the object ID for the network in which this DHCP range is located.

start—an IP address defining the lowest address or start of the range.

end—an IP address defining the highest address or end of the range.

properties—adds object properties, including the object name and user-defined fields.

Output/Response

Returns the object ID for the new DHCPv4 range.

Related Methods

- *Get IPv4 DHCP Range* on page 105
- *Get IPv4 DHCP Ranges* on page 105
- *Update IPv4 DHCP Range* on page 106
- *IPv4 DHCP Range Generic Methods* on page 106

Add IPv4 DHCP Range By Size

addDHCP4RangeBySize() adds IPv4 DHCP ranges by offset and percentage.

```
long addDHCP4RangeBySize( long networkId, String offset, String size, String
properties )
```

Parameters

networkId—the object ID for the network in which this DHCP range is located.

offset—an integer value specifying the point where the range should begin. The positive values indicate that the starting IP address of the range will be counted from the Network ID (first IP address) and forward in the range. The negative values indicate that the starting IP address of the range will be counted from the Network Broadcast Address (last IP address) and backward in the range.

size—the size of the range. Currently the range size can only be specified in a relative size in proportion to the parent network size. To define the relative range size, **defineRangeBy** must be set with the *OFFSET_AND_PERCENTAGE* value in the properties field.

properties—optional object properties that can contain the object name, the value of **defineRangeBy**, and user-defined fields. The possible values for **defineRangeBy** are *OFFSET_AND_SIZE* and *OFFSET_AND_PERCENTAGE*.

> *OFFSET_AND_SIZE* is reserved for future use.

Output/Response

Returns the object ID for the new DHCPv4 range.

Related Methods

- *Get IPv4 DHCP Range* on page 105
- *Get IPv4 DHCP Ranges* on page 105
- *Update IPv4 DHCP Range* on page 106
- *IPv4 DHCP Range Generic Methods* on page 106

Get IPv4 Range by IP Address

getIPRangedByIP() returns the DHCP Range containing the specified IPv4 address. Use this method to find the Configuration, IPv4 Block, IPv4 Network, or DHCP Range containing a specified address. You can specify the type of object to be returned, or you can leave the type of object null to find the most direct container for the object.

```
APIEntity getIPRangedByIP( long containerId, String type, String address )
```

Parameters

containerId—the object ID of the container in which the IPv4 address is located. This can be a Configuration, IPv4 Block, IPv4 Network, or DHCP Range. When you do not know the block, network, or range in which the address is located, specify the configuration.

type—the type of object containing the IPv4 address. Specify `ObjectTypes.IP4Block`, `ObjectTypes.IP4Network`, or `ObjectTypes.DHCP4Range` to find the block, network, or range containing the IPv4 address. Specify `null` to return the most direct container for the IPv4 address.

address—an IPv4 address.

Output/Response

Returns an APIEntity for the object containing the specified address. If no object is found, returns `null`. If `ObjectTypes.IP4Block`, `ObjectTypes.IP4Network`, or `ObjectTypes.DHCP4Range` is specified as the **type** parameter, returns an object of the specified type. If `null` is specified as the **type** parameter, returns the most direct container for the IPv4 address.

Related Methods

- *Get IPv4 Range by IP Address* on page 104
- *Get IPv4 DHCP Range* on page 105
- *Get IPv4 DHCP Ranges* on page 105
- *Get Entity by Name* on page 48
- *Get Entity by ID* on page 48
- *Get Entities* on page 49

Get IPv4 DHCP Range

getEntityByRange() returns an IPv4 DHCP range by calling it using its range.

```
APIEntity getEntityByRange( long parentId, String address1, String address2, String
type )
```

Parameters

parentId—the object ID of the parent object of the DHCP range.

address1—an IP address defining the lowest address or start of the range.

address2—an IP address defining the highest address or end of the range.

type—the type of object returned: DHCP4Range. This must be one of the constants listed in *Object Types* on page 220.

Output/Response

Returns the specified DHCP4Range object from the database.

Related Methods

- *Add IPv4 DHCP Range* on page 103
- *Get IPv4 DHCP Ranges* on page 105
- *Update IPv4 DHCP Range* on page 106
- *IPv4 DHCP Range Generic Methods* on page 106

Get IPv4 DHCP Ranges

getEntites() returns multiple IPv4 DHCP ranges for the specified parent ID.

```
APIEntity[] getEntities( long parentId, String type, int start, int count )
```

Parameters

parentId—the object ID of the parent object of the DHCP range.

type—the type of object returned: DHCP4Range. This must be one of the constants listed in *Object Types* on page 220.

start—indicates where in the list of child ranges to start returning range objects. The list begins at an index of 0.

count—the maximum number of DHCP range objects to return.

Output/Response

Returns an array of DHCPv4 range objects from the database.

Related Methods

- *Add IPv4 DHCP Range* on page 103
- *Get IPv4 DHCP Range* on page 105
- *Update IPv4 DHCP Range* on page 106
- *IPv4 DHCP Range Generic Methods* on page 106

Get Max Allowed Range

getMaxAllowedRange() finds the maximum possible address range to which the existing IPv4 DHCP range can be extended. This method only supports the IPv4 DHCP range.

```
public String[] getMaxAllowedRange ( long rangeId )
```

Parameters

rangeId—the object ID of the IPv4 DHCP range.

Output/Response

Returns the possible start address and end address for the specified IPv4 DHCP range object in the form of array of length 2.

Related Methods

- *Add IPv4 DHCP Range* on page 103
- *Get IPv4 DHCP Range* on page 105
- *Update IPv4 DHCP Range* on page 106
- *IPv4 DHCP Range Generic Methods* on page 106

Update IPv4 DHCP Range

A DHCP range’s **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add IPv4 DHCP Range* on page 103
- *Get IPv4 DHCP Range* on page 105
- *Get IPv4 DHCP Ranges* on page 105
- *IPv4 DHCP Range Generic Methods* on page 106

IPv4 DHCP Range Generic Methods

DHCP ranges can be deleted using the generic **delete()** method. For more information, see *Deleting Objects* on page 56.

Related Methods

- *Add IPv4 DHCP Range* on page 103
- *Get IPv4 DHCP Range* on page 105
- *Get IPv4 DHCP Ranges* on page 105
- *Update IPv4 DHCP Range* on page 106

IPv6 DHCP Ranges

DHCPv6 ranges indicate the portion of a network that is dedicated to DHCPv6. Ranges can have deployment options assigned to them to control the exact settings that clients receive. Proteus then manages the deployment of the DHCPv6 ranges to the managed server and activates the configuration.

Add IPv6 DHCP Range

addDHCP6Range() adds IPv6 DHCP ranges.

```
long addDHCP6Range( long networkId, String start, String end, String properties )
```

Parameters

networkId—the object ID for the network in which this DHCPv6 range is located.

start—an IP address defining the lowest address or start of the range.

end—an IP address defining the highest address or end of the range.

properties—adds object properties, including the object name and user-defined fields.

Output/Response

Returns the object ID for the new DHCPv6 range.

Related Methods

- *Add IPv6 DHCP Range* on page 107
- *Get IPv6 DHCP Range* on page 108
- *Update IPv6 DHCP Range* on page 109
- *IPv6 DHCP Range Generic Methods* on page 109

Get IPv6 Range by IP Address

getIPRangedByIP() returns the DHCPv6 Range containing the specified IPv6 address. Use this method to find the Configuration, IPv6 Block, IPv6 Network, or DHCPv6 Range containing a specified address. You can specify the type of object to be returned, or you can leave the type of object null to find the most direct container for the object.

```
APIEntity getIPRangedByIP( long containerId, String type, String address )
```

Parameters

containerId—the object ID of the container in which the IPv6 address is located. This can be a Configuration, IPv6 Block, IPv6 Network, or DHCPv6 Range. When you do not know the block, network, or range in which the address is located, specify the configuration.

type—the type of object containing the IPv6 address. Specify *ObjectTypes.IP6Block*, *ObjectTypes.IP6Network*, or *ObjectTypes.DHCP6Range* to find the block, network, or range containing the IPv6 address. Specify *null* to return the most direct container for the IPv6 address.

address—an IPv6 address.

Output/Response

Returns an APIEntity for the object containing the specified address. If no object is found, returns *null*. If *ObjectTypes.IP6Block*, *ObjectTypes.IP6Network*, or *ObjectTypes.DHCP6Range* is specified as the **type** parameter, returns an object of the specified type. If *null* is specified as the **type** parameter, returns the most direct container for the IPv6 address.

Related Methods

- *Add IPv6 DHCP Range* on page 107
- *Get IPv6 DHCP Range* on page 108
- *Update IPv6 DHCP Range* on page 109
- *IPv6 DHCP Range Generic Methods* on page 109

Get IPv6 DHCP Range

getEntityByRange() returns an IPv6 DHCP range by calling it using its range.

```
APIEntity getEntityByRange( long parentId, String address1, String address2, String
type )
```

Parameters

parentId—the object ID of the parent object of the DHCPv6 range.

address1—an IP address defining the lowest address or start of the range.

address2—an IP address defining the highest address or end of the range.

type—the type of object returned: DHCPv6 Range. This must be one of the constants listed in *Object Types* on page 220.

Output/Response

Returns the specified DHCP6Range object from the database.

Related Methods

- *Add IPv6 DHCP Range* on page 107
- *Get Multiple IPv6 DHCP Ranges* on page 108
- *Update IPv6 DHCP Range* on page 109
- *IPv6 DHCP Range Generic Methods* on page 109

Get Multiple IPv6 DHCP Ranges

getEntites() returns multiple IPv6 DHCP ranges for the specified parent ID.

```
APIEntity[] getEntities( long parentId, String type, int start, int count )
```

Parameters

parentId—the object ID of the parent object of the DHCPv6 range.

type—the type of object returned: DHCPv6 Range. This must be one of the constants listed in *Object Types* on page 220.

start—indicates where in the list of child ranges to start returning range objects. The list begins at an index of 0.

count—the maximum number of DHCPv6 range objects to return.

Output/Response

Returns an array of DHCPv6 range objects from the database.

Related Methods

- *Add IPv6 DHCP Range* on page 107
- *Add IPv6 DHCP Range* on page 107
- *Update IPv6 DHCP Range* on page 109
- *IPv6 DHCP Range Generic Methods* on page 109

Update IPv6 DHCP Range

A DHCPv6 range's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add IPv6 DHCP Range* on page 107
- *Get IPv6 DHCP Range* on page 108
- *Get Multiple IPv6 DHCP Ranges* on page 108
- *IPv6 DHCP Range Generic Methods* on page 109

IPv6 DHCP Range Generic Methods

DHCPv6 ranges can be deleted using the generic **delete()** method. For more information, see *Deleting Objects* on page 56.

Related Methods

- *Add IPv6 DHCP Range* on page 107
- *Get Multiple IPv6 DHCP Ranges* on page 108
- *Get Multiple IPv6 DHCP Ranges* on page 108
- *Update IPv6 DHCP Range* on page 109

DHCP Client Options

These are the DHCP options that can be added to a DHCP configuration to specify deployment instructions relating to extra settings for client configuration. For more information about these options, refer to RFCs 2132, 2242, 2610, 2241, and 2485. Readers are also encouraged to examine RFCs 1497 and 1122 for background information.

Options that accept Boolean values are activated by a value of 1 unless otherwise specified. When specifying a list of IPv4 addresses, the first address takes precedence.

Add DHCP Client Option

addDHCPClientDeploymentOption() adds DHCP client options and returns the object ID for the new option object.

```
long addDHCPClientDeploymentOption( long entityId, String name, String value,
String properties )
```

Parameters

entityId—the object ID for the entity to which the deployment option is being added.

name—the name of the DHCPv4 client option being added. This name must be one of the constants listed in *DHCP Client Options* on page 206.

value—the value being assigned to the option.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new DHCPv4 client object.

Related Methods

- *Get DHCP Client Option* on page 110
- *Update DHCP Client Option* on page 111
- *Delete DHCP Client Option* on page 111

Get DHCP Client Option

getDHCPClientDeploymentOption() returns DHCP client options.

```
APIDeploymentOption getDHCPClientDeploymentOption( long entityId, String name, long
serverId )
```

Parameters

entityId—the object ID for the entity to which the deployment option has been applied.

name—the name of the DHCPv4 client option being returned. This name must be one of the constants listed in *DHCP Client Options* on page 206.

serverId—the specific server to which this option is deployed. To return an option that has not been assigned to a server, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Output/Response

Returns the specified DHCPv4 client option object from the database.

Related Methods

- *Add DHCP Client Option* on page 110
- *Update DHCP Client Option* on page 111
- *Delete DHCP Client Option* on page 111

Update DHCP Client Option

updateDHCPClientDeploymentOption() updates DHCP client options.

```
void updateDHCPClientDeploymentOption( APIDeploymentOption option )
```

Parameters

option—this is the DHCP client option object to be updated.

Output/Response

None.

Related Methods

- *Add DHCP Client Option* on page 110
- *Get DHCP Client Option* on page 110
- *Delete DHCP Client Option* on page 111

Delete DHCP Client Option

deleteDHCPClientDeploymentOption() deletes DHCP client options.

```
void deleteDHCPClientDeploymentOption( long entityId, String name, long serverId )
```

Parameters

entityId—the object ID for the entity from which the deployment option will be deleted.

name—the name of the DHCPv4 client option to be deleted. This name must be one of the constants listed in *DHCP Client Options* on page 206.

serverId—the specific server to which this option is deployed. To delete an option that has not been assigned to a server, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Output/Response

None.

Related Methods

- *Add DHCP Client Option* on page 110
- *Get DHCP Client Option* on page 110
- *Update DHCP Client Option* on page 111

DHCP6 Client Options

These are the DHCPv6 options that can be added to a DHCP configuration to specify deployment instructions relating to extra settings for client configuration.

Add DHCP6 Client Option

addDHCP6ClientDeploymentOption() adds DHCPv6 client options and returns the database object ID for the new option object.

```
long addDHCP6ClientDeploymentOption( long entityId, String name, String value,
String properties )
```

Parameters

entityId—the object ID for the entity to which the deployment option is being added.

name—the name of the DHCPv6 client option being added. This name must be one of the constants listed in the *DHCP6 Client Options* on page 209.

value—the value being assigned to the option

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID of the new DHCPv6 client object.

Related Methods

- *Get DHCP6 Client Option* on page 112
- *Update DHCP6 Client Option* on page 113
- *Delete DHCP6 Client Option* on page 113

Get DHCP6 Client Option

getDHCP6ClientDeploymentOption() returns DHCPv6 client options.

```
APIDeploymentOption getDHCP6ClientDeploymentOption( long entityId, String name,
long serverId )
```

Parameters

entityId—the object ID for the entity to which the deployment option is being added.

name—the name of the DHCPv6 client option being added. This name must be one of the constants listed in the *DHCP6 Client Options* on page 209.

serverID—the specific server to which this option is deployed. To return an option that has not been assigned to a server role, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Output/Response

Returns the specified DHCP6 client option object from the database.

Related Methods

- *Add DHCP6 Client Option* on page 112
- *Update DHCP6 Client Option* on page 113
- *Delete DHCP6 Client Option* on page 113

Update DHCP6 Client Option

updateDHCP6ClientDeploymentOption() updates DHCPv6 client options.

```
void updateDHCP6ClientDeploymentOption(APIDeploymentOption option)
```

Parameters

option—this is the DHCPv6 client option object that is updated.

Output/Response

None.

Related Methods

- *Add DHCP6 Client Option* on page 112
- *Get DHCP6 Client Option* on page 112
- *Delete DHCP6 Client Option* on page 113

Delete DHCP6 Client Option

deleteDHCP6ClientDeploymentOption() deletes DHCPv6 client options.

```
void deleteDHCP6ClientDeploymentOption(long entityId, String name, long serverId)
```

Parameters

entityID—the database object ID for the entity from which this deployment option will be deleted.

name—the name of the DHCPv6 client option being deleted. This name must be one of the constants listed in the *DHCP6 Client Options* on page 209.

serverID—the specific server to which this option is deployed. To delete an option that has not been assigned to a server role, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Output/Response

None.

Related Methods

- *Add DHCP6 Client Option* on page 112
- *Get DHCP6 Client Option* on page 112
- *Update DHCP6 Client Option* on page 113

DHCP Custom Options

The **addCustomOptionDefinition()** method is available to add DHCP custom option definitions.

Add Custom Deployment Option

addCustomOptionDefinition() adds a custom deployment option.

```
long addCustomOptionDefinition( long configurationId, String name, long optionId,
String optionType, boolean allowMultiple, String properties )
```

Parameters

configurationId—the object ID of the parent configuration.

name—the name of the custom deployment option. This value cannot be null or empty.

optionId—the option code for the custom deployment option. This value must be within the range of 151 to 174, 178 to 207, 212 to 219, 222 to 223, or 224 to 254.

optionType—the type of custom deployment option. This value must be one of the items listed in *DHCP Custom Option Types* on page 210.

allowMultiple—determines whether or not the custom option requires multiple values. If set to true, the option applies only to the IP4 option type. The default value is false. The value cannot be null or empty.

In Perl script, only an empty string and 0 (zero) are considered as **false**; other values are considered as **true**. Therefore, a string containing the word “false” is considered to be true because the string is not empty.

In Perl, set the **allowMultiple** data type to *string* and set the value to either **true** or **false**:

```
SOAP::Data->type( 'string' )->
        name( 'allowMultiple' )->
        value( "false")->
        attr({xmlns => ''})
```

Or, set the **allowMultiple** data type to *boolean*. Set the value to either **0** or an empty string to represent **false**. Set the value to any other text to represent **true**.

```
SOAP::Data->type( 'boolean' )
        ->name( 'allowMultiple' )
        ->value( 0 )
        ->attr({xmlns => ''})
```

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID of the new custom option definition.

DHCP Service Options

These are the DHCP service options that can be added to a DHCP configuration to specify deployment instructions relating to extra settings for service configuration.

Add DHCP Service Option

addDHCPServiceDeploymentOption() adds DHCP service options.

```
long addDHCPServiceDeploymentOption( long entityId, String name, String value,
String properties )
```

Parameters

entityId—the object ID for the entity to which the deployment option is being added.

name—the name of the DHCPv4 service option being added. This name must be one of the constants listed in *DHCP Service Options* on page 204.

value—the value being assigned to the option.

When adding the DDNS hostname option, you need to specify the value in the following format: [**Type**],[**Position**],[**Data**] for IP and MAC type, and [**Type**],[**Data**] for FIXED type. Where:

- **Type**—type of DDNS hostname. The possible values are DHCPServiceOptionConstants.DDNS_HOSTNAME_TYPE_IP, DHCPServiceOptionConstants.DDNS_HOSTNAME_TYPE_MAC, or DHCPServiceOptionConstants.DDNS_HOSTNAME_TYPE_FIXED.
- **Position**—specify where you wish to add the data value to the IP or MAC address. The possible values are DHCPServiceOptionConstants.DDNS_HOSTNAME_POSITION_PREPEND, or DHCPServiceOptionConstants.DDNS_HOSTNAME_POSITION_APPEND.

  This is only required for IP or MAC type with Data.
- **Data**—For IP and MAC address, this value is used to be prepended or appended to the IP address or MAC address. For FIXED type, this value must be specified and will be used for the DDNS hostname. This is optional for IP and MAC type.

If you are adding the following DHCPv4 service options, leave the **value** field as empty:

- ALLOW_DHCP_CLASS_MEMBERS
- DENY_DHCP_CLASS_MEMBERS
- ALLOW_MAC_POOL
- DENY_MAC_POOL

properties—adds object properties, including user-defined fields.

- When adding the ALLOW_DHCP_CLASS_MEMBERS or DENY_DHCP_CLASS_MEMBERS option, use the following new parameter: dhcpMatchClass=object ID.
- When adding the ALLOW_MAC_POOL or DENY_MAC_POOL option, use the following new parameter: macPool=object ID.

Output/Response

Returns the object ID for the new DHCPv4 service option.

Related Methods

- *Get DHCP Service Option* on page 116
- *Update DHCP Service Option* on page 116
- *Delete DHCP Service Option* on page 116

Get DHCP Service Option

getDHCPServiceDeploymentOption() returns DHCP service options for a specified object.

```
APIDeploymentOption getDHCPServiceDeploymentOption( long entityId, String name,
long serverId )
```

Parameters

entityId—the object ID for the entity to which the deployment option is assigned.

name—the name of the DHCPv4 service option being retrieved. This name must be one of the constants listed in *DHCP Service Options* on page 204.

serverId—specifies the server to which the option is deployed for the specified entity. To retrieve an option that has not been assigned to a server role, specify 0 as a value. Omitting this parameter from the method call will result in an error.

Output/Response

Returns the requested DHCPv4 service option object from the database.

Related Methods

- *Add DHCP Service Option* on page 115
- *Update DHCP Service Option* on page 116
- *Delete DHCP Service Option* on page 116

Update DHCP Service Option

updateDHCPServiceDeploymentOption() updates DHCP service options.

```
void updateDHCPServiceDeploymentOption( APIDeploymentOption option )
```

Parameters

option—the DHCP service option object to be updated.

Output/Response

None.

Related Methods

- *Add DHCP Service Option* on page 115
- *Get DHCP Service Option* on page 116
- *Delete DHCP Service Option* on page 116

Delete DHCP Service Option

deleteDHCPServiceDeploymentOption() deletes DHCP service options.

```
void deleteDHCPServiceDeploymentOption( long entityId, String name, long serverId )
```

Parameters

entityId—the object ID for the entity from which this deployment option is being deleted.

name—the name of the DHCPv4 service option being deleted. This name must be one of the constants listed in *DHCP Service Options* on page 204.

serverId—specifies the server to which the option is deployed for the specified entity. To retrieve an option that has not been assigned to a server role, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Related Methods

- *Add DHCP Service Option* on page 115
- *Get DHCP Service Option* on page 116
- *Update DHCP Service Option* on page 116

DHCP6 Service Options

These are the DHCPv6 service options that can be added to a DHCP configuration to specify deployment instructions relating to extra settings for service configuration.

Add DHCP6 Service Option

addDHCP6ServiceDeploymentOption() adds DHCPv6 service options.

```
long addDHCP6ServiceDeploymentOption( long entityId, String name, String value,
String properties )
```

Parameters

entityId—the object ID for the entity to which the deployment option is being added.

name—the name of the DHCPv6 service option being added. This name must be one of the constants listed in *DHCP6 Service Options* on page 206.

value—the value being assigned to the option.

When adding the DDNS hostname option, you need to specify the value in the following format: [**Type**],[**Data**] for IP and DUID. Where:

- **Type**—type of DDNS hostname. The possible values are DHCPServiceOptionConstants.DDNS_HOSTNAME_TYPE_IP, DHCPServiceOptionConstants.DDNS_HOSTNAME_TYPE_DUID.
- **Data**—For IP and DUID, this value is used to form the DDNS hostname. This is optional.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new DHCPv6 service option.

- *Get DHCP6 Service Option* on page 118
- *Update DHCP6 Service Option* on page 118
- *Delete DHCP6 Service Option* on page 118

Get DHCP6 Service Option

getDHCP6ServiceDeploymentOption() returns DHCPv6 service options for a specified object.

```
APIDeploymentOption getDHCP6ServiceDeploymentOption( long entityId, String name,
long serverId )
```

Parameters

entityId—the database object ID for the entity to which the deployment option is assigned.

name—the name of the DHCPv6 service option being retrieved. This name must be one of the constants listed in *DHCP6 Service Options* on page 206.

serverId—specifies the server to which the option is deployed for the specified entity. To retrieve an option that has not been assigned to a server role, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Output/Response

Returns the requested DHCPv6 service option object from the database.

Related Methods

- *Add DHCP6 Service Option* on page 117
- *Update DHCP6 Service Option* on page 118
- *Delete DHCP6 Service Option* on page 118

Update DHCP6 Service Option

updateDHCP6ServiceDeploymentOption() updates DHCPv6 service options.

```
void updateDHCP6ServiceDeploymentOption( APIDeploymentOption option )
```

Parameters

option—the DHCPv6 service option object to be updated.

Output/Response

None.

Related Methods

- *Add DHCP6 Service Option* on page 117
- *Get DHCP6 Service Option* on page 118
- *Delete DHCP6 Service Option* on page 118

Delete DHCP6 Service Option

deleteDHCP6ServiceDeploymentOption() deletes DHCP service options.

```
void deleteDHCP6ServiceDeploymentOption( long entityId, String name, long serverId
)
```

Parameters

entityId—the object ID for the entity from which this deployment option is being deleted.

name—the name of the DHCPv4 service option being deleted. This name must be one of the constants listed in *DHCP6 Service Options* on page 206.

serverId—specifies the server to which the option is deployed for the specified entity. To return an option that has not been assigned to a server role, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Related Methods

- *Add DHCP6 Service Option* on page 117
- *Get DHCP6 Service Option* on page 118
- *Update DHCP6 Service Option* on page 118

DHCP Vendor Options

Add DHCP Vendor Deployment Option

addDHCPVendorDeploymentOption() adds a DHCP vendor deployment to specified objects.

```
long addDHCPVendorDeploymentOption ( long parentId, long optionId, String value,
String properties )
```

Parameters

parentId—the object ID of the parent object for the DHCP vendor deployment option. The parent object must not be a DNS object. Valid parent types are Configuration, IP4Block, IP4Network, IP4Addr, IP4DHCPRange, Server, MACAddr, and MACPool.

optionId—the object ID of the vendor option definition. All DHCP vendor client deployment options have a fixed option code of 60 and a unique option sub-code. The unique sub-code is set with the optionId value in the **addVendorOptionDefinition()** method.

value—the value for the option. This value cannot be null or empty. The value should be appropriate for its option type. For example, if the option type is **IP4** and **allowMultiple** is set as **true** in the vendor option definition, then the value of the DHCP vendor client deployment option should be multiple IPv4 addresses in a comma-separated list.

properties—adds object properties, including user-defined fields. This value can be null. If the DHCP vendor client deployment option is intended for use with a specific server, the object ID of the server must be specified in the **properties** string.

Output/Response

Returns the object ID of the new DHCP vendor deployment option.

Related Methods

- *Add Vendor Option Definition* on page 120
- *Add Vendor Profile* on page 121
- *Delete DHCP Vendor Deployment Option* on page 121
- *Get DHCP Vendor Deployment Option* on page 122
- *Update DHCP Vendor Deployment Option* on page 123

Add Vendor Option Definition

addVendorOptionDefinition() adds a vendor option definition to a vendor profile.

```
long addVendorOptionDefinition ( long vendorProfileId, long optionId, String name,
String optionType, String description, boolean allowMultiple, String properties )
```

Parameters

vendorProfileId—the object ID of the parent vendor profile.

optionID—the deployment option ID. This value must be within the range of 1 to 254.

name—the name of the vendor option. This value cannot be null or empty.

optionType—the option type. This value must be one of the types listed in *Vendor Profile Option Types* on page 213.

description—a description of the vendor option. This value cannot be null or empty.

allowMultiple—determines whether or not the custom option requires multiple values. The default value is **false**. This value cannot be null or empty.

In Perl script, only an empty string and 0 (zero) are considered as **false**; other values are considered as **true**. Therefore, a string containing the word "false" is considered to be true because the string is not empty.

In Perl, set the **allowMultiple** data type to *string* and set the value to either **true** or **false**:

```
SOAP::Data->type( 'string' )->
        name( 'allowMultiple' )->
        value( "false")->
        attr({xmlns => ''})
```

Or, set the **allowMultiple** data type to *boolean*. Set the value to either **0** or an empty string to represent **false**. Set the value to any other text to represent **true**.

```
SOAP::Data->type( 'boolean' )
        ->name( 'allowMultiple' )
        ->value( 0 )
        ->attr({xmlns => ''})
```

properties—adds object properties, including user-defined fields. This value can be null.

Output/Response

Returns the object ID of the new vendor option definition.

Related Methods

- *Add DHCP Vendor Deployment Option* on page 119
- *Add Vendor Profile* on page 121
- *Delete DHCP Vendor Deployment Option* on page 121
- *Get DHCP Vendor Deployment Option* on page 122
- *Update DHCP Vendor Deployment Option* on page 123

Add Vendor Profile

addVendorProfile() adds a vendor profile and returns the object ID for the new vendor profile.

```
long addVendorProfile ( String identifier, String name, String description, String
properties )
```

Parameters

identifier—the Vendor Class Identifier.

name—a descriptive name for the vendor profile. This name is not matched against DHCP functionality.

description—a description of the vendor profile.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new vendor profile.

Related Methods

- *Add DHCP Vendor Deployment Option* on page 119
- *Add Vendor Option Definition* on page 120
- *Delete DHCP Vendor Deployment Option* on page 121
- *Get DHCP Vendor Deployment Option* on page 122
- *Update DHCP Vendor Deployment Option* on page 123

Delete DHCP Vendor Deployment Option

deleteDHCPVendorDeploymentOption() deletes a specified DHCP vendor deployment option.

```
void deleteDHCPVendorDeploymentOption ( long entityId, long optionId, long serverId
)
```

Parameters

entityId—the object ID of the object to which the DHCP vendor deployment option is assigned.

optionId—the object ID of the DHCP vendor deployment option to be deleted.

serverId—the object ID of the server where the DHCP vendor deployment option is used. If the option is generic, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Output/Response

None.

Related Methods

- *Add DHCP Vendor Deployment Option* on page 119
- *Add Vendor Option Definition* on page 120
- *Add Vendor Profile* on page 121
- *Get DHCP Vendor Deployment Option* on page 122
- *Update DHCP Vendor Deployment Option* on page 123

Get DHCP Vendor Deployment Option

getDHCPVendorDeploymentOption() retrieves a DHCP vendor deployment option.

```
APIDeploymentOption getDHCPVendorDeploymentOption ( long entityId, long optionId,
long serverId )
```

Parameters

entityId—the object ID of the entity to which the DHCP vendor deployment option is assigned. This must be the ID of a configuration, IPv4 block, IPv4 network, IPv4 address, IPV4 DHCP rage, server, MAC address, or MAC Pool.

optionId—the object ID of the DHCP vendor option definition.

serverId—the specific server to which this option is deployed for the specified entity. To return an option that has not been assigned to a server, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Output/Response

Returns an APIDeploymentOption for the DHCP vendor client deployment option. The APIDeploymentOption is null if the specified option does not exist.

The **property** string of the returned APIDeploymentOption contains at least contain the following substring to represent vendor option definitions:

```
optionId=optionID|optionType=Integer(-128 to 127)|optionDescription=description
multipleSignedInt8|optionAllowMultiple=boolean|server=serverID|
```

`server=serverID` only appears if the DHCP vendor client deployment option is used for a specific server.

Related Methods

- *Add DHCP Vendor Deployment Option* on page 119
- *Add Vendor Option Definition* on page 120
- *Add Vendor Profile* on page 121
- *Delete DHCP Vendor Deployment Option* on page 121
- *Update DHCP Vendor Deployment Option* on page 123

Update DHCP Vendor Deployment Option

updateDHCPVendorDeploymentOption() updates the specified DHCP vendor deployment option.

```
void updateDHCPVendorDeploymentOption ( APIDeploymentOption option )
```

Parameters

option—the object ID of the DHCP vendor deployment option to be updated.

Output/Response

None.

Related Methods

- *Add DHCP Vendor Deployment Option* on page 119
- *Add Vendor Option Definition* on page 120
- *Add Vendor Profile* on page 121
- *Delete DHCP Vendor Deployment Option* on page 121
- *Get DHCP Vendor Deployment Option* on page 122

DHCP Match Classes

Match classes in Proteus (also known as *classes* in ISC DHCP and as *user* and *vendor classes* in Microsoft DHCP) allow you to restrict address allocation and assign options to clients that match specified criteria.

For example, using a match class, you could assign a specific DHCP lease length to clients that match a MAC address pattern or clients that are configured to send a specific identifier. A DHCP client becomes a member of a class when it matches the specified criteria.

Proteus provides the following seven match criteria values:

- MATCH_HARDWARE
- MATCH_DHCP_CLIENT_ID
- MATCH_DHCP_VENDOR_ID
- MATCH_AGENT_CIRCUIT_ID
- MATCH_AGENT_REMOTE_ID
- CUSTOM_MATCH
- CUSTOM_MATCH_IF

Add DHCP Match Classes

addDHCPMatchClass() adds DHCP match classes to Proteus.

```
public long addDHCPMatchClass (long configurationId, String name, String
matchCriteria, String properties)
```

Parameters

configurationId—the object ID of the configuration to which the DHCP match class is being added.

name—the name of the DHCP match class.

matchCriteria—a string defining the match criteria. The value must be one of the constants listed in *DHCP Class Match Criteria* on page 209.

properties—a string containing the following elements:

- **description**—a description of the match class.
- **matchOffset**—Match Offset value for the MatchClass. It refers to the point where the match should begin.
- **matchLength**—Match Length value for the Matchclass. It refers to the number of characters to match.

- **matchOffset** and **matchLength** are only applicable to the following five constants:
  - DHCP_CLASS_HARDWARE
  - DHCP_CLASS_CLIENT_ID
  - DHCP_CLASS_VENDOR_ID
  - DHCP_CLASS_AGENT_CIRCUIT_ID
  - DHCP_CLASS_AGENT_REMOTE_ID
- **matchOffset** and **matchLength** must be specified together.

- **customMatchRawString**—a raw string that maps directly to a data or boolean expression for DHCP_CLASS_CUSTOM_MATCH and DHCP_CLASS_CUSTOM_MATCH_IF constants. Use the syntax and grammar supported by the ISC's DHCP daemon. End the string with a ";" semicolon. If you omit the semicolon, one is automatically added when the condition is deployed.

Output/Response

Returns the object ID of the new DHCP match class added.

Update DHCP Match Classes

A DHCP Match class' name property can be updated using the generic **update()** method. For more information, refer to ***Updating Objects*** on page 54.

Parameters

properties—a string containing options, in addition to the following element:

- **ignoreError**—If true, the validation errors for the available match values violating the match conditions will be ignored.

Delete DHCP Match Classes

DHCP Match Classes can be deleted using the generic **delete()** method. For more information, refer to ***Deleting Objects*** on page 56.

Add DHCP Sub Classes

addDHCPSubClass() adds DHCP match class values.

```
public long addDHCPSubClass (long matchClassId, String matchValue, String
properties)
```

Parameters

matchClassId—the object ID of the match class in which the DHCP match class value is being defined.

matchValue—the value of the DHCP match value to be matched with the match class. The length of the match value must be equal to the length, in bytes, specified in the match class.

properties—a string containing the following element:

- **description**—a description of the match class.

Output/Response

Returns the object ID of the new DHCP match class value.

Update DHCP Sub Classes

A DHCP Sub class' name property can be updated using the generic **update()** method. For more information, refer to ***Updating Objects*** on page 54.

Delete DHCP Match Classes

DHCP Sub Classes can be deleted using the generic **delete()** method. For more information, refer to ***Deleting Objects*** on page 56.

DNS

The DNS service part of the Proteus API implements DNS structures through views and zones. All supported DNS resource record types can be manipulated through this part of the API. Control of DNS options allows you to customize the DNS service.

DNS Views

Proteus treats all DNS structures as views. Proteus creates the default view clause in the **named.conf** files and then creates additional views as needed to add customized DNS response groups to the structure. Each view is a child of a configuration object. Beneath the views are zones, which can contain sub-zones or resource records. All views can be associated with an Access Control List (ACL) to filter client requests and provide different sets of DNS information in response to requests from different clients. BIND views and Proteus views are the same. Views in Proteus use the same IP address to deliver different DNS services to different clients depending on their IP address. In a Proteus configuration, you can add views for each client group that requires filtering against an ACL.

Proteus differs from standard implementations of views in the area of zone transfers. Proteus only assigns a single IP address for DNS to a managed server. To send notifications to a slave server to request a zone transfer, Adonis servers recognize each other through the use of TSIG keys. The TSIG keys enable each Adonis appliance to be dealt with on an individual basis with high security in terms of data integrity and authentication. Proteus handles all of the implementation details for these advanced features while providing views-based service on a single port and on a single address.

Add DNS View

addView() adds DNS views.

```
long addView( long configurationId, String name, String properties )
```

Parameters

configurationId—the object ID of the parent configuration in which this DNS view is located.

name—the name of the view.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new DNS view.

Related Methods

- *Update DNS View* on page 126
- *DNS View Generic Methods* on page 127

Update DNS View

A DNS view’s **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add DNS View* on page 126
- *DNS View Generic Methods* on page 127

DNS View Generic Methods

DNS views use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add DNS View* on page 126
- *Update DNS View* on page 126

Add Access Control List (ACL)

public long addACL() adds an Access Control List to a view.

```
public long addACL( long configurationId, String name, String properties )
```

Parameters

configurationId—the object ID of the configuration on which ACL need to be added.

name—the name of the ACL.

properties—a string containing the comma-separated list of options in the following format:

```
aclValues=IP4Address, IP6Address, IP4Network’s CIDR, IP6Network’s CIDR, ACL’s name,
TSIG key’s name, Predefined BIND ACL values
```

- **IP4Address**—the IPv4 address.
- **IP6Address**—the IPv6 address.
- **IP4Network’s CIDR**—the CIDR notation defining the IPv4 network.
- **IP6Network’s CIDR**—the CIDR notation defining the IPv6 network.
- **ACL’s name**—the name of the ACL (Example: acl aclName).
- **TSIG key’s name**—the name of the TSIG key (Example: key TSIGName).
- **Predefined BIND ACL values**—Example: ‘none’, 'any', 'localhost', 'localnets' or 'All', 'None', 'Local Host', 'Local Networks'.

> Use an exclamation mark (!) to exclude a certain option. For example, !none, !acl aclName, !10/24, etc.

Output/Response

Returns the object ID for the new ACL object.

Related Methods

- *Add DNS View* on page 126
- *Update DNS View* on page 126
- *DNS View Generic Methods* on page 127
- *Update Access Control List (ACL)* on page 128

Update Access Control List (ACL)

An Access Control List (ACL) can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

- *Add DNS View* on page 126
- *Update DNS View* on page 126
- *DNS View Generic Methods* on page 127
- *Add Access Control List (ACL)* on page 127

DNS Zones

In Proteus, DNS zones are child objects of views. Zones can contain sub-zones and resource records. Sub-zones can be created several layers deep, as required.

Add Entity for DNS Zones

addEntity() is a generic method for adding configurations, DNS zones, and DNS resource records. When using **addEntity()** to add a zone, you must specify a single zone name without any . (dot) characters. The parent object must be either a DNS view or another DNS zone.

```
long addEntity( long parentId, APIEntity entity )
```

Parameters

parentId—the object ID of the parent DNS view or DNS zone to which the zone is added.

APIEntity—the zone name, without any . (dot) characters, to be added.

Output/Response

Returns the object ID for the new DNS zone.

Related Methods

- *Get Zones by Hint* on page 129
- *Add Zone* on page 129
- *Zone Generic Methods* on page 130
- *Add TFTP File* on page 160

Add Zone

addZone() adds DNS zones. When using **addZone()**, you can use . (dot) characters to create the top-level domain and subzones.

```
long addZone( long parentId, String absoluteName, String properties )
```

Parameters

parentId—the object ID for the parent object to which the zone is being added. For top-level domains, the parent object is a DNS view. For sub zones, the parent object is a top-level domain or DNS zone.

absoluteName—the complete FQDN for the zone with no trailing dot (for example, example.com).

properties—adds object properties, including a flag for deployment, an optional network template association, and user-defined fields in the format:

```
deployable=<true|false>, template=<template id>, <userField>=<userFieldValue>
```

The **deployable** flag is **false** by default and is optional. To make the zone deployable, set the **deployable** flag to **true**.

Output/Response

Returns the object ID for the new DNS zone.

Related Methods

- *Add Entity for DNS Zones* on page 128
- *Get Zones by Hint* on page 129
- *Update Zone* on page 130
- *Zone Generic Methods* on page 130

Get Zones by Hint

getZonesByHint() returns an array of accessible zones of child objects for a given containerId value.

```
public APIEntity[] getZonesByHint( long containerId, int start, int count, String
options )
```

Parameters

containerId—the object ID for the container object. It can be the object ID of any object in the parent object hierarchy. The highest parent object can be the configuration level.

start—indicates where in the list of objects to start returning objects. The list begins at an index of 0.

count—indicates the maximum number of child objects that this method will return. The maximum number of child objects cannot exceed more than 10.

options—a string containing options. The Option names available in the ObjectProperties are *ObjectProperties.hint*, *ObjectProperties.accessRight*, and *ObjectProperties.overrideType*. Multiple options can be separated by a | (pipe) character. For example:

```
hint=ab|overrideType=HostRecord|accessRight=ADD
```

The values for *ObjectProperties.hint* option can be the prefix of a zone name. For example:

```
String options = ObjectProperties.hint + "=abc|"
```

The values for the *ObjectProperties.accessRight* and *ObjectProperties.overrideType* options must be one of the constants listed in *Access Right Values* on page 203 and *Object Types* on page 220. For example:

```
String options = ObjectProperties.accessRight + "=" + AccessRightValues.AddAccess +
"|"+ ObjectProperties.overrideType + "=" + ObjectTypes.HostRecord;
```

Output/Response

Returns an array of zones based on the input argument without their properties fields populated, or returns an empty array if containerId is invalid. If no access right option is specified, the View access level will be used by default.

Related Methods

- *Add Entity for DNS Zones* on page 128
- *Add Zone* on page 129
- *Update Zone* on page 130
- *Zone Generic Methods* on page 130

Update Zone

A DNS zone's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Entity for DNS Zones* on page 128
- *Add Zone* on page 129
- *Get Zones by Hint* on page 129
- *Zone Generic Methods* on page 130

Zone Generic Methods

DNS zones use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Entity for DNS Zones* on page 128
- *Add Zone* on page 129
- *Get Zones by Hint* on page 129
- *Update Zone* on page 130

Get Key Signing Key

getKSK() returns a string containing all active Key Signing Keys (KSK) for a given entityId value in a specified output format with its start time and expire time, divided by a delimiter (|). The list of returned KSKs is sorted in descending order by expiry date.

```
Public String[] getKSK ( long entityId, String format )
```

Parameters

- **entityId**—the object ID of the entity associated with the KSK. The only supported entity types are Zone, IPv4 block, and IPv4 network.
- **format**—the output format of the KSK of an entity. The value must be one of the constants listed in *DNSSEC Key Format* on page 212.

Output/Response

Returns a string containing up-to two active KSK(s) of an entity in the following format:

KSK in specified format|create time|expire time.

DNS Zone Templates

DNS zone templates provide a standard set of DNS records and options that can be maintained in a central location. These templates contain the same tabs as a regular zone except for the **Deployment Roles** tab. Any settings that should be repeated in several zones, such as mail servers or web servers, should be added to a zone template so that the setting can be applied easily and consistently across zones. The records and options set in a zone template are created in any zones to which the template is applied.

If a record or an option is updated in the template, it is also updated in any zones to which the template applies. Modifying a template-applied record or option in a zone severs the link between the template and the modified record or option. If the template is re-applied, you have a choice to ignore conflicts or overwrite the conflicting objects with the settings in the template.

Zone templates can be edited to change the template name. Editing a zone template is exactly the same as building a zone, with the exception of assigning deployment roles, because zone templates are not deployable.

Add Zone Template

addZoneTemplate() adds a DNS zone template.

```
public long addZoneTemplate ( long parentId, String name, String properties )
```

Parameters

parentId—the object ID of the parent DNS view when adding a view-level zone template. The object ID of the configuration when adding a configuration-level zone template.

name—the name of the DNS zone template. This value can be null.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID of the new DNS zone template.

Related Methods

- *Update Zone Template* on page 133
- *Zone Template Generic Methods* on page 133

Assign or Update Template

assignOrUpdateTemplate() assigns, updates, or removes DNS zone and IPv4 network templates.

```
void assignOrUpdateTemplate ( long entityId, long templateId, String properties )
```

Parameters

entityId—the object ID of the IPv4 network to which the network template is to be assigned or updated, or the object ID of the zone to which the zone template is to be assigned or updated.

templateId—the object ID of the DNS zone template or IPv4 network template. To remove a template, set this value to 0 (zero).

properties—a string containing the following settings:

- **ObjectProperties.templateType**—specifies the type of template on which this operation is being performed. The possible values are **ObjectProperties.IP4NetworkTemplateType** (Assigning or updating IP4NetowrkTemplate on an IP4Network) and **ObjectProperties.zoneTemplateType** (Assigning or updating zoneTemplate on a DNS zone). This is mandatory.

Along with **ObjectProperties.templateType**, user can also specify the reapply mode for various properties of the template.

- For Network template, the following additional parameters can also be specified:
  - **ObjectProperties.gatewayReapplyMode**
  - **ObjectProperties.reservedAddressesReapplyMode**
  - **ObjectProperties.dhcpRangesReapplyMode**
  - **ObjectProperties.ipGroupsReapplyMode**
  - **ObjectProperties.optionsReapplyMode**
- For Zone Template, the following additional parameter can also be specified:
  - **ObjectProperties.zoneTemplateReapplyMode**

  The possible values for re-apply mode properties are:
  - **ObjectProperties.templateReapplyModeUpdate**
  - **ObjectProperties.templateReapplyModeIgnore**
  - **ObjectProperties.templateReapplyModeOverwrite**

  If the re-apply mode is not specified in the properties, the default **ObjectProperties.templateReapplyModeIgnore** mode is used.

> If you are not using Java or Perl, refer to *Object Properties* on page 213 for the actual values.

Java client example:

```
EntityProperties ntProp = new EntityProperties();

ntProp.addProperty( ObjectProperties.templateType,
ObjectProperties.IP4NetworkTemplateType );

ntProp.addProperty( ObjectProperties.gatewayReapplyMode,
ObjectProperties.templateReapplyModeUpdate );

ntProp.addProperty( ObjectProperties.reservedAddressesReapplyMode,
ObjectProperties.templateReapplyModeUpdate );

service.assignOrUpdateTemplate( ip4N20_26Id, networkTemplateId,
ntProp.getPropertiesString() );
```

Perl client example:

```
SOAP::Data->type( 'string' )->name( 'properties' )->

value( ObjectProperties::templateType."=".ObjectProperties::
IP4NetworkTemplateType."|".

ObjectProperties:: gatewayReapplyMode."=".ObjectProperties::
templateReapplyModeUpdate."|" )

->attr({xmlns => ''}) )->result;
```

Output/Response

None.

Related Methods

- *Add Zone Template* on page 131
- *Update Zone Template* on page 133
- *Zone Template Generic Methods* on page 133

Update Zone Template

A zone template's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Zone Template* on page 131
- *Zone Template Generic Methods* on page 133

Zone Template Generic Methods

Zone templates use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Zone Template* on page 131
- *Update Zone Template* on page 133

Add Records to DNS Zone Template

addEntity() is a generic method for adding configuration, DNS zones, and DNS resource records. You must use this method to add or update DNS resource records to DNS zone templates. For more information, see *Add Entity for Resource Records* on page 138.

ENUM Zones

ENUM zones provide voice over IP (VoIP) functionality within a DNS server. The system requires DNS to manage the phone numbers associated with client end points; Proteus provides an E164 or ENUM zone type for this purpose. The ENUM zone represents the area code for the phone prefixes and numbers stored in it. ENUM zones contain special sub-zones called prefixes that represent telephone exchanges and can contain the records for the actual devices.

VoIP devices are addressed in several ways. A uniform resource identifier (URI) string provides custom forward locator references for these devices as covered in RFC 3401. Reverse DNS is used to discover the relevant information for a device based on its phone number. Name authority pointer (NAPTR) records are used to represent this information.

Add ENUM Zone

addEnumZone() adds ENUM zones.

```
long addEnumZone( long parentId, long prefix, String properties )
```

Parameters

parentId—the object ID for the parent object of the ENUM zone.

prefix—the number prefix for the ENUM zone.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new ENUM zone.

Related Methods

- *Update ENUM Zone* on page 134
- *ENUM Number Generic Methods* on page 136

Update ENUM Zone

An ENUM zone's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add ENUM Zone* on page 134
- *ENUM Number Generic Methods* on page 136

ENUM Zone Generic Methods

ENUM zones use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add ENUM Zone* on page 134
- *Update ENUM Zone* on page 134

ENUM Numbers

ENUM number objects represent VoIP phone numbers within Proteus. This functionality is provided as an alternative to using raw NAPTR records.

Add ENUM Number

addEnumNumber() adds ENUM numbers.

```
long addEnumNumber( long parentId, int number, String properties )
```

Parameters

parentId—the object ID of the parent object for the ENUM number. The parent object for an ENUM number is always an ENUM zone.

number—the ENUM phone number.

properties—adds object properties, and user-defined fields, including the **data** string, which includes **service**, **URI**, **comment** and **ttl** values.

Output/Response

Returns the object ID for the new ENUM number record.

Related Methods

- *Update ENUM Number* on page 135
- *ENUM Number Generic Methods* on page 136

Update ENUM Number

You can update an ENUM number’s **number** property using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add ENUM Number* on page 135
- *ENUM Number Generic Methods* on page 136

ENUM Number Generic Methods

ENUM numbers use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add ENUM Number* on page 135
- *Update ENUM Number* on page 135

DNS Resource Records

In Proteus, DNS resource records describe the characteristics of a zone or sub-zone. Proteus supports the following types of resource records.

Resource Record Type	Description
Host Record	Designates an IP address for a device. A new host requires a name and an IP address. Multiple addresses may exist for the same device.
Mail Exchanger Record	Designates the host name and preference for a mail server or exchanger. An MX record requires a name and a priority value. Priorities with a lower numeric value are chosen first in assessing delivery options.
CNAME Alias Record	Specifies an alias for a host name. The alias record type requires a name.
Service Record	Defines services available within a zone, such as LDAP. A service record requires a name, priority, port, and weight. A lower priority value indicates precedence. The port value indicates the port on which the service is available. The weight value is used when multiple services have the same priority value; a higher weight value indicates precedence.
HINFO Host Info Record	Specifies optional text information about a host. The host info record includes CPU and OS information.
TXT Text Record	Associates arbitrary text with a host name. A text record includes name and text information. This record is used to support record types such as those used in Sender Policy Framework (SPF) e-mail validation.
Generic Record	The following generic record types are available: A, A6, AAAA, AFSDB, APL, CERT, DHCID, DNAME, DS, IPSECKEY, ISDN, KEY, KX, LOC, MB, MG, MINFO, MR, NS, NSAP, PTR, PX, RP, RT, SINK, SPF, SSHFP, WKS, and X25. These records contain name, type, and value information.
NAPTR Naming Authority Pointer Record	Specifies settings for applications, such as VoIP. These records are used in Proteus to populate ENUM zones.

Reference: Handling dotted resource records

The resource record adder includes two new property parameters, namely, **parentZoneName** and **linkedParentZoneName**. Both parameters should be absolute names. The record absolute name must end with the zone absolute name; otherwise an exception is thrown, and an error message appears. **Record name is not subset of parent zone: abc.bcn.example.com.**

The **linkedParentZoneName** parameter must be used only for adding or updating the linked records for CName, MX and SRV records.

The **parentZoneName** parameter refers to a destination zone for the record to be added to, and applies to all resource record you add. This parameter makes it possible to support dot-separated record names. For example, an API user can add a resource record named **abc.xyz.example.com** to the zone **example.com**.

> You should not use **parentZoneName** when updating records, because you cannot change the parent zone of a record. If you try to do this an exception is thrown.

Example 1

Suppose the zone **example.com** has a dot separated host record **abc.xyz**. A user add a sub-zone called **xyz**, and then adds a host record **abc** to this sub-zone. There are now two host records named **abc.xyz.example.com**.

- If an API user wants to link a CName record to **abc.xyz.example.com**, the linked record will be the one located in the sub-zone, because the user cannot link the record to the one in the parent zone. To allow API users to choose whatever parent zone they want, use the **linkedParentZoneName** parameter.

> Use this parameter with CName, MX and SRV records. It cannot be used as metadata fields for these records. If this parameter is used in updating other resource records, an error occurs.

Example 2

An API entity CName **abcName.example.com** has the following property string:

ttl=123|absoluteName=abcName.example.com|linkedRecordName=bcnhost.dot.bcn.com|

An API user wants to change the linked record name to **abc.bcn.example.com**. The user applies the following updates to the property string:

linkedParentZoneName=example.com|absoluteName=abcName.example.com|ttl=123|linkedRecordName=abc.bcn.example.com|

- If the API user does not use the **linkedParentZoneName** parameter, Proteus chooses the internal host record or alias record if it exists; otherwise it chooses the external host record for the linked record.

Generic Resource Records

Each resource record type has a specialized add method, but there are two general methods for adding resource records in the Proteus API:

- *Add Resource Record*
- *Add Entity for Resource Records*

Add Resource Record

addResourceRecord() is a generic method for adding resource records of any kind by specifying the name, type, and rdata arguments.

```
long addResourceRecord( long viewId, String absoluteName, String type, String
rdata, long ttl, String properties )
```

Parameters

viewId—the object ID for the parent view to which the resource record is being added.

absoluteName—the absolute name of the record, specified as an FQDN.

type—the type of record being added. Valid values for this parameter are the resource record types shown in *Object Types* on page 220:

- AliasRecord
- ExternalHostRecord
- GenericRecord
- HINFORecord
- HostRecord
- MXRecord
- NAPTRRecord
- SRVRecord
- TXTRecord

rdata—the data for the resource record in BIND format (for example, for A records, A 10.0.0.4). You can specify either a single IPv4 or IPv6 address for the record.

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new resource record.

Related Methods

- *Add Entity for Resource Records* on page 138

Add Entity for Resource Records

addEntity() is a generic method for adding configurations, DNS zones, and DNS resource records. Use this method to add resource records that have . (dot) characters in their names.

```
long addEntity( long parentId, APIEntity entity )
```

In order to add DNS resource records to a DNS zone template, you must use the **addEntity()** method.

Parameters

parentId—the object ID of the parent zone to which the record is added.

APIEntity—the resource record object being passed to the database.

Output/Response

Returns the object ID for the new resource record.

Related Methods

- *Add Resource Record* on page 137

Move Resource Records

moveResourceRecord() moves resource records between different zones that already exist.

```
public void moveResourceRecord( long resourceRecordId, String destinationZone )
```

Parameters

resourceRecordId—the object ID of the resource record to be moved.

destinationZone—fully qualified domain name of the destination DNS zone to which the resource record will be moved.

Output/Response

None.

Related Methods

- *Add Resource Record* on page 137
- *Add Entity for Resource Records* on page 138

NAPTR Records

NAPTR records specify settings for applications, such as VoIP. Proteus uses NAPTR records to populate ENUM zones.

Add NAPTR Record

addNAPTRRecord() adds NAPTR records.

```
long addNAPTRRecord( long viewId, String absoluteName, int order, int preference,
String service, String regexp, String replacement, String flags, long ttl, String
properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which this record is added.

absoluteName—the FQDN for the record.

order—specifies the order in which NAPTR records are read if several are present and are possible matches. The lower **order** value takes precedence.

preference—specifies the order in which NAPTR records are read if the **order** values are the same in multiple records. The lower **preference** value takes precedence.

service—specifies the service used for the NAPTR record. Valid settings for this parameter are listed in *ENUM Services* on page 223.

regexp—a regular expression, enclosed in double quotation marks, used to transform the client data. If a regular expression is not specified, a domain name must be specified in the **replacement** parameter.

replacement—specifies a domain name as an alternative to the **regexp**. This parameter replaces client data with a domain name.

flags—an optional parameter used to set flag values for the record.

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new NAPTR resource record.

Related Methods

- *Update NAPTR Record* on page 140
- *NAPTR Record Generic Methods* on page 140

Update NAPTR Record

A NAPTR record's **name**, **ttl**, **comment**, **order**, **preference**, **service**, **regexp**, and **replacement** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add NAPTR Record* on page 139
- *NAPTR Record Generic Methods* on page 140

NAPTR Record Generic Methods

NAPTR records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add NAPTR Record* on page 139
- *Update NAPTR Record* on page 140

External Host Records

External hosts, specified with an FQDN, represent host names that reside outside the Proteus-managed servers. External hosts are never deployed, but act as glue that lets other records (such as MX and CNAME records) link to hosts that are not managed by Proteus. Any references within a view that refer to entities completely outside of the IP space and DNS namespace managed by Proteus are defined here.

Add External Host Record

addExternalHostRecord() adds external host records.

```
long addExternalHostRecord( long viewId, String name, String properties )
```

This method will add the external host record under a zone. In order to add external host records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which this record is being added.

absoluteName—the FQDN for the host record.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new external host record.

Related Methods

- *Update External Host Record* on page 141
- *External Host Record Generic Methods* on page 141

Update External Host Record

An external host record's **name** and **comment** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add External Host Record* on page 140
- *External Host Record Generic Methods* on page 141

External Host Record Generic Methods

External host records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add External Host Record* on page 140
- *Update External Host Record* on page 141

Host Records

A host record, or A record, designates an IP address for a device. A new host requires a name and an IP address. Multiple addresses may exist for the same device. Set the time-to-live for this record to an override value here so that the record has a longer or shorter ttl. A comment field is also included.

Add Host Record

addHostRecord() adds host records for IPv4 or IPv6 addresses. All addresses must be valid addresses. When adding a host record, the *reverseRecord* property, if not explicitly set in the **properties** string, is set to **true** and Proteus creates a reverse record automatically. IPv4 addresses can be added in both workflow and non-workflow mode. IPv6 addresses can be added in non-workflow mode only. For more information on workflow mode, see *Set Workflow Level* on page 183.

```
long addHostRecord( long viewId, String absoluteName, String addresses, long ttl,
String properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which this record is added.

absoluteName—the FQDN for the host record.

addresses—a list of comma-separated IP addresses (for example, 10.0.0.5,130.4.5.2).

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new host resource record.

Related Methods

- *Get Host Record by Hint* on page 143
- *Get Dependent Records* on page 144
- *Update Host Record* on page 144
- *Host Record Generic Methods* on page 144

Add Bulk Host Records

This method adds host records to a zone linked to a DNS naming policy, each with an IP address auto-incremented starting from a specific address in a network.

addBulkHostRecord () adds host records using auto-increment from the specific starting address.

```
APIEntity[] addBulkHostRecord ( long viewId, String absoluteName, long ttl, long
networkId, String startAddress, int numberOfAddresses, String properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the view ID.

absoluteName—the FQDN name for the host record in naming policy format.

ttl—value of the Time To Live.

networkId—the network which to get the available IP addresses. Each address is used for one host record.

startAddress—the starting IPv4 address for getting the available addresses.

numberOfAddresses—the number of addresses.

properties—excludeDHCPRange=true/false, if true then IP addresses within a DHCP range will be skipped. This argument can also contain user-defined fields.

Output/Response

Returns an array of host record APIEntity objects based on available addresses and number of IP addresses required. If no addresses are available an exception is thrown.

Related Methods

- *Add Host Record* on page 141
- *Get Host Record by Hint* on page 143

Get Host Record by Hint

getHostRecordsByHint() returns an array of objects with host record type.

```
public APIEntity[] getHostRecordsByHint ( int start, int count, String options )
```

Parameters

start—indicates where in the list of objects to start returning objects. The list begins at an index of 0.

count—indicates the maximum of child objects that this method will return. The value must be less than or equal to 10.

options—a string containing options. The supported options are *hint* and *retrieveFields*. Multiple options can be separated by a | (pipe) character. For example:

```
hint=^abc|retrieveFields=false
```

If the hint option is not specified in the string, searching criteria will be based on the same as zone host record. The following wildcards are supported in the *hint* option.

- ^—matches the beginning of a string. For example: **^ex** matches **ex**ample but not t**ex**t.
- $—matches the end of a string. For example: **ple$** matches exam**ple** but not **ple**ase.
- ^ $—matches the exact characters between the two wildcards. For example: **^example$** only matches **example**.
- ?—matches any one character. For example: **ex?t** matches exit.
- *—matches one or more characters within a string. For example: **ex*t** matches **exit** and **ex**cellen**t**.

The default value for the *retrieveFields* option is set to false. If the option is set to true, user-defined field will be returned. If the options string does not contain *retrieveFields*, user-defined field will not be returned.

Output/Response

Returns an array of host record APIEntity objects.

Related Methods

- *Add Host Record* on page 141
- *Add Bulk Host Records* on page 142

Get IP Address with Host Records

getNetworkLinkedProperties (long networkId) returns an IP address with linked host records.

```
APIEntity[] getNetworkLinkedProperties( long networkId )
```

Parameters

networkId—network ID

Output/Response

Returns an array of IP address APIEntity objects with their linked host records. The output has the following format:

hostId:hostName:zoneId:zoneName:viewId:viewName:hasAlias;

Related Methods

- *Get IPv4 Address* on page 86
- *Add Host Record* on page 141

Get Dependent Records

> This method is deprecated. Using this method now returns an error message. Use the **getLinkedEntities()** method instead. For more information, see *Get Linked Entities* on page 56.

getDependentRecords() returns any CNAME, MX, or SRV resource records that reference the specified host record.

```
APIEntity[] getDependentRecords( long entityId, int start, int count )
```

Parameters

entityId—the object ID for the host record for which you want to retrieve dependent records.

start—indicates where in the list of dependent records to start returning objects. This list begins at an index of 0.

count—the maximum number of dependent records to return.

Output/Response

Returns an array of APIEntity objects representing the CNAME, MS, or SRV records referencing the specified host record.

Related Methods

- *Add Host Record* on page 141
- *Update Host Record* on page 144
- *Host Record Generic Methods* on page 144

Update Host Record

A host records's **name**, **ttl**, **comment**, and **addresses** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Host Record* on page 141
- *Get Dependent Records* on page 144
- *Host Record Generic Methods* on page 144

Host Record Generic Methods

Host records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Host Record* on page 141
- *Get Dependent Records* on page 144
- *Update Host Record* on page 144

Alias Records

This is a Canonical Name or CNAME record, used to specify an alias for a host name. The alias record type requires only a name to be supplied. The time-to-live for this record can be set to an override value so that this record has a longer or shorter ttl. A comment field is also included.

Add Alias Record

addAliasRecord() adds alias records. This method attempts to link to an existing host record. If an existing host record cannot be located, the method attempts to link to an existing external host record. If neither can be located, the method fails.

```
long addAliasRecord( long viewId, String absoluteName, String linkedRecordName,
long ttl, String properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which this record is being added.

absoluteName—the FQDN of the alias.

linkedRecordName—the name of the record to which this alias will link.

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new alias resource record.

Related Methods

- *Update Alias Record* on page 146
- *Alias Record Generic Methods* on page 146

Get Aliases by Hint

getAliasesByHint() returns an array of CNAMEs with linked record name.

```
public APIEntity[] getAliasesByHint ( int start, int count, String options )
```

Parameters

start—indicates where in the list of objects to start returning objects. The list begins at an index of 0.

count—indicates the maximum of child objects that this method will return. The value must be less than or equal to 10.

options—a string containing options. The supported options are *hint* and *retrieveFields*. Multiple options can be separated by a | (pipe) character. For example:

```
hint=^abc|retrieveFields=false
```

If the hint option is not specified in the string, searching criteria will be based on the same as zone alias. The following wildcards are supported in the *hint* option.

- ^—matches the beginning of a string. For example: **^ex** matches **ex**ample but not t**ex**t.
- $—matches the end of a string. For example: **ple$** matches exam**ple** but not **ple**ase.
- ^ $—matches the exact characters between the two wildcards. For example: **^example$** only matches **example**.
- ?—matches any one character. For example: **ex?t** matches exit.
- *—matches one or more characters within a string. For example: **ex*t** matches **exit** and **ex**cellen**t**.

The default value for the *retrieveFields* option is set to false. If the option is set to true, user-defined field will be returned. If the options string does not contain *retrieveFields*, user-defined field will not be returned.

Output/Response

Returns an array of Alias APIEntity objects.

Update Alias Record

An alias record's **name**, **ttl**, **comment**, and **linked record** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Alias Record* on page 145
- *Alias Record Generic Methods* on page 146

Alias Record Generic Methods

Alias records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* **on page 48** and *Deleting Objects* **on page 56**.

Related Methods

- *Add Alias Record* on page 145
- *Update Alias Record* on page 146

Text Records

Text records associate arbitrary text with a host name. This record features **name** and **text** fields and is used to support record types such as those used in SPF email validation. The time-to-live for this record can be set to an override value here so that this record has a longer or shorter ttl. A comment field is also included.

Add Text Record

addTXTRecord() adds TXT records.

```
long addTXTRecord( long viewId, String absoluteName, String txt, long ttl, String
properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which the record is being added.

absoluteName—the FQDN of the text record.

txt—the text data for the record.

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new TXT record.

Related Methods

- *Update Text Record* on page 147
- *Text Record Generic Methods* on page 147

Update Text Record

A text record's **name**, **ttl**, **comment**, and **text data** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Text Record* on page 147
- *Text Record Generic Methods* on page 147

Text Record Generic Methods

Text records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Text Record* on page 147
- *Update Text Record* on page 147

HINFO Records

The Host Info or HINFO resource record contains optional text information about a host. The standard version of this record has **name**, **cpu** and **os** fields. The time-to-live for this record can be set to an override value here so that it has a longer or shorter ttl. A comment field is also included. This record type is a good candidate for user-defined fields to track information about hosts on the network.

Add HINFO Record

addHINFORecord() adds HINFO records.

```
long addHINFORecord( long viewId, String absoluteName, String cpu, String os, long
ttl, String properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which the HINFO record is being added.

absoluteName—the FQDN of the HINFO record.

cpu—a string providing central processing unit information.

os—a string providing operating system information.

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new HINFO resource record.

Related Methods

- *Update HINFO Record* on page 148
- *HINFO Record Generic Methods* on page 148

Update HINFO Record

An HINFO record's **name**, **ttl**, **comment**, **cpu**, and **os** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add HINFO Record* on page 148
- *HINFO Record Generic Methods* on page 148

HINFO Record Generic Methods

HINFO records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add HINFO Record* on page 148
- *Update HINFO Record* on page 148

MX Records

A Mail Exchanger or MX record designates the host name and preference value for a mail server or exchanger, as defined in RFC 974. An MX Record requires a name and a priority value. Priorities with a lower numeric value are chosen first in assessing delivery options. The time-to-live for this record can be set to an override value, so that this record has a longer or shorter ttl. A comment field is also included. This method attempts to link to an existing host record. If an existing host record cannot be located, the method attempts to link to an existing external host record. If neither can be located, the method fails.

Add MX Record

addMXRecord() adds MX records.

```
long addMXRecord( long viewId, String absoluteName, int priority, String
linkedRecordName, long ttl, String properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which the MX record is being added.

absoluteName—the FQDN for the record.

priority—specifies which mail server to send clients to first when multiple matching MX records are present. Multiple MX records with equal priority values are referred to in a round-robin fashion.

linkedRecordName—the FQDN of the host record to which this MX record is linked.

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new MX resource record.

Related Methods

- *Update MX Record* on page 149
- *MX Record Generic Methods* on page 149

Update MX Record

An MX record's **name**, **ttl**, **comment**, **linked record**, and **priority** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add MX Record* on page 149
- *MX Record Generic Methods* on page 149

MX Record Generic Methods

MX records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add MX Record* on page 149
- *Update MX Record* on page 149

SRV Records

Service records define services that are available within the zone, such as LDAP. An SRV record requires a name by which it is known in Proteus. The time-to-live for this record can be set to an override value here so that this record has a longer or shorter ttl. A comment field is also included. This method attempts to link to an existing host record. If an existing host record cannot be located, the method attempts to link to an existing external host record. If neither can be located, the method fails.

Add SRV Record

addSRVRecord() adds SRV records.

```
long addSRVRecord( long viewId, String absoluteName, int priority, int port, int
weight, String linkedRecordName, long ttl, String properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which the SRV record is being added.

absoluteName—the FQDN of the SRV record.

priority—specifies which SRV record to use when multiple matching SRV records are present. The record with the lowest value takes precedence.

port—the TCP/UDP port on which the service is available.

weight—if two matching SRV records within a zone have equal priority, the weight value is checked. If the weight value for one object is higher than the other, the record with the highest weight has its resource records returned first.

linkedRecordName—the FQDN of the host record to which this SRV record is linked.

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new SRV record.

Related Methods

- *Update SRV Record* on page 150
- *SRV Record Generic Methods* on page 151

Update SRV Record

An SRV record's **name**, **ttl**, **comment**, **linked record**, **priority**, **port**, and **weight** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add SRV Record* on page 150
- *SRV Record Generic Methods* on page 151

SRV Record Generic Methods

SRV records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add SRV Record* on page 150
- *Update SRV Record* on page 150

Start of Authority Records

Start of Authority (SOA) records are used to define administrative information relating to particular zones.

Add Start of Authority Record

addStartOfAuthority() adds SOA records through the Proteus API:

```
long addStartOfAuthority( long parentId, String email, long refresh, long retry,
long expire, long minimum, String properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

parentId—the object ID of the parent object of the SOA record.

email— specifies the email address of the administrator for the zones to which the SOA applies.

refresh—the amount of time that a slave server waits before attempting to refresh zone files from the master server. This is specified in seconds using a 32-bit integer value. A value between 1200 and 4300 seconds is recommended in RFC 1912.

retry—specifies the amount of time that the slave server should wait before re-attempting a zone transfer from the master server after the refresh value has expired. This is specified as a number of seconds using a 32-bit integer value.

expire—specifies the length of time that a slave server will use a non-updated set of zone data before it stops sending queries. This is specified as a number of seconds using a 32-bit integer. A value of 1209600 to 2419200 or 2 to 4 weeks is recommended in RFC 1912.

minimum—specifies the maximum amount of time that a negative cache response is held in cache. A negative cache response is a response to a DNS query that does not return an IP address (a failed request). Until this value expires, queries for this DNS record return an error. The maximum value for this field is 10800 seconds, or 3 hours.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new SOA record.

Related Methods

- *Update Start of Authority Record* on page 152
- *Start of Authority Record Generic Methods* on page 152

Update Start of Authority Record

An SOA record's **name**, **email**, **refresh**, **retry**, **expire**, and **minimum** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Start of Authority Record* on page 151
- *Start of Authority Record Generic Methods* on page 152

Start of Authority Record Generic Methods

SOA records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Start of Authority Record* on page 151
- *Update Start of Authority Record* on page 152

Generic Records

Use the generic resource record methods to add and update the following resource record types: A6, AAAA, AFSDB, APL, CERT, DNAME, DNSKEY, DS, ISDN, KEY, KX, LOC, MB, MG, MINFO, MR, NS, NSAP, PX, RP, RT, SINK, SSHFP, WKS, and X25.

The fields available are **name**, **type** (which defines the custom record type), and **data** (the rdata value for the custom type). The time-to-live for this record can be set to an override value, so the record has a longer or shorter ttl. A comment field is also included.

Add Generic Record

addGenericRecord() adds Generic records.

```
long addGenericRecord( long viewId, String absoluteName, String type, String rdata,
long ttl, String properties )
```

This method will add the record under a zone. In order to add records under templates, you must use *Add Entity for Resource Records* on page 138.

Parameters

viewId—the object ID for the parent view to which the record is being added.

absoluteName—the FQDN of the record.

type—the type of record being added. Valid settings for this parameter are the generic resource record types supported in Proteus: A6, AAAA, AFSDB, APL, CERT, DNAME, DNSKEY, DS, ISDN, KEY, KX, LOC, MB, MG, MINFO, MR, NS, NSAP, PX, RP, RT, SINK, SSHFP, WKS, and X25.

rdata—the data for the resource record in BIND format (for example, for A records, A 10.0.0.4).

ttl—the time-to-live value for the record. To ignore the ttl, set this value to **-1**.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new generic resource record.

Related Methods

- *Update Generic Record* on page 153
- *Generic Record Generic Methods* on page 153

Update Generic Record

A generic record's **name**, **type**, **rdata**, **ttl**, and **comment** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Generic Record* on page 152
- *Generic Record Generic Methods* on page 153

Generic Record Generic Methods

Generic records use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Generic Record* on page 152
- *Update Generic Record* on page 153

DNS Options

DNS options define the deployment of Proteus DNS services. Proteus supports most of the options used by both BIND and Microsoft DNS.

Add DNS Option

addDNSDeploymentOption() adds DNS options.

```
long addDNSDeploymentOption( long entityId, String name, String value, String
properties )
```

Parameters

entityId—the object ID for the entity to which this deployment option is being added.

name—the name of the DNS option being added. This name must be one of the constants listed in *DNS Options* on page 210.

value—the value being assigned to the option.

properties—adds object properties, including comments and user-defined fields.

Related Methods

- *Get DNS Option* on page 154
- *Update DNS Option* on page 154
- *Delete DNS Option* on page 154

Get DNS Option

getDNSDeploymentOption() retrieves DNS options.

```
APIDeploymentOption getDNSDeploymentOption( long entityId, String name, long
serverId )
```

Parameters

entityId—the object ID for the entity to which the deployment option is assigned.

name—the name of the DNS option. This name must be one of the constants listed in *DNS Options* on page 210.

serverId—specifies the server to which this option is assigned. To retrieve an option that has not been assigned to a server role, set this value to **0**(zero). Omitting this parameter from the method call will result in an error.

Related Methods

- *Add DNS Option* on page 153
- *Update DNS Option* on page 154
- *Delete DNS Option* on page 154

Update DNS Option

updateDNSDeploymentOption() updates DNS options.

```
void updateDNSDeploymentOption( APIDeploymentOption option )
```

Parameters

option—the object ID of the DNS option to be updated.

Related Methods

- *Add DNS Option* on page 153
- *Get DNS Option* on page 154
- *Delete DNS Option* on page 154

Delete DNS Option

deleteDNSDeploymentOption() deletes DNS options.

```
void deleteDNSDeploymentOption( long entityId, String name, long serverId )
```

Parameters

entityId—the object ID for the entity to which the deployment option is assigned.

name—the name of the DNS option being deleted. This name must be one of the constants listed in *DNS Options* on page 210.

serverId—specifies the server to which the option is assigned. To delete an option that has not been assigned to a server role, set this value to **0** (zero). Omitting this parameter from the method call will result in an error.

Related Methods

- *Add DNS Option* on page 153
- *Get DNS Option* on page 154
- *Update DNS Option* on page 154

DNS Response Policies

The Response Policies feature allows users to manage a recursive DNS resolver attempting to respond to the queries that might not be desirable or legal. You can set the types of response policies based on your needs and deploy to a DNS server managed under Proteus. By setting up these response policies, you can block, redirect, or allow particular domain name queries that you wish to and must prevent. For example:

- If you are a corporate user and want to prevent employees from being connected to any harmful website, you can setup the response policies and block these harmful websites so that they does not return the query response or the employees can simply be redirected to an appropriate website.
- If you need to follow a government regulation that mandates certain DNS blocking, the response policies can be used to implement this requirement.

There are three different types of response policies that can be set based on user requirements:

- **Blacklist**—Matching items in the list of blacklist object return an NXDomain result.
- **Blackhole**—Matching items in this response policy object return a NOERROR result with no answers.
- **Whitelist**—Matching items in this response policy object are excluded from further processing.

Add Response Policy

addResponsePolicy() adds a DNS response policy.

```
public long addResponsePolicy (long configurationId, String name, String
responsePolicyType, long ttl, String properties)
```

Parameters

configurationId—the object ID of the configuration to which the response policy is being added.

name—the name of the DNS response policy being added.

responsePolicyType—the type of response policy being added. The available values are *BLACKLIST*, *BLACKHOLE* and *WHITELIST*.

> The **responsePolicyType** values need to be in *CAPITAL* letters.

ttl—the time-to-live value in seconds.

properties—a string containing options, including comments and user-defined fields.

Output/Response

Returns the object ID of the new DNS response policy added.

Related Methods

- *Upload Response Policy Item* on page 156.

Update Response Policy

A response policy's ***name***, ***ttl*** and ***responsePolicyType*** properties can be updated using the generic **update()** method. For more information, refer to *Updating Objects* on page 54.

Response Policy Generic Methods

Response policy uses the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Upload Response Policy Item

uploadResponsePolicyItems() uploads one response policy file containing a list of fully qualified domain names (FQDNs).

```
public void uploadResponsePolicyItems (long parentId, byte[] policyItemsData)
```

Parameters

parentId—the object ID of the parent response policy under which the response policy item file is being uploaded.

policyItemsData—the file to be uploaded under the response policy. This file is passed to Proteus as a byte array.

This actual file size should be no more than 75 MB.

Output/Response

None.

Related Methods

- *Add Response Policy* on page 155.

Deployment options

Deployment is the process by which the configuration in Proteus becomes a running set of services on Proteus-managed servers, and Deployment Options define the deployment of Proteus DNS and DHCP services. Deployment options can be applied at many different levels within a configuration such as server, block, network, DNS Views, or DNS Zones level.

Getting deployment options

This is the generic API method for getting Deployment options for Proteus DNS and DHCP services.

Get Deployment Options

getDeploymentOptions() returns DNS and DHCP deployment options.

```
public APIDeploymentOption[] getDeploymentOptions (long entityId, String
optionTypes, long serverId )
```

Parameters

entityId—the object ID of the entity to which the DNS or DHCP deployment option is assigned.

optionTypes—the type of deployment options. Multiple options can be separated by a | (pipe) character. This value must be one of the items listed in *DHCP Deployment Role Types* on page 203.

- Invalid deployment option types will be ignored. For example, if the user passes DHCPv6ClientOtion for IPv4 networks, it will be ignored as it is not a valid deployment option for IPv4 network.
- If specified as null, all deployment options for the specified entity will be returned.

serverId—the specific server to which options are deployed. The valid values are as follows:

- **>0**—returns only the options that are linked to the specified server ID.
- **<0**—returns all options regardless of the server ID specified.
- **=0**—returns only the options that are linked to all servers.

Output/Response

Returns the requested deployment option(s) from the database.

Related Methods

- *Get DHCP Client Option* on page 110.
- *Get DHCP6 Client Option* on page 112.
- *Get DHCP Service Option* on page 116.
- *Get DHCP6 Service Option* on page 118.
- *Get DHCP Vendor Deployment Option* on page 122.
- *Get DNS Option* on page 154.

TFTP

Proteus uses Trivial File Transfer Protocol (TFTP) to provide configuration files to end-point devices.

TFTP Groups

TFTP files are organized into a list of tree structures. Each tree has a root, called a TFTP group. The leaves of this tree are files, and the nodes of the tree are folders. TFTP groups are child objects of configurations. Each tree structure reflects the directory structure on a target TFTP server.

Add TFTP Group

addTFTPGroup() adds TFTP groups.

```
long addTFTPGroup( long configurationId, String name, String properties )
```

Parameters

configurationId—the object ID of the configuration to which the TFTP group is being added.

name—the name of the TFTP group.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new TFTP group.

Related Methods

- *Update TFTP Group* on page 158
- *TFTP Group Generic Methods* on page 158

Update TFTP Group

A TFTP group’s **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add TFTP Group* on page 158
- *TFTP Group Generic Methods* on page 158

TFTP Group Generic Methods

TFTP groups use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add TFTP Group* on page 158
- *Update TFTP Group* on page 158

TFTP Folders

TFTP folders are used to create the directory structure on the TFTP server.

Add TFTP Folder

addTFTPFolder() adds TFTP folders.

```
long addTFTPFolder( long parentId, String name, String properties )
```

Parameters

parentId—the object ID of the parent object of the TFTP folder. The parent is either a TFTP group or another TFTP folder object.

name—the name of the TFTP folder.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new TFTP folder.

Related Methods

- *Update TFTP Folder* on page 159
- *TFTP Folder Generic Methods* on page 159

Update TFTP Folder

A TFTP folder's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add TFTP Folder* on page 159
- *TFTP Folder Generic Methods* on page 159

TFTP Folder Generic Methods

TFTP folders use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add TFTP Folder* on page 159
- *Update TFTP Folder* on page 159

TFTP Files

TFTP files contain configuration information that is passed to the client end-point devices to be configured.

Add TFTP File

addTFTPFile() adds TFTP files.

```
long addTFTPFile( long parentId, String name, String version, byte[] data, String
properties )
```

Parameters

parentId—the object ID of the parent object of the TFTP file. The parent will always be a TFTP folder.

name—the name of the TFTP file.

version—the version of the file. This parameter is optional.

data—the file to be uploaded and distributed to clients by TFTP. The file is passed to Proteus as a byte array.

properties—adds object properties, including comments and user-defined fields.

Output/Response

Returns the object ID for the new TFTP file.

Related Methods

- *Update TFTP File* on page 160
- *TFTP File Generic Methods* on page 160

Update TFTP File

A TFTP file's **name**, **version**, **description**, and **data** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add TFTP File* on page 160
- *TFTP File Generic Methods* on page 160

TFTP File Generic Methods

TFTP files use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add TFTP File* on page 160
- *Update TFTP File* on page 160

Servers and Deployment

Proteus deploys settings and services separately from designing and configuring the actual services. Server objects must be added to Proteus, and then server roles can associate various DNS, DHCP, and TFTP services to the servers on which they will run. Deployment can be scheduled within the Proteus interface, but only immediate deployments can be performed through the Proteus API.

Servers

To use an Adonis server with Proteus, you must add the server to the Proteus configuration. This involves providing information in the Add Server screen, and then connecting to the server, or using the **addServer()** method. A successful connection places the Adonis server under the control of Proteus and disables the native Adonis command server agent. As a result, the server is no longer managed through the Adonis Management Console and responds only to commands from Proteus.

Add Server

Servers are added to a Proteus configuration so that services can be deployed to them using deployment roles and server options. Servers can be added with the API or in the Proteus GUI without connecting to them if the server objects need to be created and configured before the actual servers are available. The method described here adds the server to a Proteus configuration only, and does not connect to the server.

Before deploying a configuration, you must connect to servers added using this method or using the Proteus web interface. For more information about connecting to existing servers using the Proteus web interface refer to the *Proteus Administration Guide* and the online help. Server control is available through the Proteus web interface and the Proteus Administration Console.

Proteus v4.0.6 and Adonis v7.0.0 feature support for ***dedicated management*** on multi-interface Adonis appliances (Adonis hardware models 1900, 1925, and 1950). These Adonis appliances include support for three interface ports (Services, XHA, Management) and four interface ports (Services, XHA, Management, and Redundancy through port bonding: eth0 + eth3).

The procedure for configuring an Adonis server and adding it to Proteus will vary according to the number of interfaces on your Adonis appliance, and the number of interfaces that you wish to utilize.

The following table describes the interfaces that are being used by different types of Adonis.

Number of ports	eth0	eth1	eth2	eth3
2-port Adonis	Services / Management	XHA	N/A	N/A
3-port Adonis	Services	XHA	Dedicated Management	N/A
4-port Adonis	Services	XHA	Dedicated Management	Redundancy

If you are using multi-port Adonis appliances and want to use ***dedicated management***, you must enable it from the Administration Console before adding an Adonis server to Proteus.

addServer() adds servers to Proteus.

```
long addServer( long configurationId, string name, string defaultInterfaceAddress,
string fullHostName, string profile, string properties )
```

Existing customers who have upgraded their Proteus API to v4.0.6 may need to update their API calls to add a server with dedicated management enabled. For more information, refer to KB-939.

Parameters

configurationId—the object ID of the configuration to which the server is being added.

name—the name of the server within Proteus.

defaultInterfaceAddress—the physical IP address for the server within Proteus.

fullHostName—the DNS FQDN by which the server is referenced.

profile—the server capability profile. The profile describes the type of server or appliance being added and determines the services that can be deployed to this server. This must be one of the constants found in *Server Capability Profiles* on page 226.

properties—a string containing the following options:

> - For Adonis servers without multi-port support, the interface-related property options will be ignored.

- **connected**—either *true* or *false*; indicates whether or not to connect to a server. In order to add and configure multi-port Adonis servers, this option must be set to *true*. If *false*, other interface property options will be ignored.
- **password**—the server password (by default, **bluecat**).
- **servicesIPv4Address**—IPv4 address used only for services traffic such as DNS, DHCP, DHCPv6, and TFTP. If ***dedicated management*** is enabled, this option must be specified.
- **servicesIPv4Netmask**—IPv4 netmask used only for services traffic such as DNS, DHCP, DHCPv6, and TFTP. If ***dedicated management*** is enabled, this option must be specified.
- **servicesIPv6Address**—IPv6 address used only for services traffic such as DNS, DHCP, DHCPv6, and TFTP. This is *optional*.
- **servicesIPv6Subnet**—IPv6 subnet used only for services traffic such as DNS, DHCP, DHCPv6, and TFTP. This is *optional*.
- **xhaIPv4Address**—IPv4 address used for XHA. This is *optional*.
- **xhaIPv4Netmask**—IPv4 netmask used for XHA. This is *optional*.
- **redundancyScenario**—networking redundancy scenarios. The possible values are *ACTIVE_BACKUP (Failover))*and *IEEE_802_3AD (Load Balancing)*.

Output/Response

Returns the object ID for the new server.

Related Methods

- *Deploy Server* on page 164
- *Server Generic Methods* on page 167
- *Get Servers Associated with a Deployment Role* on page 168
- *Get Published Interface* on page 168

Import Server

importServer() is used to import Windows DNS or DHCP services from Managed Windows servers.

```
void importServer( long serverId, boolean importDns, boolean importDhcp, string
properties )
```

Parameters

long serverId—the server Id

> You must set at least one of the following Boolean parameters, but you cannot set both of them to **False**.

boolean importDns—import DNS service if true

boolean importDhcp—import DHCP service if true

String properties—reserved for future use.

Output/Responses

Returns void.

Related Methods

- *Add Server* on page 161
- *Replace Server* on page 163
- *Deploy Server* on page 164
- *Server Generic Methods* on page 167

Replace Server

replaceServer() allows you to replace a server.

```
void replaceServer( long serverId, string name, string defaultInterfaceAddress,
string fullHostName, string password, boolean upgrade, string properties)
```

Parameters

serverId—the object ID of the server that needs to be replaced.

name—name of the server to be replaced.

defaultInterfaceAddress—management interface address for the server.

fullHostName—the DNS FQDN by which the server is referenced.

password—the server password (by default, **bluecat**).

upgrade—flag indicating that server needs to be upgraded or not. True means server needs to be upgraded.

properties—a string containing the following options:

> For Adonis servers without multi-port support, the interface-related property options will be ignored.

- **servicesIPv4Address**—IPv4 address used only for services traffic such as DNS, DHCP, DHCPv6 and TFTP. If ***dedicated management*** is enabled, this option must be specified. If ***dedicated management*** is disabled, this address must be the same as defaultInterfaceAddress which is management interface address.
- **servicesIPv4Netmask**—IPv4 netmask used only for services traffic such as DNS, DHCP, DHCPv6 and TFTP. If ***dedicated management*** is enabled, this option must be specified. If ***dedicated management*** is disabled, this netmask address must be the same as the management interface netmask address.
- **servicesIPv6Address**—IPv6 address used only for services traffic such as DNS, DHCP, DHCPv6 and TFTP. This is *optional*.
- **servicesIPv6Subnet**—IPv6 subnet used only for services traffic such as DNS, DHCP, DHCPv6 and TFTP. This is *optional*.
- **xhaIPv4Address**—IPv4 address used for XHA. This is *optional*.
- **xhaIPv4Netmask**—IPv4 netmask used for XHA. This is *optional*.
- **redundancyScenario**—networking redundancy scenarios. The possible values are *ACTIVE_BACKUP (Failover)* and *IEEE_802_3AD (Load Balancing)*.
- **resetServices**—allows you to replace the Adonis server while maintaining existing configurations for DNS, DHCP, and TFTP services. Define this option only if you have modified the IPv4 or IPv6 addresses of the Services interface or wish to reset configurations for DNS, DHCP, and TFTP services on the Adonis server. The value is either *true* or *false* (by default, *false*).

> Resetting Adonis services will result in a service outage. This service outage will last until you have deployed services to the replacement system. Only reset Adonis services if you are replacing the Adonis server with a new appliance of a different type and/or reconfiguring the IPv4 or IPv6 addresses of the Services interface on the appliance. BlueCat recommends that you schedule a maintenance window before performing a reset of Adonis services.

Output/Response

Replaces the server using the existing server ID.

Related Methods

- *Add Server* on page 161
- *Server Generic Methods* on page 167
- *Get Servers Associated with a Deployment Role* on page 168
- *Get Published Interface* on page 168

Deploy Server

Deployment is the process through which the configuration created in Proteus becomes a running set of services on the Proteus-managed servers. Deployment takes account of the IP, DHCP, and DNS design determined during configuration. This is represented by a set of service configuration files deployed to the servers.

deployServer() deploys servers. When this method is invoked, the server is immediately deployed.

```
void ( long serverId )
```

Parameters

serverId—the object ID of the server to be deployed.

Related Methods

- *Add Server* on page 161.
- *Deploy Server Configuration* on page 165.
- *Quick Deployment* on page 166.
- *Server Generic Methods* on page 167.
- *Get Servers Associated with a Deployment Role* on page 168.
- *Get Published Interface* on page 168.

Deploy Server Configuration

This method allows you to deploy specific configuration(s) to a particular server.

deployServerConfig () deploys specific configurations to a particular server.

```
void deployServerConfig ( long serverID, String properties )
```

Parameters

serverId—the database object ID of the server that will immediately be deployed.

properties—a string containing property names. The property names available in the ObjectProperties are ***ObjectProperties.services***, and ***ObjectProperties.forceDNSFullDeployment***. Multiple options can be separated by a | (pipe) character. For example:

```
ObjectProperties.services=DNS|forceDNSFullDeployment=true
```

The values for properties are:

- **services**—the name of the valid service configuration that needs to be deployed. These are the valid values for the services: DNS, DHCP, DHCPv6, and TFTP.
- **forceDNSFullDeployment**—a boolean value. set to true to perform a full DNS deployment. Omit this parameter from the method call to perform a differential deployment.

Output/Response

Deploys the configuration(s) for a particular server.

Related Methods

- *Deploy Server* on page 164.
- *Deploy Server Services* on page 165.
- *Quick Deployment* on page 166.
- *Server Generic Methods* on page 167.
- *Get Servers Associated with a Deployment Role* on page 168.
- *Get Published Interface* on page 168.

Deploy Server Services

This method allows you to deploy specific service(s) to a particular server.

deployServerServices (serverId, String services) deploys specific services to a particular server.

```
void deployServerServices( long serverId, String properties )
```

Parameters

serverId—the object ID of the server for which deployment services need to be deployed.

services—the name of the valid service that need to be deployed. If the service is null, the behaviour is the same as calling the API method deployServer (serverId). The valid values are: ‘services=DNS’, ‘services=DHCP’, ‘services=DHCPv6’, and ‘services=TFTP’.

Output/Response

Deploys the service(s) for a particular server.

Related Methods

- *Deploy Server* on page 164.
- *Deploy Server Configuration* on page 165.
- *Quick Deployment* on page 166.
- *Server Generic Methods* on page 167.
- *Get Servers Associated with a Deployment Role* on page 168.
- *Get Published Interface* on page 168.

Quick Deployment

This method allows you to instantly deploy changes you made to DNS resource records. This function applies only to DNS resource records that you have changed and does not deploy any other data.

quickDeploy() deploys changes you made to DNS resource records instantly.

```
public void quickDeploy( long zoneId, String properties )
```

Parameters

zoneId—the object ID of the DNS zone for which deployment service needs to be deployed.

properties—a string containing the *services* option. It can *also be* ***null***.

  - **services**—the name of the valid service that need to be deployed. The *only* valid service name for quick deployment is ***DNS***. Any other service names will throw an error.

Output/Response

Instantly deploys changes to DNS resource records made since the last full deployment or quick deployment.

Related Methods

- *Deploy Server* on page 164
- *Deploy Server Configuration* on page 165.
- *Deploy Server Services* on page 165.
- *Deployment Status* on page 167.

Deployment Status

This method returns the status code for the deployment for a particular server.

getServerDeploymentStatus() returns the server's deployment status.

```
int getServerDeploymentStatus( long serverId, String properties )
```

Parameters

serverId—the object ID of the server whose deployment status needs to be checked.

properties—ignore this for now; the valid value is null.

Output/Response

Returns status code for deployment of a particular serve. These are the possible returning code values:

- EXECUTING = -1
- INITIALIZING = 0
- QUEUED = 1
- CANCELLED = 2
- FAILED = 3
- NOT_DEPLOYED = 4
- WARNING = 5
- INVALID = 6
- DONE = 7
- NO_RECENT_DEPLOYMENT = 8

Related Methods

- *Deploy Server* on page 164
- *Quick Deployment* on page 166
- *Server Generic Methods* on page 167
- *Get Servers Associated with a Deployment Role* on page 168
- *Get Published Interface* on page 168

Server Generic Methods

Servers use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Server* on page 161
- *Deploy Server* on page 164
- *Get Servers Associated with a Deployment Role* on page 168
- *Get Published Interface* on page 168

Get Published Interface

To get the published interface IP address for a server, use the getEntities() method with the **PublishedServerInterface** type to return the server properties. The **publiushedInterfaceAddress** property appears in the returned properties string.

Related Methods

- *Get Entities* on page 49
- *Add Server* on page 161
- *Deploy Server* on page 164
- *Server Generic Methods* on page 167
- *Get Servers Associated with a Deployment Role* on page 168

DNS and DHCP Deployment Roles

Deployment roles determine the general pattern of the deployment. A deployment role exists on a particular server interface (physical or published) specified with an IP address. Each server interface can have multiple DNS roles and one DHCP role with the most locally-specified server role taking precedence. The addition of a deployment role is allowed only if that role is possible under that server’s service capability profile as described in the *Proteus Administration Guide*.

Get Servers Associated with a Deployment Role

getServerForRole() returns a list of all servers for the specified deployment role.

```
APIEntity getServerForRole ( long roleId )
```

Parameters

roleId—the object ID for the deployment role whose servers are to be returned.

Output/Response

Returns an APIEntity object representing the servers associated with the specified deployment role.

Related Methods

- *Add Server* on page 161
- *Deploy Server* on page 164
- *Server Generic Methods* on page 167
- *Get Deployment Roles for DNS and IP Address Space Objects* on page 169
- *Get DHCP Deployment Role* on page 171
- *Get DNS Deployment Role* on page 173
- *Get DNS Deployment Role for View* on page 174

Get Server’s Associated Deployment Roles

getServerDeploymentRoles() returns a list of all deployment roles assigned to the server.

```
public APIDeploymentRole[] getServerDeploymentRoles ( long serverId )
```

Parameters

serverId—the object ID of the server with which deployment roles are associated.

Output/Response

Returns a list of all deployment roles associated with the server.

Related Methods

- *Get Servers Associated with a Deployment Role* on page 168

Get Deployment Roles for DNS and IP Address Space Objects

getDeploymentRoles() returns the DNS and DHCP deployment roles for a specified Proteus object. For DNS views and zones, **getDeploymentRoles()** returns DNS deployment roles. For IP address space objects, such as IPv4 blocks and networks, IPv6 blocks and networks, DHCP classes, and MAC pools, **getDeploymentRoles()** returns DNS and DHCP deployment roles.

```
APIDeploymentRole[] getDeploymentRoles ( long entityId )
```

Parameters

entityId—the object ID for a DNS view, DNS zone, IPv4 block or network, IPv6 block or network, DHCP class, or MAC pool.

Output/Response

Returns an array of APIDeploymentRole objects representing the deployment roles associated with the specified object. The properties string contains the following elements:

- **view**—for DNS deployment roles set for IP address space objects.
- **zoneTransServerInterface**—the server interface for zone transfers for the deployment role types of slave, stealth slave, forwarder and stub.
- **inherited**—returns *true* or *false* to indicate whether the deployment role was inherited or not.

> The Perl syntax for returning a value with the method has changed in Proteus version 3.1.
>
> For more information, see *Change to Perl Syntax in Proteus v3.1* on page 44.

Related Methods

- *Get Servers Associated with a Deployment Role* on page 168
- *Get DHCP Deployment Role* on page 171
- *Get DNS Deployment Role* on page 173
- *Get DNS Deployment Role for View* on page 174

Move Deployment Roles

moveDeploymentRoles() moves all DNS and DHCP deployment roles from a server to the specified interface of another server.

```
void moveDeploymentRoles ( long sourceServerId, long targetServerInterfaceId,
boolean moveDnsRoles, boolean moveDhcpRoles, String options )
```

> You ***CANNOT*** move deployment roles if the target server has deployment roles associated with it. You ***MUST*** remove all deployment roles assigned to the target server before moving the roles.

Parameters

sourceServerId—the object ID of the server that contains the roles.

targetServerInterfaceId—the object ID of the server interface of the server to which the roles are to be moved.

moveDnsRoles—if set to *true*, DNS roles will be moved to the target server interface.

moveDhcpRoles—if set to *true*, DHCP roles will be moved to the target server interface.

> Either the **moveDnsRoles** or **moveDhcpRoles** parameter must be set to *true*.

options—this parameter is reserved for future use.

Output/Response

None.

Related Methods

- *Get Servers Associated with a Deployment Role* on page 168
- *Get DHCP Deployment Role* on page 171
- *Get DNS Deployment Role* on page 173
- *Get DNS Deployment Role for View* on page 174

DHCP Deployment Roles

The DHCP server role can be set to either **master** or **none**. Roles set to **none** are not deployed. Roles can also be applied at many points throughout a configuration, with the most local roles taking precedence over those assigned to objects higher in the object hierarchy.

Add DHCP Deployment Role

addDHCPDeploymentRole() adds a DHCP deployment role to a specified object.

```
long addDHCPDeploymentRole( long entityId, long serverInterfaceId, String type,
String properties )
```

Parameters

entityId—the object ID for the object to which the deployment role is to be added.

serverInterfaceId—the object ID of the server interface to which the role is to be deployed.

type—the type of DHCP role to be added. The type must be one of those listed in *DHCP Deployment Role Types* on page 203.

properties—a string containing options including:

- **inherited**—either *true* of *false;* indicates whether or not the deployment role was inherited.

- **secondaryServerInterfaceId**—the object ID of the secondary server interface for a DHCP failover.

Output/Response

Returns the object ID for the new DHCP server role object.

Related Methods

- *Get DHCP Deployment Role* on page 171
- *Update DHCP Deployment Role* on page 171
- *Delete DHCP Deployment Role* on page 172

Get DHCP Deployment Role

getDHCPDeploymentRole() retrieves the DHCP deployment role assigned to a specified object.

```
APIDeploymentRole getDHCPDeploymentRole( long entityId, long serverInterfaceId )
```

Parameters

entityId—the object ID for the object to which the deployment role is assigned.

serverInterfaceId—the object ID of the server interface to which the role is assigned.

Output/Response

Returns the DHCP deployment role assigned to the specified object, or returns null if no role is defined. For information about the output properties, refer to *Property Options Reference* on page 255.

Related Methods

- *Add DHCP Deployment Role* on page 170
- *Update DHCP Deployment Role* on page 171
- *Delete DHCP Deployment Role* on page 172

Update DHCP Deployment Role

updateDHCPDeploymentRole() updates a DHCP deployment role.

```
void updateDHCPDeploymentRole( APIDeploymentRole role )
```

Parameters

role—the DHCP deployment role object to be updated.

Related Methods

- *Add DHCP Deployment Role* on page 170
- *Get DHCP Deployment Role* on page 171
- *Delete DHCP Deployment Role* on page 172

Delete DHCP Deployment Role

deleteDHCPDeploymentRole() deletes DHCP deployment roles.

```
void deleteDHCPDeploymentRole( long entityId, long serverInterfaceId )
```

Parameters

entityId—the object ID for the object from which the deployment role is to be deleted.

serverInterfaceId—the object ID of the server interface from which the deployment roles is to be deleted.

Output/Response

None.

Related Methods

- *Add DHCP Deployment Role* on page 170
- *Get DHCP Deployment Role* on page 171
- *Update DHCP Deployment Role* on page 171

DNS Deployment Roles

At a minimum, DNS roles must be applied at the view level in order for DNS deployment to occur. They can also be applied further into the DNS core if desired. For Reverse DNS, a DNS deployment role must be applied to either a block or a network in order to deploy the Reverse DNS settings for that object and its sub-objects. The following DNS server roles are available:

DNS Role	Description
None	This DNS role is not deployed. Use this option for DNS objects that exist, but should not be deployed.
Master	This role deploys details and options consistent with a DNS master. This role is also used on an Adonis 250 with the appropriate DNS options to create a caching-only DNS server.
Hidden Master	This role deploys details and options consistent with a DNS master. However, no name server records are created for the server, thus hiding it from DNS queries.
Slave	This role deploys details and options consistent with a DNS slave.
Stealth Slave	A stealth slave is a DNS slave server that does not have any name server records pointing to it. This is useful for testing purposes or for having a hot spare stand-by server. However, this is not a commonly used DNS role.
Forwarder	This role deploys details and options consistent with a DNS forwarder. You must use both the forwarding policy and forwarding options to make this role function properly.
Stub	A stub zone contains only the name server records for a domain. Proteus generates name server records automatically during deployment, so a zone deployed within a stub role will not contain any user-selected details or options.
Recursion	This role creates DNS caching servers. The options and root zone associated with this role are described in the *Proteus Administration Guide*.

Add DNS Deployment Role

addDNSDeploymentRole() adds a DNS deployment role to a specified object.

```
long addDNSDeploymentRole( long entityId, long serverInterfaceId, String type,
String properties )
```

Parameters

entityId—the object ID for the object to which the deployment role is to be added.

serverInterfaceId—the object ID of the server interface to which the role is to be added.

type—the type of DNS role to be added. The type must be one of those listed in *DNS Deployment Role Types* on page 212.

properties—adds object properties, including the **view** associated with this DNS deployment role and user-defined fields.

Output/Response

Returns the object ID for the new DNS server role object.

Related Methods

- *Get DNS Deployment Role* on page 173
- *Get DNS Deployment Role for View* on page 174
- *Update DNS Deployment Role* on page 174
- *Delete DNS Deployment Role* on page 175
- *Delete DNS Deployment Role for View* on page 175

Get DNS Deployment Role

getDNSDeploymentRole() retrieves a DNS deployment role from a specified object.

```
APIDeploymentRole getDNSDeploymentRole( long entityId, long serverInterfaceId )
```

Parameters

entityId—the object ID for the object to which the DNS deployment role is assigned.

serverInterfaceId—the object ID of the server interface to which the DNS deployment role is assigned.

Output/Response

Returns the DNS deployment roles from the specified object, or returns null if no role is defined. For information about the output properties, refer to *Property Options Reference* on page 255.

Related Methods

- *Add DNS Deployment Role* on page 173
- *Get DNS Deployment Role for View* on page 174
- *Update DNS Deployment Role* on page 174
- *Delete DNS Deployment Role* on page 175
- *Delete DNS Deployment Role for View* on page 175

Get DNS Deployment Role for View

getDNSDeploymentRoleForView() retrieves the DNS deployment role assigned to a view-level objects in the IP space for ARPA zones.

```
APIDeploymentRole getDNSDeploymentRoleForView( long entityId, long
serverInterfaceId, long viewId )
```

Parameters

entityId—the object ID for the object to which the deployment role is assigned.

serverInterfaceId—the object ID of the server interface to which the role is assigned.

viewId—the view in which the DNS deployment role is assigned.

Output/Response

Returns the requested APIDeploymentRole object. For information about the output properties, refer to *Property Options Reference* on page 255.

Related Methods

- *Add DNS Deployment Role* on page 173
- *Get DNS Deployment Role* on page 173
- *Update DNS Deployment Role* on page 174
- *Delete DNS Deployment Role* on page 175
- *Delete DNS Deployment Role for View* on page 175

Update DNS Deployment Role

updateDNSDeploymentRole() updates a specified DNS deployment role.

```
void updateDNSDeploymentRole( APIDeploymentRole role )
```

Parameters

role—the DNS deployment role object to be updated.

Output/Response

None.

Related Methods

- *Add DNS Deployment Role* on page 173
- *Get DNS Deployment Role* on page 173
- *Get DNS Deployment Role for View* on page 174
- *Delete DNS Deployment Role* on page 175
- *Delete DNS Deployment Role for View* on page 175

Delete DNS Deployment Role

deleteDNSDeploymentRole() delete a specified DNS deployment roles.

```
void deleteDNSDeploymentRole( long entityId, long serverInterfaceId )
```

Parameters

entityId—the object ID for the object from which this DNS deployment role is to be deleted.

serverInterfaceId—the object ID of the server interface to which the DNS deployment role is assigned.

Output/Response

None.

Related Methods

- *Add DNS Deployment Role* on page 173
- *Get DNS Deployment Role* on page 173
- *Get DNS Deployment Role for View* on page 174
- *Update DNS Deployment Role* on page 174
- *Delete DNS Deployment Role for View* on page 175

Delete DNS Deployment Role for View

deleteDNSDeploymentRoleForView() delete the DNS deployment role assigned to view-level objects in the IP space for ARPA zones.

```
void deleteDNSDeploymentRoleForView( long entityId, long serverInterfaceId, long
viewId )
```

Parameters

entityId—the object ID for the IPv4 network object from which the deployment role is to be deleted.

serverInterfaceId—the object ID of the server interface to which the DNS deployment role is assigned.

viewId—the view from which the DNS deployment role is to be deleted.

Output/Response

None.

Related Methods

- *Add DNS Deployment Role* on page 173
- *Get DNS Deployment Role* on page 173
- *Get DNS Deployment Role for View* on page 174
- *Update DNS Deployment Role* on page 174
- *Delete DNS Deployment Role* on page 175

TFTP Deployment Roles

TFTP deployment roles are used to assign TFTP services to DHCP servers.

Add TFTP Deployment Role

addTFTPDeploymentRole() adds a TFTP deployment role to a specified object.

```
long addTFTPDeploymentRole( long entityId, long serverId, String properties )
```

Parameters

entityId—the object ID for the object to which the TFTP deployment role is to be added.

serverId—the object ID of the server interface to which the TFTP deployment role is to be added.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new TFTP deployment role object.

Related Methods

- *Update TFTP Deployment Role* on page 176
- *TFTP Deployment Role Generic Methods* on page 176

Update TFTP Deployment Role

TFTP deployment roles cannot be updated.

Related Methods

- *Add TFTP Deployment Role* on page 176
- *TFTP Deployment Role Generic Methods* on page 176

TFTP Deployment Role Generic Methods

TFTP deployment roles use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add TFTP Deployment Role* on page 176
- *Update TFTP Deployment Role* on page 176

Crossover High Availability (XHA)

Adonis Crossover High Availability (XHA) provides disaster recovery through the use of redundant appliances: XHA makes two Adonis appliances function as a single appliance. If one of the appliances fails for any reason, the other takes its place and continues providing services. The pair appears as a single server for DNS queries because both servers share an IP address. Each server in the pair has its own IP addresses for management through Proteus. For details about XHA, please refer to *Proteus Administration Guide version 4.0*.

Requirements for creating an XHA pair

Before you create an XHA pair in Proteus, you must have the following requirements in place:

- You must have at least two connected and managed Adonis servers in the configuration. The servers must be of the same profile, such as two Adonis 1750 or Adonis 1000 units. For instructions on adding servers, see ***Working with Servers*** on page 599.
- In order to create an XHA pair with the Active node on which the dedicated management interface enabled, the dedicated management interface on the Passive node must be enabled.
- The Active and Passive nodes must be on the same network.
- The servers for the XHA pair must *not* be associated with a deployment schedule. For information on viewing the servers in a deployment schedule, see ***Setting Scheduled Deployment*** on page 618.
- The server intended for the passive role must not be associated with a deployment role. For instructions on how to view the deployment roles assigned to a server, see ***Viewing Deployment Roles*** on page 599.
- To avoid split-brain scenarios (where both servers are active or passive at the same time), the use of XHA Backbone Communication is *mandatory*.

> If you are currently using the XHA/eth1 ports for another purpose, you can reset and then reconfigure them for XHA communication, but you *cannot* use the eth1 ports for XHA communication *and* for their previous purpose.
>
> If you are upgrading from a previous version of Adonis, you *must* delete each eth1 port to reset it. This is because previous versions did not support eth1, and it is not reset automatically. For more information, refer to the *Adonis Administration Guide*.

Creating an XHA

With XHA prerequisites are met, you can create an XHA pair.

> You cannot configure interface and network settings of Adonis appliances that are part of a functioning XHA pair. Please configure interface and network settings before creating a XHA pair.

Create XHA

createXHAPair() creates an XHA pair.

```
public long createXHAPair( long configurationId, long activeServerId, long
passiveServerId, String activeServerNewIPv4Address, String properties )
```

Parameters

configurationId—the object ID of the configuration in which the XHA servers are located.

activeServerId—the object ID of the active Adonis server.

passiveServerId—the object ID of the passive Adonis server.

activeServerNewIPv4Address—the new IPv4 address for the active server.

> This is the physical interface of the active server used during creation of the pair. The original IP address of the active server is assigned to the virtual interface.

properties—a string containing the following options:

- **activeServerPassword**—the deployment password for the active server (by default, **bluecat**).
- **passiveServerPassword**—the deployment password for the passive server (by default, **bluecat**).
- **pingAddress**—an IPv4 address that is accessible to both active and passive servers in the XHA pair.
- **ip6Address**—an optional IPv6 address for the XHA pair.
- **newManagementAddress**—the new IPv4 address for the Management interface for the active server (*only* for **Adonis servers with dedicated management** *enabled*).
- **backboneActiveServerIPv4Address**—the IPv4 address of the XHA interface for the active server (eth1).
- **backboneActiveServerIPv4Netmask**—the IPv4 netmask of the XHA interface for the active server (eth1).
- **backbonePassiveServerIPv4Address**—the IPv4 address of the XHA interface for the passive server (eth1).
- **backbonePassiveServerIPv4Netmask**—the IPv4 netmask of the XHA interface for the passive server (eth1).
- **activeServerIPv4AddressForNAT**—the inside virtual IPv4 address for the active server.
- **passiveServerIPv4AddressForNAT**—the inside virtual IPv4 address for the passive server.
- **activeServerNewIPv4AddressForNAT**—the inside physical IPv4 address for the active server.

Output/Response

Returns the object ID for the XHA pair created.

Related Methods

- *Edit XHA* on page 178.
- *Break XHA* on page 180.
- *Failover XHA* on page 181.

Edit XHA

editXHAPair() updates the XHA pair created.

```
public void editXHAPair( long xHAServerId, String name, String properties )
```

Parameters

xHAServerId—the object ID of the XHA server.

name—the name of the XHA server being updated.

properties—a string containing the following options:

- **backboneActiveServerIPv4Address**—the IPv4 address of the XHA interface for the active server (eth1).
- **backboneActiveServerIPv4Netmask**—the IPv4 netmask of the XHA interface for the active server (eth1).
- **backbonePassiveServerIPv4Address**—the IPv4 address of the XHA interface for the passive server (eth1).
- **backbonePassiveServerIPv4Netmask**—the IPv4 netmask of the XHA interface for the passive server (eth1).
- **overrideDHCPValidation**—either *true* or *false*; indicates whether or not the deployment validation settings set at the configuration level is inherited.
- **checkDHCPConfigurationDeployment**—either *true* or *false*; checks the syntax of the **dhcpd.conf** file and validate data deployed from Proteus.
- **overrideDNSValidation**—either *true* or *false*; indicates whether or not the deployment validation settings set at the configuration level is inherited.
- **checkDNSConfigurationDeployment**—either *true* or *false*; checks the syntax of the **named.conf** file and validate data deployed from Proteus.
- **checkDNSZonesDeployment**—either *true* or *false*; checks the syntax of each DNS zone file and validate data deployed from Proteus.
- **postLoadZoneIntegrityValidationDNSDeploy**—checks the syntax based on the mode selected. The available modes are as follows:
  - **Full**—checks for the following conditions:
    - If MX records refer to A or AAAA records, for both in-zone and out-of-zone hostnames.
    - If SRV records refer to A or AAAA records, for both in-zone and out-of-zone hostnames.
    - If Delegation NS records refer to A or AAAA records, for both in-zone and out-of-zone hostnames
    - If glue address records in the zone match those specified by the child.
  - **Local**—checks for the following conditions:
    - If MX records refer to A or AAAA records, for in-zone hostnames.
    - If SRV records refer to A or AAAA records, for in-zone hostnames.
    - If Delegation NS records refer to an A or AAAA record, for in-zone hostnames.
    - If glue address records in the zone match those specified by the child.
  - **Full-sibling**—performs the same checks as in **Full** mode but does not check the glue records.
  - **Local-sibling**—performs the same checks as in **Local** mode but does not check the glue records.
  - **None**—disables all post-load zone integrity checks.
- **checkNamesValidationModeDNSDeploy**—checks names. Specify *Ignore*, *Warn* or *Fail* to determine how Proteus handles conditions found by this check.

- **checkIfMXRecordsAreIPsDNSDeploy**—checks if MX records point to an IP address rather than an A or AAAA record. Specify *Ignore*, *Warn* or *Fail* to determine how Proteus handles conditions found by this check.
- **checkIfMXRecordsPointToCNAMEsDNSDeploy**—checks if MX records point to a CNAME record rather than an A or AAAA record. Specify *Ignore*, *Warn* or *Fail* to determine how Proteus handles conditions found by this check.
- **checkIfNSRecordsAreIPsDNSDeploy**—checks if NS record point to an IP address rather than an A or AAAA record. Specify *Ignore*, *Warn* or *Fail* to determine how Proteus handles conditions found by this check.
- **checkIfSRVRecordsPointToCNAMEsDNSDeploy**—checks if SRV record point to a CNAME record rather than an A or AAAA record. Specify *Ignore*, *Warn* or *Fail* to determine how Proteus handles conditions found by this check.
- **checkForNonTerminalWildcardsDNSDeploy**—checks for wildcards in zone names that do not appear as the last segment of a zone name. Specify *Ignore* or *Warn* to determine how Proteus handles conditions found by this check.

Output/Response

None.

Related Methods

- *Create XHA* on page 177.
- *Break XHA* on page 180.
- *Failover XHA* on page 181.

Breaking an XHA

Breaking an XHA pair returns each server to its original stand-alone state. The server that held the active role remains connected to Proteus while the server that held the passive role is disconnected and has HA-NODE2 appended to its name. Each server is re-assigned its original IP address.

Break XHA

breakXHAPair() breaks an XHA pair and returns each server to its original stand-alone state.

```
public void breakXHAPair( long xHAServerId, boolean breakInProteusOnly )
```

Parameters

- **xHAServerId**—the object ID of the XHA server.
- **breakInProteusOnly**— either *true* or *false*; determines whether or not the XHA pair breaks in Proteus interface only. This argument breaks the XHA pair in Proteus even if the XHA settings are not removed on the actual servers.

Output/Response

Breaks an XHA pair.

Related Methods

- *Create XHA* on page 177.
- *Edit XHA* on page 178.
- *Failover XHA* on page 181.

XHA Failover

Under normal operation, XHA automatically fails over in the event of a hardware, network or service failure related to the Active node. However, you can perform a manual XHA failover for maintenance or verification purposes.

Failover XHA

failoverXHA() performs a manual XHA failover.

```
public void failoverXHA( long xHAServerId )
```

Parameters

- **xHAServerId**—the object ID of the XHA server.

Output/Response

Performs a manual XHA failover.

Related Methods

- *Create XHA* on page 177.
- *Edit XHA* on page 178.
- *Break XHA* on page 180.

Proteus Objects

The other objects managed by the Proteus API are native Proteus objects. These objects are part of the Proteus server rather than the services it manages. For more information about Proteus object types, refer to *Proteus Object Hierarchy* on page 24, and to the *Proteus Administration Guide*.

Configurations

Proteus provides a separation between the logical design of a network and its implementation on the actual network hardware. An administrator designs a network as a configuration. The configuration uses global elements such as users and groups, and local elements such as DNS and IP designs. When combined, these create a complete logical network design. During this process or afterward, servers (defined for each configuration) can be associated with different parts of the configuration using the various deployment roles available within the configuration.

Add Configuration

addEntity() is a generic method for adding configurations, DNS zones, and DNS resource records.

```
long addEntity( long parentId, APIEntity entity )
```

Parameters

parentId—for configurations, always set the **parentId** value to 0 (zero), which is the root element.

entity—the configuration object, including its name, sharedNetwork, and user-defined fields.

Output/Response

Returns the object ID for the new configuration.

Related Methods

- *Update Configuration* on page 182
- *Configuration Generic Methods* on page 182

Update Configuration

A configuration’s **name** and **sharedNetwork** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Configuration* on page 182
- *Configuration Generic Methods* on page 182

Configuration Generic Methods

Configurations use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Configuration* on page 182
- *Update Configuration* on page 182

Groups and Users

Proteus is designed to accommodate environments that require the ability to host multiple concurrent users who could be located in different regions. Proteus can also be run by a single administrator.

Add Group

addUserGroup() adds user groups.

```
long addUserGroup( String name, String properties )
```

Parameters

name—the name of the user group.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new Proteus user group.

To add users to a user group, use the linkEntities() method, specifying the user ID and the group ID. It does not matter in which order you specify the user ID and the group ID. Either of the following will add a user to a user group:

```
void linkEntities ( long user_id, long group_id, String properties )
```

or

```
void linkEntities ( long group_id, long user_id, String properties )
```

Related Methods

- *Update Group* on page 183
- *Group Generic Methods* on page 184
- *Add User* on page 184
- *Update User* on page 185
- *User Generic Methods* on page 185

Update Group

A user group’s **name** property can be updated using the generic update() method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Group* on page 183
- *Group Generic Methods* on page 184
- *Add User* on page 184
- *Update User* on page 185
- *User Generic Methods* on page 185

Group Generic Methods

User groups use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Group* on page 183
- *Update Group* on page 183
- *Add User* on page 184
- *Update User* on page 185
- *User Generic Methods* on page 185

Add User

addUser() adds Proteus users.

```
long addUser( String username, String password, String properties )
```

Parameters

username—the name of the user.

password—the Proteus password for the user. The password must be set even if the authenticator property option is defined.

properties—adds object properties, including **authenticator**, **securityPrivilege**, **historyPrivilege**, **email**, **phoneNumber**, **user type**, **user access type**, and user-defined fields. Multiple property values can be separated by a | (pipe) character. For example: **my $properties = “email=$email|phoneNumber=$tel|authenticator=1368969|”**

> You must add a | (pipe) character at the end in the properties string.

- **authenticator—**the object ID of the external authenticator defined in Proteus.
- **securityPrivilege—**a security privilege type for Non-Administrator users with GUI, API, or GUI and API access. **NO ACCESS** is the default value.
- **historyPrivilege—**a history privilege type for Non-Administrator users with GUI, or GUI and API access. **HIDE** is the default value.
- **email—**the email address for the user. This is required.
- **phoneNumber—**the phone number for the user.
- **userType**—**ADMIN** or **REGULAR** (non-administrator—**REGULAR** is the default value).

- **userAccessType**—**API**, **GUI**, or **GUI and API**. This is required.

UserType	UserAccessType	Privileges
ADMIN	n/a	History and Security privileges are set automatically.
REGULAR	GUI	History and Security privileges are set to a user-specific value.
REGULAR	API	Security privilege is set to a user-specific value.
REGULAR	GUI and API	History and Security privileges are set to a user-specific value.

Output/Response

Returns the object ID for the new Proteus user.

To add users to a user group, use the **linkEntities()** method, specifying the user ID and the group ID. It does not matter in which order you specify the user ID and the group ID. Either of the following adds a user to a user group:

```
void linkEntities ( long user_id, long group_id, String properties )
```

or

```
void linkEntities ( long group_id, long user_id, String properties )
```

Related Methods

- *Add Group* on page 183
- *Update Group* on page 183
- *Group Generic Methods* on page 184
- *Update User* on page 185
- *User Generic Methods* on page 185

Update User

A Proteus user’s **securityPrivilege** and **historyPrivilege** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Group* on page 183
- *Update Group* on page 183
- *Group Generic Methods* on page 184
- *Add User* on page 184
- *User Generic Methods* on page 185

User Generic Methods

Proteus user objects use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Group* on page 183
- *Update Group* on page 183
- *Group Generic Methods* on page 184
- *Add User* on page 184
- *Update User* on page 185

Authenticators

Proteus includes a fully featured authentication subsystem. The Proteus administrator uses this system to securely log in to Proteus and administer the system when it is being configured. Proteus also supports mixed-mode authentication through RADIUS, LDAP, Microsoft Active Directory, or Kerberos. Support for RSA Secure ID is accomplished through the RADIUS authentication module.

The necessary settings must be in place before Proteus can pass authentication information to these remote systems. Also, the authentication method must be associated with a Proteus user. This is accomplished by creating an *authenticator* and assigning it to a user. Authenticators are system objects that represent a connection to an external authentication system. The use of that system's native safeguards applies for communications between it and Proteus. Proteus acts as a proxy client for the authentication system, validating the identity of a Proteus user without managing or validating the user's password or credentials.

After the users are authenticated against the external system, they are considered to be validated in Proteus until they close their sessions, or until it is invalidated by a session time-out. Authentication is not a substitute for Proteus user management. Being a Proteus user is still a requirement to log in to the system. Authenticators move the responsibility of validating credentials to another system.

Many organizations centralize control over internal digital identities. In such scenarios, suspending or revoking credentials and password management are tightly controlled. Proteus is designed to be deployed within all major network authentication frameworks. This lets Proteus assist with enforcing network standards, rather than requiring a circumvention.

A user may be assigned several authenticators. These are used in order of primary-secondary.

Update Authenticator

An authenticator's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Authenticator Generic Methods* on page 186

Authenticator Generic Methods

Authenticators use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Update Authenticator* on page 186

Access Rights

Proteus is arranged as a hierarchy of objects with the server itself at the highest level. This offers a security and privilege system that is both simple and adaptive because different subsections of the server do not have separate systems. An object may be an entire configuration, a single subnet, a tag, etc. However, this means that existing security schemes must be mapped to the Proteus architecture. This section outlines the concepts necessary to perform this mapping.

The Proteus server is a hierarchical structure with a configuration, user, group, or Object Tag group as the root element of the tree. Because everything in this hierarchy is an object, a user can have a different set of rights for each object within the system. However, permissions are more likely granted for a certain level as well as for everything below that level (with certain exceptions).

Access rights within Proteus can be assigned to both users and groups. Furthermore, multiple rights can exist for the same object. Three simple rules dictate a user's access rights for any object:

- Administrators always have full control over any system object.
- Local rights take precedence over rights assigned higher in the object hierarchy.
- In the case of conflicting object rights, the most permissive right always takes precedence.

These three rules cover all of the possible cases for access rights. For more information about Proteus user rights and security, refer to the *Proteus Administration Guide*.

Add Access Right

addAccessRight() adds access rights to a specified object.

```
long addAccessRight( long entityId, long userId, String value, String overrides,
String properties )
```

Parameters

entityId—the object ID of the entity to which the access right is being added. Set this to 0 if the access right is being added to the root level (default access rights).

userId—the object ID of the user to whom this access right applies.

value—the value of the access right being added. Valid values for this parameter are listed in *Access Right Values* on page 203.

overrides—a list of type-specific overrides in the following format:

```
"objectType=accessValue|objectType=accessValue"
```

properties—a string including the following options:

- **workflowLevel**—valid values for this option are as follows:
  - **None**—changes made by the user or group take effect immediately.
  - **Recommend**—changes made by the user or group are saved as change requests and must be reviewed and approved before they take effect.
  - **Approve**—changes made by the user or group take effect immediately and the user or group can approve change requests from other users or groups.
- **deploymentAllowed**—either *true* or *false*; to indicate whether or not the user or group can perform a full deployment of data from the configuration to a managed server.

- **quickDeploymentAllowed**—either *true* or *false*; to indicate whether or not the user or group can instantly deploy changed DNS resource records.

> - All these Properties are *optional*.
> - The **deploymentAllowed** property is applicable only for configuration, server or root with *Full* access.
> - The **workflowLevel** property is applicable only for *Change*, *Add*, or *Full* access rights.

Output/Response

Returns the object ID for the new access right.

Related Methods

- *Get Access Right* on page 188
- *Get Access Rights for Entity* on page 189
- *Get Access Rights for User* on page 189
- *Update Access Rights* on page 190
- *Delete Access Rights* on page 191

Get Access Right

getAccessRight() retrieves an access right for a specified object.

```
getAccessRight( long entityId, long userId )
```

> If the full access right is set on the parent object, the **getAccessRight()** method for the child object will retrieve the full access right even if there is a *hide* override set for the child object type. It is the caller's responsibility to evaluate the returned APIAccessRight's value and overrides to determine the effective access level for the child object.

Parameters

entityId—the object ID of the entity to which the access right is assigned.

userId—the object ID of the user to whom the access right is applied.

Output/Response

Returns the access right for the specified object.

Related Methods

- *Add Access Right* on page 187
- *Get Access Rights for Entity* on page 189
- *Get Access Rights for User* on page 189
- *Update Access Rights* on page 190
- *Delete Access Rights* on page 191

Get Access Rights for Entity

getAccessRightsForEntity() retrieves access rights for entities.

```
[] getAccessRightsForEntity( long entityId, int start, int count )
```

Parameters

entityId—the object ID of the entity whose access rights are returned.

start—indicates where in the list of child access right objects to start returning objects. The list begins at an index of 0.

count—the maximum number of access right child objects to return.

Output/Response

Returns an array of access right objects.

> The Perl syntax for returning a value with the method has changed in Proteus version 3.1. For more information, see *Change to Perl Syntax in Proteus v3.1* on page 44.

Related Methods

- *Add Access Right* on page 187
- *Get Access Right* on page 188
- *Get Access Rights for User* on page 189
- *Update Access Rights* on page 190
- *Delete Access Rights* on page 191

Get Access Rights for User

getAccessRightsForUser() retrieves access rights for a specified user.

```
[] getAccessRightsForUser( long userId, int start, int count )
```

Parameters

userId—the object ID of the user whose access rights are returned.

start—indicates where in the list of child access right objects to start returning objects. The list begins at an index of 0.

count—the maximum number of access right child objects to return.

Output/Response

Returns an array of access right objects.

> The Perl syntax for returning a value with the method has changed in Proteus version 3.1. For more information, see *Change to Perl Syntax in Proteus v3.1* on page 44.

Related Methods

- *Add Access Right* on page 187
- *Get Access Right* on page 188
- *Get Access Rights for Entity* on page 189
- *Update Access Rights* on page 190
- *Delete Access Rights* on page 191

Update Access Rights

updateAccessRight() updates access rights for a specified object.

```
void updateAccessRight( long entityId, long userId, String value, String overrides,
String properties )
```

Parameters

entityId—the object ID of the entity to which the access right is assigned.

userId—the object ID of the user to whom the access right is assigned. This value is not mutable.

value—the new value for the access right. Valid entries are listed in *Access Right Values* on page 203.

overrides—a list of potentially modified type-specific overrides in the following format:

```
"objectType=accessValue|objectType=accessValue"
```

properties—a string including the following options:

- **workflowLevel**—valid values for this option are as follows:
  - **None**—changes made by the user or group take effect immediately.
  - **Recommend**—changes made by the user or group are saved as change requests and must be reviewed and approved before they take effect.
  - **Approve**—changes made by the user or group take effect immediately and the user or group can approve change requests from other users or groups.
- **deploymentAllowed**—either *true* or *false*; to indicate whether or not the user or group can perform a full deployment of data from the configuration to a managed server.
- **quickDeploymentAllowed**—either *true* or *false*; to indicate whether or not the user or group can instantly deploy changed DNS resource records.perform a full deployment of data from the configuration to a managed server.

- All these Properties are *optional*.
- The **deploymentAllowed** property is applicable only for configuration, server or root with *Full* access.
- The **workflowLevel** property is applicable only for *Change*, *Add*, or *Full* access rights.

Output/Response

None.

Related Methods

- *Add Access Right* on page 187
- *Get Access Right* on page 188
- *Get Access Rights for Entity* on page 189
- *Get Access Rights for User* on page 189
- *Delete Access Rights* on page 191

Delete Access Rights

deleteAccessRight() deletes an access right for a specified object.

```
void deleteAccessRight( long entityId, long userId )
```

Parameters

entityId—the object ID of the entity to which the access right is assigned.

userId—the object ID of the user to whom this access right is applied.

Output/Response

None.

Related Methods

- *Add Access Right* on page 187
- *Get Access Right* on page 188
- *Get Access Rights for Entity* on page 189
- *Get Access Rights for User* on page 189
- *Update Access Rights* on page 190

Object Tag Groups

Object tags can change the entire scheme by which users navigate Proteus. By tagging various objects, companies can assign privileges based on existing business authority regimes, and limit access to system objects using familiar business models. Proteus object tags are arranged in a hierarchical tree structure. This should accommodate most element-based XML designs, because any realistic number of elements are supported at each level of the hierarchy below a top-level or root tag known as a tag group. The system supports more than one hundred levels of tags, so it can accommodate complex nested structures.

The object tagging structure comprises large sets of XML elements that are without attributes. They begin with a root element and all subsequent tags belong to branches below the tag group in a series of parent-child relationships. The tag groups cannot be applied to objects.

Add Object Tag Group

addTagGroup() adds object tag groups.

```
long addTagGroup( String name, String properties )
```

Parameters

name—the name of the tag group.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new tag group.

Related Methods

- *Update Object Tag Group* on page 192
- *Object Tag Group Generic Methods* on page 192

Update Object Tag Group

A tag group's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Object Tag Group* on page 191
- *Object Tag Group Generic Methods* on page 192

Object Tag Group Generic Methods

This object implements the generic **get()** and **delete()** methods for entities.

Related Methods

- *Add Object Tag Group* on page 191
- *Update Object Tag Group* on page 192

Object Tags

Object tag groups use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Add Object Tag

addTag() adds object tags.

```
long addTag( long parentid, String name, String properties )
```

Parameters

parentid—the object ID of the parent for this object tag. The parent is either an object tag or an object tag group.

name—the name of the object tag.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID for the new object tag.

Related Methods

- *Assign Object Tag* on page 193
- *Remove Object Tag* on page 193
- *Update Object Tag* on page 194
- *Object Tag Generic Methods* on page 194

Assign Object Tag

> This method is deprecated. Using this method now returns an error message. Use the **linkEntities()** method instead. For more information, see *Link Entities* on page 57.

The following method is used to assign object tags to objects through the Proteus API:

```
void tagEntity( long entityId, long tagId )
```

Parameters

entityId—the object ID of the entity to which the tag is assigned.

tagId—the object ID of the tag that is assigned.

Output/Response

None.

Related Methods

- *Add Object Tag* on page 192
- *Remove Object Tag* on page 193
- *Update Object Tag* on page 194
- *Object Tag Generic Methods* on page 194

Remove Object Tag

> This method is deprecated. Using this method now returns an error message. Use the **unlinkEntities()** method instead. For more information, see *Unlink Entities* on page 58.

untagEntity() removes object tags from specified objects.

```
void untagEntity( long entityId, long tagId )
```

Parameters

entityId—the object ID of the entity from which the tag is to be removed.

tagId—the object ID of the tag to be removed.

Output/Response

None.

Related Methods

- *Add Object Tag* on page 192
- *Assign Object Tag* on page 193
- *Update Object Tag* on page 194
- *Object Tag Generic Methods* on page 194

Update Object Tag

An object tag's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add Object Tag* on page 192
- *Assign Object Tag* on page 193
- *Remove Object Tag* on page 193
- *Object Tag Generic Methods* on page 194

Object Tag Generic Methods

Object tags use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add Object Tag* on page 192
- *Assign Object Tag* on page 193
- *Remove Object Tag* on page 193
- *Update Object Tag* on page 194

Database Management

Configure Replication

configureReplication() enables database replication on a remote system in order to automate the setup of replication between two or three Proteus systems.

```
void configureReplication ( String standbyServer, boolean compressReplication, long
replicationQueueThreshold, long replicationBreakThreshold, String properties )
```

This API method must be run against the Proteus system that will be primary.

Parameters

- **standbyServer**—the IP address of the standby server.

> The standby server must be accessible from the primary server and must have database access from the primary server. To enable database access, refer to the *Configuring database replication* section in the ***Proteus Administration Guide***.

- **compressReplication**—the boolean value. Set to true to compress the database replication files.

> Compressing database replication files is a resource-intensive process that might affect system performance. Use caution when performing this action.

- **replicationQueueThreshold**—a value to specify the threshold size of the replication directory in megabytes (MB). The valid values are in the range of *16* to *99999999*.
- **replicationBreakThreshold**—a value to specify the threshold size of the replication break in gigabytes (GB). The valid values are in the range of *5* to *30*. This value multiplied by 1024 must be greater than the value of replicationQueueThreshold.
- **properties**—a string containing the following property:
  - **secondStandbyServer**—the IP address of the second standby server. This is optional.

> Any property string other than *secondStandbyServer* option will be ignored.

Output/Response

None.

Devices

Add Device

addDevice() adds a device to a configuration. A *device* is an actual physical component, such as a router or printer or other equipment to which one or more IP addresses are assigned. Devices are organized by device types and device sub-types. A *device type* is a general category of devices; a *device sub-type* is a more specific category of devices. For example, a general device type might be *Printers*. More specific device sub-types might include *Laser Printers*, *Plotters*, and *Imagesetters*.

```
long addDevice ( long configurationId, String name, long deviceTypeId, long
deviceSubtypeId, String ip4Addresses, String ip6Addresses, String properties )
```

Parameters

configurationId—the object ID of the configuration in which the device is to be located.

name—the descriptive name of the device.

deviceTypeId—the object ID of the device type with which the device is associated.

deviceSubtypeId—the object ID of the device sub-type with which the device is associated.

ip4Addresses—one or more IPv4 addresses to which the device is assigned. Specify multiple addresses in a comma-delimited list.

ip6Addresses—one or more IPv6 addresses to which the device is assigned. Specify multiple addresses in a comma-delimited list.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID of the new device.

Related Methods

- *Add Device Type* on page 196
- *Add Device Subtype* on page 196

Add Device Type

addDeviceType() adds a device type to Proteus. Use device types and device sub-types to categorize and organize devices on the network.

```
long addDeviceType ( String name, String properties )
```

Parameters

name—the descriptive name for the device type.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID of the new device type.

Related Methods

- *Add Device* on page 195
- *Add Device Subtype* on page 196

Add Device Subtype

addDeviceSubtype() adds a device sub-type to Proteus. Use device types and device sub-types to categorize and organize devices on the network.

```
long addDeviceSubtype ( long parentId, String name, String properties )
```

Parameters

parentId—the object ID of the parent device type object.

name—the descriptive name for the device sub-type.

properties—adds object properties, including user-defined fields.

Output/Response

Returns the object ID of the new device sub-type.

Related Methods

- *Add Device* on page 195
- *Add Device Type* on page 196

MAC Pools

Media Access Control (MAC) pools are used to group MAC addresses for functionality such as Network Access Control (NAC). Each MAC pool can be linked to multiple MAC addresses, and each MAC address can be linked to multiple IP addresses of different networks. However, each MAC address can belong to only one MAC pool, and each IP address can belong to only one MAC address. The MAC pools include one default global 'Deny' pool object that the user cannot delete. All pools created by the user can be deleted. A MAC pool contains a name (required), and optional links to MAC addresses.

Get MAC Addresses in Pool

This method is deprecated. Using this method now returns an error message. Use the **getLinkedEntities()** method instead. For more information, see *Get Linked Entities* on page 56.

getMACAddressesInPool() returns a list of the MAC address objects within a specified MAC pool.

```
APIEntity[] getMACAddressesInPool(long macPoolId, int start, int count )
```

Parameters

macPoolId—the object ID for the MAC pool.

start—indicates where in the list of children to start returning objects. The list begins at an index of 0.

count—this is the maximum number of child objects to return.

Output/Response

Returns an array of MAC address objects.

Related Methods

- *Update MAC Pool* on page 197
- *MAC Pool Generic Methods* on page 198

Update MAC Pool

A MAC pool's **name** property can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Get MAC Addresses in Pool* on page 197
- *MAC Pool Generic Methods* on page 198

MAC Pool Generic Methods

MAC pools use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Get MAC Addresses in Pool* on page 197
- *Update MAC Pool* on page 197

MAC Addresses

MAC address objects are used to reference the MAC addresses of endpoints.

Add MAC Address

addMACAddress() adds MAC addresses.

```
long addMACAddress( long configurationId, String macAddress, String properties )
```

Parameters

configurationId—the object ID of the parent configuration in which the MAC address resides.

macAddress—the MAC address in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

properties—adds object properties, including the **name**, MAC Pool ID (**macPool**), and user-defined fields.

> To assign a MAC address to the DENY MAC pool, use the **denyMACAddress()** method. For more information, see *Deny MAC Address* on page 199.

Output/Response

Returns the object ID for the new MAC address.

Related Methods

- *Associate MAC Address* on page 198
- *Deny MAC Address* on page 199
- *Is Address Allocated?* on page 199
- *Update MAC Address* on page 200
- *MAC Address Generic Methods* on page 200

Associate MAC Address

associateMACAddressWithPool() associates a MAC address with a MAC pool.

```
void associateMACAddressWithPool( long configurationId, String macAddress, long
macPool )
```

Parameters

configurationId—the object ID of the parent configuration in which the MAC address resides.

macAddress—the MAC address in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

macPool—the object ID of the MAC pool with which this MAC address is associated.

> To assign a MAC address to the DENY MAC pool, use the **denyMACAddress()** method. For more information, see *Deny MAC Address* on page 199.

Related Methods

- *Add MAC Address* on page 198
- *Deny MAC Address* on page 199
- *Is Address Allocated?* on page 199
- *Update MAC Address* on page 200
- *MAC Address Generic Methods* on page 200

Deny MAC Address

denyMACAddress() denies MAC addresses.

```
void denyMACAddress( long configurationId, String macAddress )
```

Parameters

configurationId—the object ID of the parent configuration in which the MAC address resides.

macAddress—the MAC address in the format *nnnnnnnnnnnn*, *nn-nn-nn-nn-nn-nn* or *nn:nn:nn:nn:nn:nn*, where *nn* is a hexadecimal value.

Related Methods

- *Add MAC Address* on page 198
- *Associate MAC Address* on page 198
- *Is Address Allocated?* on page 199
- *Update MAC Address* on page 200
- *MAC Address Generic Methods* on page 200

Is Address Allocated?

isAddressAllocated() queries a MAC address to determine if the address has been allocated to an IP address.

```
boolean isAddressAllocated( long configurationId, String ipAddress, String
macAddress )
```

Parameters

configurationId—the object ID of the parent configuration in which the MAC address resides.

ipAddress—the IPv4 DHCP allocated address to be checked against the MAC address.

macAddress—the MAC address in the format *nnnnnnnnnnnn* or *nn-nn-nn-nn-nn-nn*, where *nn* is a hexadecimal value.

Output/Response

Returns a Boolean value indicating whether the address is allocated.

Related Methods

- *Add MAC Address* on page 198
- *Associate MAC Address* on page 198
- *Deny MAC Address* on page 199
- *Update MAC Address* on page 200
- *MAC Address Generic Methods* on page 200

Update MAC Address

A MAC address's **name** and **macPoolId** properties can be updated using the generic **update()** method. For more information, see *Updating Objects* on page 54.

Related Methods

- *Add MAC Address* on page 198
- *Associate MAC Address* on page 198
- *Deny MAC Address* on page 199
- *Is Address Allocated?* on page 199
- *MAC Address Generic Methods* on page 200

MAC Address Generic Methods

MAC addresses use the generic **get()** and **delete()** methods for entities. For more information, see *Getting Objects* on page 48 and *Deleting Objects* on page 56.

Related Methods

- *Add MAC Address* on page 198
- *Associate MAC Address* on page 198
- *Deny MAC Address* on page 199
- *Is Address Allocated?* on page 199
- *Update MAC Address* on page 200

Workflow Change Requests

You can use change requests to manage the creation of network, resource records, zones, and IP address assignments. Workflow permissions are assigned to users along with access rights. Users with a default access right of **Change**, **Add**, or **Full Access** can be assigned one of these workflow levels:

- **None**—the user is not affected by the change request process and can create networks, resource records, zones, and IP address assignments. Users with the **None** level cannot access or work with workflow change requests.
- **Recommend**—when the user creates, edits, or deletes a network, resource record, zone, or IP address assignment, Proteus creates a change request for the object. The change request must

be approved before the object is actually created, edited, or deleted. Users with the **Recommend** level can review their change requests.

- **Approve**—the user can approve change requests made by other users. Users with the **Approve** level can create, edit, or delete networks, resource records, zones, and IP address assignments.

For information about adding access rights and Workflow Levels, refer to *Add Access Right* on page 187. Users who are assigned the workflow level of **Recommend** create a change request each time they add, edit, or delete a network, resource record, zone, or IP address.

The following objects support workflow mode operations:

- Zone
- HostRecord
- AliasRecord (CName)
- MXRecord
- TXTRecord
- GenericRecord
- HINFORecord
- NAPTRRecord
- SRVRecord
- IP4Network
- IP4Address

The following API operations support workflow mode operations:

- Add (for all objects except IP4Address)
- Update
- Delete

Migration

You can use XML files to migrate data from other systems into Proteus. For more information on the migration document type definition (DTD) and performing migrations from the Proteus web interface, see **Migration** in the *Proteus Administration Guide* and Proteus online help.

The proteus API provides two methods for managing the migration service:

- **migrateFile()** migrates a specified XML file in to Proteus.
- **isMigrationRunning()** indicates if migrations are queued or in progress.

Migrate a File

migrateFile() migrates the specified XML file in to Proteus. The file must be located in the **/data/migration/incoming** directory on the Proteus server. The filename must not include a path.

```
void migrateFile( String filename )
```

Parameters

filename—the filename of the XML file in the data/migration/incoming directory. Do not include a path in the filename.

Output/Response

None.

Migration Status

isMigrationRunning() returns true or false to indicate if the migration service is running. Specify a filename to determine if the specified file is migrating. Specify null to determine if any migration files are migrating or queued for migration.

```
boolean isMigrationRunning( String filename )
```

Parameters

filename—the filename of an XML file in the data/migration/incoming directory. Do not include a path in the filename. This value can be null.

Output/Response

Returns a Boolean value indicating if the specified file is currently migrating. When null is specified for the filename, returns a true if there are any migration files queued for migration or currently migrating.

Chapter 4

API Constants

The Proteus API uses various types of constants in its methods. This list includes all of the constants used in the API methods:

Access Right Values

Property Key	Property Value
HideAccess	HIDE
ViewAccess	VIEW
AddAccess	ADD
ChangeAccess	CHANGE
FullAccess	FULL

DHCP Deployment Role Types

Property Key	Property Value
NONE	NONE
MASTER	MASTER

Deployment Services

Property Key	Property Value
DNS	DNS
DHCP	DHCP
TFTP	TFTP
DHCPv6	DHCPv6

Deployment Status

Property Key	Property Value
EXECUTING	-1
INITIALIZING	0
QUEUED	1
CANCELLED	2
FAILED	3
NOT_DEPLOYED	4
WARNING	5
INVALID	6
DONE	7
NO_RECENT_DEPLOYMENT	8

DHCP Service Options

Property Key	Property Value
DEFAULT_LEASE_TIME	default-lease-time
MAX_LEASE_TIME	max-lease-time
MIN_LEASE_TIME	min-lease-time
CLIENT_UPDATES	client-updates
DDNS_DOMAINNAME	ddns-domainname
DDNS_HOSTNAME	ddns-hostname
DDNS_REV_DOMAINNAME	ddns-rev-domainname
DDNS_TTL	ddns-ttl
DDNS_UPDATES	ddns-updates
PING_CHECK	ping-check
ALWAYS_BROADCAST	always-broadcast
ALWAYS_REPLY_RFC1048	always-reply-rfc1048
DYNAMIC_BOOTP_LEASE_LENGTH	dynamic-bootp-lease-length
FILENAME	filename
GET_LEASE_HOSTNAMES	get-lease-hostnames
MIN_SECS	min-secs
NEXT_SERVER	next-server
SERVER_IDENTIFIER	server-identifier

Property Key	Property Value
SITE_OPTION_SPACE	site-option-space
STASH_AGENT_OPTIONS	stash-agent-options
UPDATE_OPTIMIZATION	update-optimization
UPDATE_STATIC_LEASES	update-static-leases
USE_LEASE_ADDR_FOR_DEFAULT_ROUTE	use-lease-addr-for-default-route
ONE_LEASE_PER_CLIENT	one-lease-per-client
ALLOW_MAC_POOL	allow-mac-pool
DENY_MAC_POOL	deny-mac-pool
DENY_UNKNOWN_MAC_ADDRESSES	deny-unknown-mac-addresses
LOAD_BALANCE_OVERRIDE	load-balance-override
LOAD_BALANCE_SPLIT	load-balance-split
MCLT	mclt
MAX_RESPONSE_DELAY	max-response-delay
MAX_UNACKED_UPDATES	max-unacked-updates
DHCP_CLASS_LEASE_LIMIT	dhcp-class-lease-limit
ALLOW_DHCP_CLASS_MEMBERS	allow-dhcp-class-members
DENY_DHCP_CLASS_MEMBERS	deny-dhcp-class-members
APPLY_MAC_AUTHENTICATION_POLICY	apply-mac-authentication-policy
DENY_DHCP_CLIENTS	deny-dhcp-clients
CONFLICT_DETECTION	conflict-detection
UPDATE_CONFLICT_DETECTION	update-conflict-detection
DO_REVERSE_UPDATES	do-reverse-updates

DHCPServiceOptionConstants

Property Key	Property Value
DDNS_HOSTNAME_TYPE_IP	ip
DDNS_HOSTNAME_TYPE_MAC	mac
DDNS_HOSTNAME_TYPE_FIXED	fixed
DDNS_HOSTNAME_TYPE_DUID	duid
DDNS_HOSTNAME_POSITION_APPEND	append
DDNS_HOSTNAME_POSITION_PREPEND	prepend

DHCP6 Service Options

Property Key	Property Value
DEFAULT_LEASE_TIME	default-lease-time
CLIENT_UPDATES	client-updates
DDNS_DOMAINNAME	ddns-domainname
DDNS_HOSTNAME	ddns-hostname
DDNS_TTL	ddns-ttl
DDNS_UPDATES	ddns-updates
LIMIT_ADDRESSES_PER_IA	limit-addresses-per-ia
DO_REVERSE_UPDATES	do-reverse-updates
SERVER_PREFERENCE	server-preference
PREFERRED_LIFETIME	preferred-lifetime
RAPID_COMMIT	rapid-commit

DHCP Client Options

Property Key	Property Value
TIME_OFFSET	time-offset
ROUTER	router
TIME_SERVER	time-server
IEN_NAME_SERVER	ien-name-server
DNS_SERVER	dns-server
LOG_SERVER	log-server
COOKIE_SERVER	cookie-server
LPR_SERVER	lpr-server
IMPRESS_SERVER	impress-server
RESOURCE_LOCATION_SERVER	resource-location-server
HOST_NAME	host-name
BOOT_SIZE	boot-size
MERIT_DUMP_FILE	merit-dump-file
DOMAIN_NAME	domain-name
SWAP_SERVER	swap-server
ROOT_PATH	root-path
EXTENSIONS_PATH	extensions-path

Property Key	Property Value
IP_FORWARDING	ip-forwarding
NON_LOCAL_SOURCE_ROUTING	non-local-source-routing
POLICY_FILTER_MASKS	policy-filter-masks
MAX_DATAGRAM_REASSEMBLU	max-datagram-reassembly
DEFAULT_IP_TTL	default-ip-ttl
PATH_MTU_AGING_TIMEOUT	path-mtu-aging-timeout
PATH_MTU_PLATEAU_TABLE	path-mtu-plateau-table
INTERFACE_MTU	interface-mtu
ALL_SUBNETS_LOCAL	all-subnets-local
BROADCAST_ADDRESS	broadcast-address
PERFORM_MASK_DISCOVERY	perform-mask-discovery
MASK_SUPPLIER	mask-supplier
ROUTER_DISCOVERY	router-discovery
ROUTER_SOLICITATION_ADDRESS	router-solicitation-address
STATIC_ROUTES	static-routes
TRAILER_ENCAPSULATION	trailer-encapsulation
ARP_CACHE_TIMEOUT	arp-cache-timeout
IEEE_802_3_ENCAPSULATION	ieee-802-3-encapsulation
DEFAULT_TCP_TTL	default-tcp-ttl
TCP_KEEP_ALIVE_INTERVAL	tcp-keep-alive-interval
TCP_KEEP_ALIVE_GARBAGE	tcp-keep-alive-garbage
NIS_DOMAIN	nis-domain
NIS_SERVER	nis-server
VENDOR_ENCAPSULATED_OPTIONS	vendor-encapsulated-options
NTP_SERVER	ntp-server
WINS_NBNS_SERVER	wins-nbns-server
NETBIOS_OVER_TCP_IP_NBDD	netbios-over-tcp-ip-nbdd
WINS_NBT_NODE_TYPE	wins-nbt-node-type
NETBIOS_SCOPE_ID	netbios-scope-id
X_WINDOW_FONT_MANAGER	x-window-font-manager
X_WINDOW_DISPLAY_MANAGER	x-window-display-manager
NETWARE_IP_DOMAIN	nwip.domain
NETWARE_IP_NSQ_BROADCAST	nwip.nsq-broadcast

Property Key	Property Value
NETWARE_IP_PREFERRED_DSS	nwip.preferred-dss
NETWARE_IP_NEAREST_NWIP_SERVER	nwip.nearest-nwip-server
NETWARE_IP_AUTO_RETRIES	nwip.auto-retries
NETWARE_IP_AUTO_RETRY_DELAY	nwip.auto-retry-delay
NETWARE_IP_1_1_COMPATIBILITY	nwip.1-1-compatibility
NETWARE_IP_PRIMARY_DSS	nwip.primary-dss
NIS_PLUS_DOMAIN_NAME	nis-plus-domain-name
NIS_PLUS_SERVER	nis-plus-server
TFTP_SERVER_NAME	tftp-server-name
BOOT_FILE_NAME	boot-file-name
MOBILE_IP_HOME_AGENT	mobile-ip-home-agent
SMTP_SERVER	smtp-server
POP3_SERVER	pop3-server
NNTP_SERVER	nntp-server
WWW_SERVER	www-server
FINGER_SERVER	finger-server
IRC_SERVER	irc-server
STREET_TALK_SERVER	street-talk-server
STREET_TALK_DIRECTORY_ASSISTANCE_SERVER	street-talk-directory-assistance-server
SLP_DIRECTORY_AGENT	slp-directory-agent
SLP_SERVICE_SCOPE	slp-service-scope
NDS_SERVER	nds-server
NDS_TREE_NAME	nds-tree-name
NDS_CONTEXT	nds-context
UAP_SERVER	uap-server
NAME_SERVICE_SEARCH	name-service-search
DOMAIN_SEARCH	domain-search
SIP_SERVERS	sip-server
CLASSLESS_STATIC_ROUTE_OPTION	classless-static-route-option
CCC_PRIMARY_DHCP_SERVER_ADDRESS	cablelabs.primary-dhcp-server
CCC_SECONDARY_DHCP_SERVER_ADDRESS	cablelabs.secondary-dhcp-server
CCC_PROVISIONING_SERVER_ADDRESS	cablelabs.provisioning-server
CCC_AS_BACKOFF_AND_RETRY	cablelabs.as-backoff-retry

Property Key	Property Value
CCC_AP_BACKOFF_RETRY	cablelabs.ap-backoff-retry
CCC_KERBEROS_REALM_NAME	cablelabs.kerberos-realm-name
CCC_TICKET_GRANTING_SERVER_UTILIZATION	cablelabs.ticket-granting-server-utilization
CCC_PROVISIONING_TIMER_VALUE	cablelabs.provisioning-timer-value
TFTP_SERVER_ADDRESS	tftp-server
IP_TELEPHONE	ip-telephone
WPAD_URL	wpad-url

DHCP6 Client Options

Property Key	Property Value
UNICAST	unicast
DNS_SERVERS	dns-servers
DOMAIN_SEARCH_LIST	domain-search-list
SNTP_SERVERS	sntp-servers
INFORMATION_REFRESH_TIME	information-refresh-time

DHCP Class Match Criteria

Property Key	Property Value
DHCP_CLASS_HARDWARE	MATCH_HARDWARE
DHCP_CLASS_CLIENT_ID	MATCH_DHCP_CLIENT_ID
DHCP_CLASS_VENDOR_ID	MATCH_DHCP_VENDOR_ID
DHCP_CLASS_AGENT_CIRCUIT_ID	MATCH_AGENT_CIRCUIT_ID
DHCP_CLASS_AGENT_REMOTE_ID	MATCH_AGENT_REMOTE_ID
DHCP_CLASS_CUSTOM_MATCH	CUSTOM_MATCH
DHCP_CLASS_CUSTOM_MATCH_IF	CUSTOM_MATCH_IF

DHCP Custom Option Types

Property Key	Property Value
IP4	IP4
TEXT	TEXT
UNSIGNED_INT_8	UNSIGNED_INT_8
UNSIGNED_INT_16	UNSIGNED_INT_16
UNSIGNED_INT_32	UNSIGNED_INT_32
UNSIGNED_INT_64	UNSIGNED_INT_64
SIGNED_INT_8	SIGNED_INT_8
SIGNED_INT_16	SIGNED_INT_16
SIGNED_INT_32	SIGNED_INT_32
BOOLEAN	BOOLEAN
IP4_MASK	IP4_MASK
IP4_RANGE	IP4_RANGE
IP4_BLOCK	IP4_BLOCK
STRING	STRING
BINARY	BINARY
ENCAPSULATED	ENCAPSULATED

DNS Options

Property Key	Property Value
ALLOW_XFER	allow-xfer
ALSO_NOTIFY	also-notify
ALLOW_DDNS	allow-ddns
ALLOW_RECURSION	allow-recursion
ALLOW_QUERY	allow-query
FORWARDING_POLICY	forwarding-policy
FORWARDING	forwarding
NOTIFY	notify
MAX_CACHE_TTL	max-cache-ttl
MAX_NEG_CACHE_TTL	max-neg-cache-ttl
TRANSFERS_IN	transfers-in
TRANSFERS_OUT	transfers-out

Property Key	Property Value
TCP_CLIENTS	tcp-clients
MAX_TRANSFER_TIME_OUT	max-transfer-time-out
MAX_TRANSFER_TIME_IN	max-transfer-time-in
MAX_TRANSFER_IDLE_OUT	max-transfer-idle-out
MAX_TRANSFER_IDLE_IN	max-transfer-idle-in
TRANSFER_FORMAT	transfer-format
MAX_CACHE_SIZE	max-cache-size
RECURSIVE_CLIENTS	recursive-clients
TRANSFERS_PER_NS	transfers-per-ns
LAME_TTL	lame-ttl
ALLOW_UPDATE_FORWARDING	allow-update-forwarding
VERSION	version
MATCH_CLIENTS	match-clients
DENY_CLIENTS	deny-clients
CACHE	cache
ALLOW_NOTIFY	allow-notify
ZONE_DEFAULT_TTL	zone-default-ttl
DNSSEC_ENABLE	dnssec-enable
DNSSEC_VALIDATION	dnssec-validation
DNSSEC_KEY_DIRECTORY	dnssec-key-directory
DNSSEC_TRUST_ANCHORS	dnssec-trust-anchors
DNSSEC_MUST_BE_SECURE	dnssec-must-be-secure
ALLOW_QUERY_CACHE	allow-query-cache
DNSSEC_ACCEPT_EXPIRED	dnssec-accept-expired
START_OF_AUTHORITY	start-of-authority

DNS Option Values

Property Key	Property Value
SINGLE	SINGLE
MANY_ANSWERS	MANY_ANSWERS
FIRST	FIRST
ONLY	ONLY

DNS Deployment Role Types

Property Key	Property Value
NONE	NONE
MASTER	MASTER
MASTER_HIDDEN	MASTER_HIDDEN
SLAVE	SLAVE
SLAVE_STEALTH	SLAVE_STEALTH
FORWARDER	FORWARDER
STUB	STUB
RECURSION	RECURSION
AD_MASTER	AD_MASTER

DNS Zones Deployment Validation Check

Property Key	Property Value
FAIL	FAIL
WARN	WARN
IGNORE	IGNORE
NONE	NONE
FULL	FULL
FULL_SIBLING	FULL_SIBLING
LOCAL	LOCAL
LOCAL_SIBLING	LOCAL_SIBLING

DNSSEC Key Format

Property Key	Property Value
TRUST_ANCHOR	TRUST_ANCHOR
DNS_KEY	DNS_KEY
DS_RECORD	DS_RECORD

IP Assignment Action Values

Property Key	Property Value
MAKE_STATIC	MAKE_STATIC
MAKE_RESERVED	MAKE_RESERVED
MAKE_DHCP_RESERVED	MAKE_DHCP_RESERVED

Object Properties

Property Key	Property Value
name	name
sharedNetwork	sharedNetwork
CIDR	CIDR
start	start
end	end
template	template
deployable	deployable
authenticator	authenticator
securityPrivilege	securityPrivilege
historyPrivilege	historyPrivilege
email	email
phoneNumber	phoneNumber
users	users
version	version
description	description
addresses	addresses
address	address
state	state
server	server
serverInterface	serverInterface
zoneTransServerInterface	zoneTransServerInterface
macPool	macPool
view	view
refresh	refresh
retry	retry

Property Key	Property Value
expire	expire
minimum	minimum
absoluteName	absoluteName
userAccessType	userAccessType
allowDuplicateHost	allowDuplicateHost
pingBeforeAssign	pingBeforeAssign
defaultDomains	defaultDomains
dnsRestrictions	dnsRestrictions
defaultView	defaultView
comments	comments
ttl	ttl
reverseRecord	reverseRecord
txt	txt
parentZoneName	parentZoneName
linkedParentZoneName	linkedParentZoneName
linkedRecordName	linkedRecordName
priority	priority
port	port
weight	weight
order	order
preference	preference
service	service
regexp	regexp
replacement	replacement
flags	flags
os	os
cpu	cpu
type	type
rdata	rdata
prefix	prefix
identifier	identifier
parentId	parentId
parentType	parentType

Property Key	Property Value
addressIds	addressIds
linkToExternalHost	linkToExternalHost
defaultInterfaceAddress	defaultInterfaceAddress
publishedInterfaceAddress	publishedInterfaceAddress
secondaryServerInterfaceId	secondaryServerInterfaceId
fullHostName	fullHostName
profile	profile
connected	connected
upgrade	upgrade
readOnly	readOnly
servicesIPv4Address	servicesIPv4Address
servicesIPv4Netmask	servicesIPv4Netmask
servicesIPv6Address	servicesIPv6Address
servicesIPv6Subnet	servicesIPv6Subnet
xhaIPv4Address	xhaIPv4Address
xhaIPv4Netmask	xhaIPv4Netmask
redundancyScenario	redundancyScenario
xHAServerId	xHAServerId
activeServerId	activeServerId
passiveServerId	passiveServerId
activeServerNewIPv4Address	activeServerNewIPv4Address
activeServerPassword	activeServerPassword
passiveServerPassword	passiveServerPassword
pingAddress	pingAddress
ip6Address	ip6Address
newManagementAddress	newManagementAddress
activeServerIPv4AddressForNAT	activeServerIPv4AddressForNAT
passiveServerIPv4AddressForNAT	passiveServerIPv4AddressForNAT
activeServerNewIPv4AddressForNAT	activeServerNewIPv4AddressForNAT
backboneActiveServerIPv4Address	backboneActiveServerIPv4Address
backboneActiveServerIPv4Netmask	backboneActiveServerIPv4Netmask

Property Key	Property Value
backbonePassiveServerIPv4Address	backbonePassiveServerIPv4Address
backbonePassiveServerIPv4Netmask	backbonePassiveServerIPv4Netmask
nodeType	nodeType
breakInProteusOnly	breakInProteusOnly
overrideDHCPValidation	overrideDHCPValidation
checkDHCPConfigurationDeployment	checkDHCPConfigurationDeployment
overrideDNSValidation	overrideDNSValidation
checkDNSConfigurationDeployment	checkDNSConfigurationDeployment
checkDNSZonesDeployment	checkDNSZonesDeployment
postLoadZoneIntegrityValidationDNSDeploy	postLoadZoneIntegrityValidationDNSDeploy
checkNamesValidationModeDNSDeploy	checkNamesValidationModeDNSDeploy
checkIfMXRecordsAreIPsDNSDeploy	checkIfMXRecordsAreIPsDNSDeploy
checkIfMXRecordsPointToCNAMEsDNSDeploy	checkIfMXRecordsPointToCNAMEsDNSDeploy
checkIfNSRecordsAreIPsDNSDeploy	checkIfNSRecordsAreIPsDNSDeploy
checkIfSRVRecordsPointToCNAMEsDNSDeploy	checkIfSRVRecordsPointToCNAMEsDNSDeploy
checkForNonTerminalWildcardsDNSDeploy	checkForNonTerminalWildcardsDNSDeploy
ProteusDDW	ProteusDDW
enableDHCP	enableDHCP
enableDNS	enableDNS
services	services
importViewName	importViewName
authenticationCredentialDomain	authenticationCredentialDomain
authenticationCredentialUsername	authenticationCredentialUsername
authenticationCredentialPassword	authenticationCredentialPassword
forceDNSFullDeployment	forceDNSFullDeployment
gateway	gateway
reservedAddresses	reservedAddresses
reservedBlock	RESERVED_BLOCK
reservedDHCPRange	RESERVED_DHCP_RANGE

Property Key	Property Value
ipGroup	IP_GROUP
templateType	templateType
zoneTemplateType	zonetemplate
IP4NetworkTemplateType	ip4networktemplate
zoneTemplateReapplyMode	zoneReapplyMode
templateReapplyModeIgnore	IGNORE
templateReapplyModeUpdate	UPDATE_IF_POSSIBLE
templateReapplyModeOverwrite	OVERWRITE
gatewayReapplyMode	gatewayReapplyMode
reservedAddressesReapplyMode	reservedAddressesReapplyMode
dhcpRangesReapplyMode	dhcpRangesReapplyMode
ipGroupsReapplyMode	ipGroupsReapplyMode
optionsReapplyMode	optionsReapplyMode
noGateway	noGateway
seedRouterAddress	seedRouterAddress
snmpVersion	snmpVersion
snmpPortNumber	snmpPortNumber
snmpCommunityString	snmpCommunityString
securityLevel	securityLevel
context	context
authenticationType	authenticationType
authPassphrase	authPassphrase
privacyPassphrase	privacyPassphrase
networkBoundaries	networkBoundaries
schedule	schedule
activeStatus	activeStatus
enableLayer2Discovery	enableLayer2Discovery
acceptanceCriteriaReclaim	acceptanceCriteriaReclaim
acceptanceCriteriaUnknown	acceptanceCriteriaUnknown

Property Key	Property Value
acceptanceCriteriaMismatch	acceptanceCriteriaMismatch
overrideList	overrideList
matchCriteria	matchCriteria
matchOffset	matchOffset
matchLength	matchLength
customMatchRawString	customMatchRawString
ignoreError	ignoreError
matchValue	matchValue
splitStaticAddresses	splitStaticAddresses
noServerUpdate	noServerUpdate
transientParent	transientParent
optionId	optionId
optionType	optionType
optionAllowMultiple	optionAllowMultiple
optionDescription	optionDescription
deviceTypeId	deviceTypeId
deviceSubtypeId	deviceSubtypeId
ip4Addresses	ip4Addresses
ip6Addresses	ip6Addresses
overrideNamingPolicy	overrideNamingPolicy
deleteKeys	deleteKeys
excludeDHCPRange	excludeDHCPRange
skip	skip
offset	offset
displayName	displayName
hint	hint
accessRight	accessRight
overrideType	overrideType
retrieveFields	retrieveFields

Property Key	Property Value
ignoreCase	ignoreCase
size	size
positionRangeBy	positionRangeBy
positionValue	positionValue
ipGroupBySize	ipGroupBySize
configName	configName
deviceName	deviceName
ipAddressMode	ipAddressMode
ipEntity	ipEntity
viewName	viewName
zoneName	zoneName
recordName	recordName
macAddressMode	macAddressMode
macEntity	macEntity
VCO_MODE_REQUEST_VALUE	REQUEST_VALUE
VCO_MODE_REQUEST_STATIC	REQUEST_STATIC
VCO_MODE_REQUEST_DHCP_RESERVED	REQUEST_DHCP_RESERVED
VCO_MODE_PASS_VALUE	PASS_VALUE
allowDuplicateHosts	allowDuplicateHosts
netmask	netmask
ip	ip
inherited	inherited
redirectTarget	redirectTarget
responsePolicyType	responsePolicyType
workflowLevel	workflowLevel
deploymentAllowed	deploymentAllowed
quickDeploymentAllowed	quickDeploymentAllowed
TraversalMethodology.NO_TRAVERSAL	NO_TRAVERSAL
TraversalMethodology.DEPTH_FIRST	DEPTH_FIRST
TraversalMethodology.BREADTH_FIRST	BREADTH_FIRST

Property Key	Property Value
reuseExisting	reuseExisting
secondStandbyServer	secondStandbyServer

Object Types

Property Key	Property Value
Entity	Entity
Configuration	Configuration
View	View
Zone	Zone
InternalRootZone	InternalRootZone
ZoneTemplate	ZoneTemplate
EnumZone	EnumZone
EnumNumber	EnumNumber
HostRecord	HostRecord
AliasRecord	AliasRecord
MXRecord	MXRecord
TXTRecord	TXTRecord
SRVRecord	SRVRecord
GenericRecord	GenericRecord
HINFORecord	HINFORecord
NAPTRRecord	NAPTRRecord
RecordWithLink	RecordWithLink
ExternalHostRecord	ExternalHostRecord
StartOfAuthority	StartOfAuthority
IP4Block	IP4Block
IP4Network	IP4Network
IP6Block	IP6Block
IP6Network	IP6Network
IP6Address	IP6Address
IP4NetworkTemplate	IP4NetworkTemplate

Property Key	Property Value
DHCP4Range	DHCP4Range
DHCP6Range	DHCP6Range
IP4Address	IP4Address
MACPool	MACPool
DenyMACPool	DenyMACPool
MACAddress	MACAddress
TagGroup	TagGroup
Tag	Tag
User	User
UserGroup	UserGroup
Server	Server
NetworkServerInterface	NetworkServerInterface
PublishedServerInterface	PublishedServerInterface
NetworkInterface	NetworkInterface
VirtualInterface	VirtualInterface
LDAP	LDAP
Kerberos	Kerberos
KerberosRealm	KerberosRealm
Radius	Radius
TFTPGroup	TFTPGroup
TFTPFolder	TFTPFolder
TFTPFile	TFTPFile
TFTPDeploymentRole	TFTPDeploymentRole
DNSDeploymentRole	DNSDeploymentRole
DHCPDeploymentRole	DHCPDeploymentRole
DNSOption	DNSOption
DHCPV4ClientOption	DHCPV4ClientOption
DHCPServiceOption	DHCPServiceOption
DHCPRawOption	DHCPRawOption

Property Key	Property Value
DNSRawOption	DNSRawOption
DHCPV6ClientOption	DHCPV6ClientOption
DHCPV6ServiceOption	DHCPV6ServiceOption
VendorProfile	VendorProfile
VendorOptionDef	VendorOptionDef
VendorClientOption	VendorClientOption
CustomOptionDef	CustomOptionDef
DHCPMatchClass	DHCPMatchClass
DHCPSubClass	DHCPSubClass
Device	Device
DeviceType	DeviceType
DeviceSubtype	DeviceSubtype
DeploymentScheduler	DeploymentScheduler
IP4ReconciliationPolicy	IP4ReconciliationPolicy
DNSSECSigningPolicy	DNSSECSigningPolicy
IP4IPGroup	IP4IPGroup
ResponsePolicy	ResponsePolicy

PositionRangeBy

Property Key	Property Value
START_OFFSET	START_OFFSET
END_OFFSET	END_OFFSET
START_ADDRESS	START_ADDRESS

Response Policy Type

Property Key	Property Value
BLACKLIST	BLACKLIST
BLACKHOLE	BLACKHOLE
WHITELIST	WHITELIST

Entity Categories

Property Key	Property Value
all	ALL
admin	ADMIN
Configuration	CONFIGURATION
deploymentOptions	DEPLOYMENT_OPTIONS
deploymentRoles	DEPLOYMENT_ROLES
deploymentSchedulers	DEPLOYMENT_SCHEDULER
dhcpClassObjects	DHCPCLASSES_OBJECTS
dhcpNACPolicies	DHCPNACPOLICY_OBJECTS
IP4Objects	IP4_OBJECTS
IP6Objects	IP6_OBJECTS
MACPoolObjects	MACPOOL_OBJECTS
resourceRecords	RESOURCE_RECORD
servers	SERVERS
tags	TAGS
tasks	TASKS
TFTPObjects	TFTP_OBJECTS
vendorProfiles	VENDOR_PROFILES
viewZones	VIEWS_ZONES
TSIGKeys	TSIG_KEYS
GSS	GSS
DHCPZones	DHCP_ZONES

ENUM Services

Property Key	Property Value
H323	H323
SIP	SIP
ifax_mailto	ifax mailto
pres	pres
web_http	web http
web_https	web https
ft_ftp	ft ftp

Property Key	Property Value
email_mailto	email mailto
fax_tel	fax tel
sms_tel	sms tel
sms_mailto	sms mailto
ems_tel	ems tel
ems_mailto	ems mailto
mms_tel	mms tel
mms_mailto	mms mailto
VPIM_MAILTO	VPIM MAILTO
VPIM_LDAP	VPIM LDAP
voice_tel	voice tel
pstn_tel	pstn tel
pstn_sip	pstn sip
xmpp	xmpp
im	im

User-defined Field Type

Property Key	Property Value
TEXT	TEXT
DATE	DATE
BOOLEAN	BOOLEAN
INTEGER	INTEGER
LONG	LONG
EMAIL	EMAIL
URL	URL

User-defined Field Validator Properties

Property Key	Property Value
MIN	min
MAX	max
MIN_LENGTH	minLength
MAX_LENGTH	maxLength

Property Key	Property Value
PATTERN	pattern

User History Privileges

Property Key	Property Value
HIDE	HIDE
VIEW_HISTORY_LIST	VIEW_HISTORY_LIST

User Type

Property Key	Property Value
ADMIN	ADMIN
REGULAR	REGULAR

User Access Type

Property Key	Property Value
GUI	GUI
API	API
GUI_AND_API	GUI_AND_API

User Security Privileges

Property Key	Property Value
NO_ACCESS	NO_ACCESS
VIEW_MY_ACCESS_RIGHTS	VIEW_MY_ACCESS_RIGHTS
VIEW_OTHERS_ACCESS_RIGHTS	VIEW_OTHERS_ACCESS_RIGHTS
CHANGE_ACCESS_RIGHTS	CHANGE_ACCESS_RIGHTS
ADD_ACCESS_RIGHTS	ADD_ACCESS_RIGHTS
DELETE_ACCESS_RIGHTS	DELETE_ACCESS_RIGHTS

Workflow Levels

Property Key	Property Value
None	NONE
Recommend	RECOMMEND
Approve	APPROVE

SNMPSecurityLevels

Property Key	Property Value
AUTH_PRIV	AUTH_PRIV
AUTH_NOPRIV	AUTH_NOPRIV
NOAUTH_NOPRIV	NOAUTH_NOPRIV

Server Capability Profiles

Property Key	Property Value
ADONIS_250	ADONIS_250
ADONIS_500	ADONIS_500
ADONIS_750	ADONIS_750
ADONIS_800	ADONIS_800
ADONIS_1000	ADONIS_1000
ADONIS_1200	ADONIS_1200
ADONIS_1750	ADONIS_1750
ADONIS_1900	ADONIS_1900

Property Key	Property Value
ADONIS_1925	ADONIS_1925
ADONIS_1950	ADONIS_1950
ADONIS_XMB	ADONIS_XMB
ADONIS_XMB2	ADONIS_XMB2
AFILIAS_DNS_SERVER	AFILIAS_DNS_SERVER
OTHER_DNS_SERVER	OTHER_DNS_SERVER
PROTEUS_DDW	PROTEUS_DDW
WINDOWS_SERVER	WINDOWS_SERVER

Service Types

Property Key	Property Value
DNS	DNS
DHCP	DHCP

Traversal Methodology

Property Key	Property Value
TraversalMethodology.NO_TRAVERSAL	NO_TRAVERSAL
TraversalMethodology.DEPTH_FIRST	DEPTH_FIRST
TraversalMethodology.BREADTH_FIRST	BREADTH_FIRST

Vendor Profile Option Types

Property Key	Property Value
IP4	IP4
TEXT	TEXT
UNSIGNED_INT_8	UNSIGNED_INT_8
UNSIGNED_INT_16	UNSIGNED_INT_16
UNSIGNED_INT_32	UNSIGNED_INT_32
UNSIGNED_INT_64	UNSIGNED_INT_64
SIGNED_INT_8	SIGNED_INT_8
SIGNED_INT_16	SIGNED_INT_16
SIGNED_INT_32	SIGNED_INT_32

Property Key	Property Value
BOOLEAN	BOOLEAN
IP4_MASK	IP4_MASK
STRING	STRING
BINARY	BINARY
ENCAPSULATED	ENCAPSULATED

Chapter 5

API Method Reference

API Sessions

Log in and Log out	login (String *name*, String *password*), logout()
System Information	String getSystemInfo()

Generic Methods

Updating Objects	void update (APIEntity *entity*) All extensions of this method in this table list only mutable parameters.
Update with Options	void updateWithOptions (APIEntity *entity*, String *options*)
Deleting Objects	void delete (long *ObjectId*)
Delete with Options	void deleteWithOptions (long *objectId*, String *options*)
Get Entity by Name	APIEntity getEntityByName (long *parentId*, String *name*, String *type*)
Get Entity by ID	APIEntity getEntityById (long *id*)
Get Entities	APIEntity[] getEntities (long *parentId*, String *type*, int *start*, int *count*)
Get Parent	APIEntity[] getParent(long *entityId*)
Get Entities by Name	APIEntity[] getEntitiesByName (long *parentId*, String *name*, String *type*, int *start* ,int *count*)
Get Entities by Name Using Options	APIEntity[] getEntitiesByNameUsingOptions (long *parentId*, String *name*, String *type*, int *start* ,int *count,* String *options*)
Get MAC Address	APIEntity[] getMACAddress (long *configurationId*, String *macAddress*)
Get Linked Entities	APPIEntity[] getLinkedEntities (long *entityId*, String *type*, int *start*, int *count*)

Search by Category	`APIEntity[] searchByCategory ( String` *`keyword`*`, String` *`category`*`, int` *`start`*`, int` *`count`* `)`
Search by Object Types	`APIEntity[] searchByObjectTypes ( String` *`keyword`*`, String` *`types`*`, int` *`start`*`, int` *`count`* `)`

Linked Entities

Link Entities	`void linkEntities ( long` *`entity1Id`*`, long` *`entity2Id`*`, String` *`properties`* `)`
Unlink Entities	`void unlinkEntities ( long` *`entity1Id`*`, long` *`entity2Id`*`, String` *`properties`* `)`

User-defined Fields

Get User-defined Field	`public APIUserDefinedFields[] getUserDefinedFields ( String` *`type`*`, boolean` *`requiredFieldsOnly`* `)`
Update Bulk User-defined Field	`public byte[] updateBulkUdf ( byte[]` *`data`*`, String` *`properties`* `)`

IPAM

IPv4 Blocks

Add IPv4 Block by CIDR	`long addIP4BlockByCIDR ( long parentId, String CIDR, String properties )`
Add IPv4 Block by Range	`long addIP4BlockByRange ( long parentId, String start, String end, String properties )`
Add Parent Block	`void addParentBlock ( long[] blockOrNetworkIDs )`
Add Parent Block with Properties	`public long addParentBlockWithProperties ( long[] blockOrNetworkIDs String properties )`
Get IP Range by IP Address	`APIEntity getIPRangedByIP ( long containerId, String type, String address )`
Get IPv4 Block by CIDR	`APIEntity getEntityByCIDR ( long parentId, String cidr, String type )`
Get IPv4 Block by Range	`APIEntity getEntityByRange ( long parentId, String address1, String address2, String type )`
Merge Blocks with Parent	`void mergeBlocksWithParent ( long[] blockIDs )`
Merge Selected Blocks or Networks	`void mergeSelectedBlocksOrNetworks ( long[] blockOrNetworkIds, long blockOrNetworkToKeep )`
Move IPv4 Object	`void moveIP4Object ( long objectId, String address )`
Move IP Object	`void moveIP4Object ( long objectId, String address, String options )`
Resize Range	`void resizeRange ( long objectId, String range, String options )`
Update IPv4 Block	`void update ( APIEntity entity )` For more information, see generic update() method.
IPv4 Block Generic Methods	`void delete( long objectId )`

IPv4 Networks

Add IPv4 Network	`long addIP4Network ( long blockId, String CIDR, String properties )`
Get IPv4 Range by IP Address	`APIEntity getIPRangedByIP ( long containerId, String type, String address )`
Get IPv4 Network by CIDR	`APIEntity getEntityByCIDR ( long parentId, String cidr, String type )`
Get IPv4 Network by Hint	`APIEntity[] getIP4NetworksByHint( long containerId, int start, int count, String options )`

Get IPv4 Network by Range	APIEntity getEntityByRange (long *parentId*, String *address1*, String *address2*, String *type*)
Get Next Available Network	long getNextAvailableIP4Network (long *parentId*, long *size*, boolean *isLargerAllowed*, boolean *autoCreate*)
Get Next Available IP Range	APIEntity getNextAvailableIPRange (long *parentId*, long *size*, String *type,* String *properties*)
Get Next Available IP Range	public APIEntity getNextAvailableIPRanges (long *parentId*, long *size*, String *type,* int *count,* String *properties*)
Split IPv4 Network	APIEntity[] splitIP4Network (long *networkId*, int *numberOfParts*, String *options*)
Merge Selected Blocks or Networks	void mergeSelectedBlocksOrNetworks (long[] *blockOrNetworkIds*, long *blockOrNetworkToKeep*)
Move IPv4 Object	void moveIP4Object (long *objectId*, String *address*)
Move IP Object	void moveIP4Object (long *objectId*, String *address,* String *options*)
Resize Range	void resizeRange (long *objectId*, String *range* String *options*)
Update IPv4 Network	void update (APIEntity *entity*) For more information, see generic update() method.
IPv4 Network Generic Methods	void delete(long *objectId*)
Add IPv4 Reconciliation Policy	long addIP4ReconciliationPolicy(long *parentId*, string *name*, string *properties*)

IPv4 Network Templates

Update IPv4 Network Template Name	void update (APIEntity *entity*) For more information, see generic update() method.
IPv4 Network Template Generic Methods	get() void delete(long *objectId*)
Add IPv4 Network Template	long addIP4NetworkTemplate (long *configurationId*, String *name*, String *properties*)
Assign or Update Template	void assignOrUpdateTemplate (long *entityId*, long *templateId*, String *properties*)
Re-apply Template	void reapplyTemplate (long *templateId*, String *properties*)

IPv4 Addresses

Assign IPv4 Address	long assignIP4Address (long *configurationId*, String *ip4Address*, String *macAddress*, String *hostInfo*, String *action*, String *properties*)
Assign Next Available IPv4 Address	APIEntity assignNextAvailableIP4Address (long *configurationId*, long *parentId*, String *macAddress*, String *hostInfo*, String *action*, String *properties*)
Get IPv4 Address	APIEntity getIP4Address (long *containerId*, String *address*)
Get Next IPv4 Address	APIEntity getNextIP4Address(long *parentId*, String *properties*)
Check Allocation for IPv4 Address	boolean isAddressAllocated (long *configurationId*, String *ipAddress*, String *macAddress*)
Allocate Next Available Address	String getNextAvailableIP4Address (long *parentId*)
Get Dependent Records	This method is deprecated. Using this method now returns an error message. Use the **getLinkedEntities()** method instead. For more information, see *Get Linked Entities* on page 49.
Update IPv4 Address	void update (APIEntity *entity*) For more information, see generic update() method.
IPv4 Address Generic Methods	void delete(long *objectId*)
Change IPv4 Address State	void changeStateIP4Address(long *addressId,* String *targetState,* String *macAddress*)

IPv4 Objects

Move IPv4 Object	void moveIP4Object (long *objectId*, String *address*)
Move IP Object	void moveIP4Object (long *objectId*, String *address*, String *options*)
Resize Range	void resizeRange (long *objectId*, String *range*, String *options*)

IPv4 Group

Add IPv4 IP Group by Range	long addIP4IPGroupByRange(long *parentId*, String *start*, String *end*, String *properties*)
Add IPv4 IP Group by Size	long addIP4IPGroupBySize (long *parentId*, String *name*, int *size*, String *positionRangeBy*, String *positionValue*, String *properties*)

IPv6 Objects

Add IPv6 Address	long addIP6Address (long *containerId*, String *address*, String *type*, String *name*, String *properties*)
Add IPv6 Block by MAC Address	long addIP6BlockByMACAddress (long *parentId*, String *macAddress*, String *name*, String *properties*)
Add IPv6 Block by Prefix	long addIP6BlockByPrefix (long *parentId*, String *prefix*, String *name*, String *properties*)
Add IPv6 Network by Prefix	long addIP6NetworkByPrefix (long *parentId*, String *prefix*, String *name*, String *properties*)
Get IPv6 Range by IP Address	APIEntity getIPRangedByIP (long *containerId*, String *type*, String *address*)
Assign IPv6 Address	boolean assignIP6Address (long *containerId*, String *address*, String *action*, String *macAddress*, String *hostInfo*, String *properties*)
Clear IPv6 Address	boolean clearIP6Address (long *addressId*)
Get Entity by Prefix	APPIEntity getEntityByPrefix (long *containerId*, String *prefix*, String *type*)
Get IPv6 Address	APPIEntity getIP6Address (long *containerId*, String *address*)
Reassign IPv6 Address	long reassignIP6Address (long *oldAddressId*, String *destination*, String *properties*)
Update IPv6 Objects	void update (APIEntity *entity*) For more information, see generic update() method.

Provision Devices

Add Device Instance	`String addDeviceInstance ( String configName, String deviceName, String ipAddressMode, String ipEntity, String viewName, String zoneName, String recordName, String macAddressMode, String macEntity, String options )`

De-provision Devices

Delete Device Instance	`deleteDeviceInstance ( String configName, String identifier, String options )`

DHCP

IPv4 DHCP Ranges

Add IPv4 DHCP Range	long addDHCP4Range (long *networkId*, String *start*, String *end*, String *properties*)
Add IPv4 DHCP Range By Size	long addDHCP4RangeBySize (long *networkId*, String *offset*, String *size*, String *properties*)
Get IPv4 Range by IP Address	APIEntity getIPRangedByIP (long *containerId*, String *type*, String *address*)
Get IPv4 DHCP Range	APIEntity getEntityByRange (long *parentId*, String *address1*, String *address2*, String *type*)
Get IPv4 DHCP Ranges	APIEntity[] getEntities (long *parentId*, String *type*, int *start*, int *count*)
Get Max Allowed Range	public String[] getMaxAllowedRange (long *rangeId*)
Update IPv4 DHCP Range	void update (APIEntity *entity*) For more information, see generic update() method.
IPv4 DHCP Range Generic Methods	void delete(long *objectId*)

IPv6 DHCP Ranges

Add IPv6 DHCP Range	long addDHCP6Range(long *networkId*, String *start*, String *end*, String *properties*)
Get IPv6 Range by IP Address	APIEntity getIPRangedByIP (long *containerId*, String *type*, String *address*)
Get IPv6 DHCP Range	APIEntity getEntityByRange (long *parentId*, String *address1*, String *address2*, String *type*)
Get Multiple IPv6 DHCP Ranges	APIEntity[] getEntities (long *parentId*, String *type*, int *start*, int *count*)
Update IPv6 DHCP Range	void update (APIEntity *entity*) For more information, see generic update() method.
IPv6 DHCP Range Generic Methods	void delete(long *objectId*)

DHCP Client Options

Add DHCP Client Option	long addDHCPClientDeploymentOption (long *entityId*, String *name*, String *value*, String *properties*)

Get DHCP Client Option	APIDeploymentOption getDHCPClientDeploymentOption (long *entityId*, String *name*, long *serverId*)
Update DHCP Client Option	void updateDHCPClientDeploymentOption (APIDeploymentOption *option*)
Delete DHCP Client Option	void deleteDHCPClientDeploymentOption (long *entityId*, String *name*, long *serverId*)

DHCP6 Client Options

Add DHCP6 Client Option	long addDHCP6ClientDeploymentOption (long *entityId*, String *name*, String *value*, String *properties*)
Get DHCP6 Client Option	APIDeploymentOption getDHCP6ClientDeploymentOption (long *entityId*, String *name*, long *serverId*)
Update DHCP6 Client Option	void updateDHCP6ClientDeploymentOption (APIDeploymentOption *option*)
Delete DHCP6 Client Option	void deleteDHCP6ClientDeploymentOption (long *entityId*, String *name*, long *serverId*)

DHCP Custom Options

Add Custom Deployment Option	long addCustomOptionDefinition (long *configurationId*, String *name*, long *optionId*, String *optionType*, boolean *allowMultiple*, String *properties*)

DHCP Service Options

Add DHCP Service Option	long addDHCPServiceDeploymentOption (long *entityId*, String *name*, String *value*, String *properties*)
Get DHCP Service Option	APIDeploymentOption getDHCPServiceDeploymentOption (long *entityId*, String *name*, long *serverId*)
Update DHCP Service Option	void updateDHCPServiceDeploymentOption (APIDeploymentOption *option*)
Delete DHCP Service Option	void deleteDHCPServiceDeploymentOption (long *entityId*, String *name*, long *serverId*)

DHCP6 Service Options

Add DHCP6 Service Option	long addDHCP6ServiceDeploymentOption (long *entityId*, String *name*, String *value*, String *properties*)
Get DHCP6 Service Option	APIDeploymentOption getDHCP6ServiceDeploymentOption (long *entityId*, String *name*, long *serverId*)
Update DHCP6 Service Option	void updateDHCP6ServiceDeploymentOption (APIDeploymentOption *option*)
Delete DHCP6 Service Option	void deleteDHCP6ServiceDeploymentOption (long *entityId*, String *name*, long *serverId*)

DHCP Vendor Options

Add DHCP Vendor Deployment Option	long addDHCPVendorDeploymentOption (long *parentId*, long *optionId*, String *value*, String *properties*)
Add Vendor Option Definition	long addVendorOptionDefinition (long *vendorProfileId*, long *optionId*, String *name*, String *optionType*, String *description*, boolean *allowMultiple*, String *properties*)
Add Vendor Profile	long addVendorProfile (String *identifier*, String *name*, String *description*, String *properties*)
Delete DHCP Vendor Deployment Option	void deleteDHCPVendorDeploymentOption (long *entityId*, long *optionId*, long *serverId*)
Get DHCP Vendor Deployment Option	APIDeploymentOption getDHCPVendorDeploymentOption (long *entityId*, long *optionId*, long *serverId*)
Update DHCP Vendor Deployment Option	void updateDHCPVendorDeploymentOption (APIDeploymentOption *option*)

DHCP Match Classes

Add DHCP Match Classes	long addDHCPMatchClass (long *configurationId*, String *name*, String *matchCriteria*, String *properties*)
Update DHCP Match Classes	void update (APIEntity *entity*) For more information, see generic update() method.
Delete DHCP Match Classes	void delete(long *objectId*)
Add DHCP Sub Classes	long addDHCPSubClass (long *matchClassId*, String *matchValue*, String *properties*)

Update DHCP Sub Classes	`void update ( APIEntity entity )` For more information, see generic update() method.
Delete DHCP Match Classes	`void delete( long objectId )`

DNS

DNS Views

Add DNS View	long addView (long *configurationId*, String *name*, String *properties*)
Update DNS View	void update (APIEntity *entity*) For more information, see generic update() method.
DNS View Generic Methods	getEntity() void delete(long *objectId*)
Add Access Control List (ACL)	public long addACL (long *configurationId*, String *name*, String *properties*)
Update Access Control List (ACL)	void update (APIEntity *entity*) For more information, see *Updating Objects* on page 52.

DNS Zones

Add Entity for DNS Zones	long addEntity (long *parentId*, APIEntity *entity*)
Add Zone	long addZone (long *parentId*, String *absoluteName*, String *properties*)
Get Zones by Hint	APIEntity[] getZonesByHint(long *containerId*, int *start*, int *count,* String *options*)
Update Zone	void update (APIEntity *entity*) For more information, see generic update() method.
Zone Generic Methods	getEntity() void delete(long *objectId*)
Get Key Signing Key	Public String[] getKSK (long *entityId*, String *format*)

DNS Zone Templates

Add Zone Template	long addZoneTemplate (long *parentId*, String *name*, String *properties*)
Update Zone Template	void update (APIEntity *entity*) For more information, see generic update() method.
Zone Template Generic Methods	getEntity() void delete(long *objectId*)

ENUM Zones

Add ENUM Zone	`long addEnumZone ( long `*`parentId`*`, long `*`prefix`*`, String `*`properties`*` )`
Update ENUM Zone	`void update ( APIEntity `*`entity`*` )` For more information, see generic update() method.

ENUM Zone Generic Methods	`getEntity()`
	`void delete( long `*`objectId`*` )`

ENUM Numbers

Add ENUM Number	long addEnumNumber (long *parentId*, int *number*, String *properties*)
Update ENUM Number	void update (APIEntity *entity*) For more information, see generic update() method.
ENUM Number Generic Methods	getEntity() void delete(long *objectId*)

Generic Resource Records

Add Resource Record	long addResourceRecord (long *viewId*, String *absoluteName*, String *type*, String *rdata*, long *ttl*, String *properties*)
Add Entity for Resource Records	long addEntity (long *parentId*, APIEntity *entity*)
Move Resource Records	void moveResourceRecord (long *resourceRecordId*, String *destinationZone*)

NAPTR Records

Add NAPTR Record	long addNAPTRRecord (long *viewId*, String *absoluteName*, int *order*, int *preference*, String *service*, String *regexp*, String *replacement*, String *flags*, long *ttl*, String *properties*)
Update NAPTR Record	update (int *order*, int *preference*, String *service*, String *regexp*, String *replacement*, long *ttl*) For more information, see generic update() method.
NAPTR Record Generic Methods	getEntity() void delete(long *objectId*)

External Host Records

Add External Host Record	long addExternalHostRecord (long *viewId*, String *name*, String *properties*)
Update External Host Record	void update (APIEntity *entity*) For more information, see generic update() method.
External Host Record Generic Methods	getEntity() void delete(long *objectId*)

Host Records

Add Host Record	long addHostRecord (long *viewId*, String *absoluteName*, String *addresses*, long *ttl*, String *properties*)
Add Bulk Host Records	APIEntity[] addBulkHostRecord (long *viewId*, String *absoluteName*, long *ttl*, long *networkId*, String *startAddress*, int *numberOfAddresses*, String *properties*)
Get Host Record by Hint	APIEntity[] getHostRecordsByHint (int *start*, int *count*, String *options*)
Get IP Address with Host Records	APIEntity[] getNetworkLinkedProperties(long *networkId*)
Get Dependent Records	APIEntity[] getDependentRecords(long *entityId*, int *start*, int *count*)
Update Host Record	void update (String *addresses*, long *ttl*, String *comment*) For more information, see generic update() method.
Host Record Generic Methods	getEntity() void delete(long *objectId*)

Alias Records

Add Alias Record	long addAliasRecord (long *viewId*, String *absoluteName*, String *linkedRecordName*, long *ttl*, String *properties*)
Get Aliases by Hint	APIEntity[] getAliasesByHint (int *start*, int *count*, String *options*)
Update Alias Record	void update (String *linkedRecordName*, long *ttl*, String *comment*) For more information, see generic update() method.
Alias Record Generic Methods	getEntity() void delete(long *objectId*)

Text Records

Add Text Record	long addTXTRecord (long *viewId*, String *absoluteName*, String *txt*, long *ttl*, String *properties*)
Update Text Record	void update (long *ttl*, String *comment* String *txt*) For more information, see generic update() method.
Text Record Generic Methods	getEntity() void delete(long *objectId*)

HINFO Records

Add HINFO Record	long addHINFORecord (long *viewId*, String *absoluteName*, String *cpu*, String *os*, long *ttl*, String *properties*)
Update HINFO Record	void update (long *ttl*, String *comment* String *cpu*, String *os*) For more information, see generic update() method.
HINFO Record Generic Methods	getEntity() void delete(long *objectId*)

MX Records

Add MX Record	long addMXRecord (long *viewId*, String *absoluteName*, int *priority*, String *linkedRecordName*, long *ttl*, String *properties*)
Update MX Record	void update (String *linkedRecordName*, long *ttl*, int *priority*, String *comment*) For more information, see generic update() method.
MX Record Generic Methods	getEntity() void delete(long *objectId*)

SRV Records

Add SRV Record	long addSRVRecord (long *viewId*, String *absoluteName*, int *priority*, int *port*, int *weight*, String *linkedRecordName*, long *ttl*, String *properties*)
Update SRV Record	void update (String *linkedRecordName*, long *ttl*, int *priority*, int *port*, int *weight*, String *comment*) For more information, see generic update() method.
SRV Record Generic Methods	getEntity() void delete(long *objectId*)

Start of Authority Records

Add Start of Authority Record	long addStartOfAuthority (long *parentId*, String *email*, long *refresh*, long *retry*, long *expire*, long *minimum*, String *properties*)
Update Start of Authority Record	void update (String *email*, long *refresh*, long *retry*, long *expire*, long *minimum*) For more information, see generic update() method.

Start of Authority Record Generic Methods	getEntity() void delete(long *objectId*)

Generic Records

Add Generic Record	long addGenericRecord (long *viewId*, String *absoluteName*, String *type*, String *rdata*, long *ttl*, String *properties*)
Update Generic Record	void update (String *type*, String *rdata*, long *ttl*, String *comment*) For more information, see generic update() method.
Generic Record Generic Methods	getEntity() void delete(long *objectId*)

DNS Options

Add DNS Option	long addDNSDeploymentOption (long *entityId*, String *name*, String *value*, String *properties*)
Get DNS Option	APIDeploymentOption getDNSDeploymentOption (long *entityId*, String *name*, long *serverId*)
Update DNS Option	void updateDNSDeploymentOption (APIDeploymentOption *option*)
Delete DNS Option	void deleteDNSDeploymentOption (long *entityId*, String *name*, long *serverId*)

DNS Response Policies

Add Response Policy	long addResponsePolicy (long *configurationId*, String *name*, String *responsePolicyType*, long *ttl*, String *properties*)
Upload Response Policy Item	void uploadResponsePolicyItems (long *parentId*, byte[] *policyItemsData*)

Deployment options

Getting deployment options

Get Deployment Options	APIDeploymentOption[] getDeploymentOptions (long *entityId*, String *optionTypes*, long *serverId*)

TFTP

TFTP Groups

Add TFTP Group	long addTFTPGroup (long *configurationId*, String *name*, String *properties*)
Update TFTP Group	void update (APIEntity *entity*) For more information, see generic update() method.
TFTP Group Generic Methods	getEntity() void delete(long *objectId*)

TFTP Folders

Add TFTP Folder	long addTFTPFolder (long *parentId*, String *name*, String *properties*)
Update TFTP Folder	void update (APIEntity *entity*) For more information, see generic update() method.
TFTP Folder Generic Methods	getEntity() void delete(long *objectId*)

TFTP Files

Add TFTP File	long addTFTPFile (long *parentId*, String *name*, String *version*, byte[] *data*, String *properties*)
Update TFTP File	void update (String *name*, String *version*, byte[] *data*, String *description*) For more information, see generic update() method.
TFTP File Generic Methods	getEntity() void delete(long *objectId*)

Servers and Deployment

Servers

Add Server	long addServer (long *configurationId*, string *name*, string *defaultInterfaceAddress*, string *fullHostName*, string *profile*, string *properties*)
Import Server	void importServer(long *serverId*, boolean *importDns*, boolean *importDhcp*, string *properties*)
Replace Server	void replaceServer(long *serverId*, string *name*, string *defaultInterfaceAddress*, string *fullHostName*, string *password*, boolean *upgrade*, string *properties*)
Deploy Server	void deployServer (long *serverId*)
Deploy Server Configuration	void deployServerConfig (long *serverID*, String *properties*)
Deploy Server Services	void deployServerServices(long *serverId*, String *properties*)
Quick Deployment	void quickDeploy(long *zoneId*, String *properties*)
Deployment Status	int getServerDeploymentStatus(long *serverId*, String *properties*)
Server Generic Methods	getEntity() void delete(long *objectId*)

DNS and DHCP Deployment Roles

Get Servers Associated with a Deployment Role	APIEntity getServerForRole (long *roleId*)
Get Server's Associated Deployment Roles	APIDeploymentRole[] getServerDeploymentRoles (long *serverId*)
Get Deployment Roles for DNS and IP Address Space Objects	APIDeploymentRole[] getDeploymentRoles (long *entityId*)
Move Deployment Roles	moveDeploymentRoles (long *sourceServerId*, long *targetServerInterfaceId*, boolean *moveDnsRoles*, boolean *moveDhcpRoles*, String *options*)

DHCP Deployment Roles

Add DHCP Deployment Role	long addDHCPDeploymentRole (long *entityId*, long *serverInterfaceId*, String *type*, String *properties*)

Get DHCP Deployment Role	APIDeploymentRole getDHCPDeploymentRole (long *entityId*, long *serverInterfaceId*)
Update DHCP Deployment Role	void updateDHCPDeploymentRole (APIDeploymentRole *role*)
Delete DHCP Deployment Role	void deleteDHCPDeploymentRole (long *entityId*, long *serverInterfaceId*)

DNS Deployment Roles

Add DNS Deployment Role	long addDNSDeploymentRole (long *entityId*, long *serverInterfaceId*, String *type*, String *properties*)
Get DNS Deployment Role	APIDeploymentRole getDNSDeploymentRole (long *entityId*, long *serverInterfaceId*)
Get DNS Deployment Role for View	APIDeploymentRole getDNSDeploymentRoleForView (long *entityId*, long *serverInterfaceId*, long *viewId*)
Update DNS Deployment Role	void updateDNSDeploymentRole (APIDeploymentRole *role*)
Delete DNS Deployment Role	void deleteDNSDeploymentRole (long *entityId*, long *serverInterfaceId*)
Delete DNS Deployment Role for View	void deleteDNSDeploymentRoleForView (long *entityId*, long *serverInterfaceId*, long *viewId*)

TFTP Deployment Roles

Add TFTP Deployment Role	long addTFTPDeploymentRole (long *entityId*, long *serverId*, String *properties*)
Update TFTP Deployment Role	*Not supported*
TFTP Deployment Role Generic Methods	getEntity() void delete(long *objectId*)

Crossover High Availability (XHA)

Creating an XHA

Create XHA	long createXHAPair (long *configurationId*, long *activeServerId*, long *passiveServerId*, String *activeServerNewIPv4Address*, String *properties*)
Edit XHA	void editXHAPair (long *xHAServerId*, String *name*, String *properties*)

XHA Failover

Failover XHA	`void failoverXHA ( long xHAServerId )`

Breaking an XHA

Break XHA	`void breakXHAPair ( long xHAServerId, boolean breakInProteusOnly )`

Proteus Objects

Configurations

Add Configuration	`long addEntity ( long parentId, APIEntity entity )`
Update Configuration	`void update ( String name, String properties )` For more information, see generic update() method.

Configuration Generic Methods	`getEntity()`
	`void delete( long objectId )`

Groups and Users

Add Group	`long addUserGroup ( String name, String properties )`
Update Group	`void update ( APIEntity entity )` For more information, see generic update() method.

Group Generic Methods	`getEntity()`
	`void delete( long objectId )`

Add User	`long addUser ( String username, String password, String properties )`
Update User	`void update ( properties = "securityPrivilege=<value>| historyPrivilege=<value>" )` For more information, see generic update() method.

User Generic Methods	`getEntity()`
	`void delete( long objectId )`

Authenticators

Update Authenticator	`void update ( APIEntity entity )` For more information, see generic update() method.

Authenticator Generic Methods	`getEntity()`
	`void delete( long objectId )`

Access Rights

Add Access Right	`long addAccessRight ( long entityId, long userId, String value, String overrides )`
Get Access Right	`APIAccessRight getAccessRight ( long entityId, long userId )`

Get Access Rights for Entity	APIAccessRight[] getAccessRightsForEntity (long *entityId*, int *start*, int *count*)
Get Access Rights for User	APIAccessRight[] getAccessRightsForUser (long *userId*, int *start*, int *count*)
Update Access Rights	void updateAccessRight (long *entityId*, long *userId*, String *value* String *overrides*)
Delete Access Rights	void deleteAccessRight (long *entityId*, long *userId*)

Devices

Add Device	long addDevice (long *configurationId*, String *name*, long *deviceTypeId*, long *deviceSubtypeId*, String *ip4Addresses*, String *ip6Addresses*, String *properties*)
Add Device Subtype	long addDeviceSubtype (long *parentId*, String *name*, String *properties*)
Add Device Type	long addDeviceType (String *name*, String *properties*)

Object Tag Groups

Add Object Tag Group	long addTagGroup (String *name*, String *properties*)
Update Object Tag Group	void update (APIEntity *entity*) For more information, see generic update() method.
Object Tag Group Generic Methods	getEntity() void delete(long *objectId*)

Object Tags

Add Object Tag	long addTag (long *parentid*, String *name*, String *properties*)
Assign Object Tag	This method is deprecated. Using this method now returns an error message. Use the **linkEntities()** method instead. For more information, see *Link Entities* on page 50.
Remove Object Tag	This method is deprecated. Using this method now returns an error message. Use the **unlinkEntities()** method instead. For more information, see *Unlink Entities* on page 51.
Assign Object Tag	void update (APIEntity *entity*) For more information, see generic update() method.
Object Tag Generic Methods	getEntity() void delete(long *objectId*)

Database Management

Configure Replication	void configureReplication (String *standbyServer*, boolean *compressReplication*, long *replicationQueueThreshold*, long *replicationBreakThreshold*, String *properties*)

MAC Pools

Get MAC Addresses in Pool	This method is deprecated. Using this method now returns an error message. Use the getLinkedEntities() method instead. For more information, see *Get Linked Entities* on page 49.
Update MAC Pool	void update (APIEntity *entity*) For more information, see generic update() method.
MAC Pool Generic Methods	getEntity() void delete(long *objectId*)

MAC Addresses

Add MAC Address	long addMACAddress (long *configurationId*, String *macAddress*, String *properties*)
Associate MAC Address	void associateMACAddressWithPool (long *configurationId*, String *macAddress*, long *poolId*)
Deny MAC Address	void denyMACAddress (long *configurationId*, String *macAddress*)
Is Address Allocated?	boolean isAddressAllocated (long *configurationId*, String *ipAddress*, String *macAddress*)

Update MAC Address	`void update ( String name, String macpoolId )` For more information, see generic update() method.

MAC Address Generic Methods	`getEntity()`
	`void delete( long objectId )`

Get MAC Address	`APIEntity getMACAddress ( long configurationId, String macAddress )`

Workflow Change Requests

Set Workflow Level	`void setWorkflowLevel ( String level )`

Migration

Migrate a File	`void migrateFile ( String filename )`
Migration Status	`boolean isMigrationRunning ( String filename )`

Appendix A
Property Options Reference

Property options

The following tables list the available properties that can be either updatable or read-only when using the ***get***, ***add*** or ***update*** API methods. The properties marked with *read-only* cannot be updated when committing ***add*** or ***update*** methods. Refer to these tables to find what value of properties will be returned and what values can be updated.

Configuration

Object Type	Properties	Read-only/Updatable
Configuration	None	None

Views and Zones

Object Type	Properties	Read-only/Updatable
View	None	None
Zone	deployable	Both
Zone Template	None	None
EnumZone	deployable	Both
Response Policy	None	None
EnumNumber	name	Both
	data	Both

Resource Records

Object Type	Properties	Read-only/Updatable
Host Record	ttl=time-to-live value	Both
	absolutName=the FQDN for the host record	Read-only
	addresses=a list of comma-separated IP addresses (For example: 10.0.0.5,130.4.5.2)	Both
	reverseRecord	Both
Alias Record	ttl=time-to-live value	Both
	absolutName=the FQDN for the host record	Read-only
	linkedRecordName=the name of the record to which this alias will link.	Both
External Host	None	None
Generic Record	ttl = time-to-live value	Both
	absolutName = the FQDN for the host record	Read-only
	type = Resource record type (For example: A/AAAA/PTR/SRV/MX)	Read-only
	rdata = Resource record data (comma-separated values as per the record type)	Both
Host Info Record	ttl = time-to-live value	Both
	absolutName = the FQDN for the host record	Read-only
	os = a string providing operation system information	Both
	cpu = a string providing central processing unit information	Both
Mail Exchanger Record	ttl = time-to-live value	Both
	absolutName = the FQDN for the host record	Read-only
	linkedRecordName = the FQDN of the host record to which this MX record is linked	Both
	priority = specifies which mail server to send clients to first when multiple matching MX records are present. Multiple MX records with equal priority values are referred to in a round-robin fashion.	Both

Object Type	Properties	Read-only/Updatable
Naming Authority Pointer	ttl = time-to-live value	Both
	absolutName = the FQDN for the host record	Read-only
	order = specifies the order in which NAPTR records are read, if several records are present and are possible matches. The lower order value takes precedence.	Both
	preference = specifies the order in which NAPTR records are read if the order values are the same in multiple records. The lower preference value takes precedence.	Both
	service = specifies the service used for the NAPTR record	Both
	regexp = a regular expression, enclosed in double quotation marks, used to transform the client data. If a regular expression is not specified, a domain name must be specified in the replacement parameter.	Both
	replacement = specifies a domain name as an alternative to the regexp. This parameter replaces client data with a domain name	Both
	flags = an optional parameter used to set flag values for the record.	Both
Service Record	ttl = time-to-live value	Both
	absolutName = the FQDN for the host record	Read-only
	linkedRecordName = the FQDN of the host record to which this service record is linked.	Both
	port = the TCP/UDP port on which the service is available.	Both
	priority = specifies which SRV record to use when multiple matching SRV records are present. The record with the lowest value takes precedence	Both
	weight = if two matching SRV records within a zone have equal priority, the weight value is checked. If the weight value for one object is higher than the other, the record with the highest weight has its resource records returned first.	Both
Text Record	ttl = time-to-live value	Both
	absolutName = the FQDN for the host record	Read-only
	txt	Both

Admin

Object Type	Properties	Read-only/Updatable
User	userType	Read-only
	securityPrivilege	Both
	historyPrivilege	Both
	email	Both
	phoneNumber	Both
	authenticator	Both
	userAccessType	Both
UserGroup	None	None
Authenticator	None	None

Tags

Object Type	Properties	Read-only/Updatable
Tag	None	None
TagGroup	None	None

Vendor Profiles

Object Type	Properties	Read-only/Updatable
VendorProfile	identifier = the Vendor Class Identifier	Both
VendorProfileOption	optonId = DNS Vendor Option ID	Read-only
	optionType = a data type for the option	Read-only
	optionDescription = a description of the information passed by the option	Both
	displayName = display name or screen name for the option.	Both
	optionAllowMultiple = allow the option to accept multiple values.	Read-only

DNSSEC

Object Type	Properties	Read-only/Updatable
DNSSEC Signing Policies	None	None

TFTP Objects

Object Type	Properties	Read-only/Updatable
TFTPGroup	None	None
TFTPFolder	None	None

MAC Pool Objects

Object Type	Properties	Read-only/Updatable
MACAddress	address = String representing the mac address	Read-only
	macPool = Associated mac pool's name	Read-only
MACPool	None	None

Device

Object Type	Properties	Read-only/Updatable
DeviceType	None	None
DeviceSubtype	None	None
Device	deviceTypeId = Id of associated DeviceType. If Device is associated with DeviceSubType, it will list the ID of the device type of the associated DevicesSubType.	Both
	deviceSubtypeId = Id of associated DeviceSubType (This property is available only if Device is associated with DeviceSubType.)	Both
	ip4Addresses = Comma delimited list of associated IP4Addresses.	Both
	ip6Addresses = Comma delimited list of associated IP6Addresses.	Both
TSIGKey	None	None

Kerberos Realms

Object Type	Properties	Read-only/Updatable
KerberosRealm	None	None
KDC	None	None
ServicePrincipal	None	None

Server

Object Type	Properties	Read-only/Updatable
SingleServer	defaultInterfaceAddress = IP address of Server	Both
	fullHostName = Host name of Server	Both
	profile - Profile of server. Possible values are ADONIS_1000, ADONIS_750, ADONIS_500, ADONIS_250, ADONIS_XMB, ADONIS_XMB2, ADONIS_1950, ADONIS_1900, ADONIS_1200, ADONIS_800, WINDOWS_SERVER, OTHER_DNS_SERVER, PROTEUS_DDW, AFILIAS_DNS_SERVER.	Both
	• activeNodeId = active server object ID • passiveNodeId = passive server object ID • activeNodePhysicalAddress = active server IP address • passiveNodePhysicalAddress = passive server IP address **Note**: For xHA Server type only. If the server is not in xHA, this property is not available.	Read-only
	importViewName = name of the Windows view. Applicable only for Windows servers. [Needs confirmation]	Both
	enableDNS = True if DNS is enabled, else false. This property will be present only if Server is a Windows server.	Both
	enableDHCP = True if DHCP is enabled, else false. This property will be present only if Server is windows server.	Both
	readOnly = True if Windows is added in read-only mode, else false. This property will be present only if Server is a Windows server.	Both
	authenticationCredentialUsername = Username for server. This property will be present only if Server is Windows and pmm server.	Both
	authenticationCredentialPassword - User password for server. This property will be updated only for Window server and PMM server.	Write-only
	authenticationCredentialDomain = Domain for server. This property will be present only if Server is Windows and pmm server.	Both
ScheduledDeployment	None	None

IPv4Objects

Object Type	Properties	Read-only/Updatable
IP4Block	CIDR = CIDR value of the block. (if it forms a valid CIDR.)	Read-only
	name = name of the block	Both
	defaultDomains = Comma separated IDs of the default domains.	Both
	start = Start of the block. (if it does not form a valid CIDR)	Read-only
	end = End of the block. (if it does not form a valid CIDR)	Read-only
	defaultView = ID of the default View for the block.	Both
	dnsRestrictions = Comma separated IDs of the DNS zones or Views to restrict the IPv4 blocks to be used in.	Both
	allowDuplicateHost = Duplicate host names check option property. The possible values are ***Enable*** or ***Disable***.	Both
	pingBeforeAssign = Ping check option property. The possible values are ***Enable*** or ***Disable***.	Both
	inheritAllowDuplicateHost = Duplicate host names inheritance check option property. The possible values are ***True*** or ***False***. If ***True***, the AllowDuplicateHost option set at the parent object level will be used. If ***False***, the allowDuplicateHost option must be specified and the value specified will be used.	Both
	inheritPingBeforeAssign = PingBeforeAssign option inheritance check option property. The possible values are ***True*** or ***False***. If ***True***, the PingBeforeAssign option set at the parent object level will be used. If ***False***, the PingBeforeAssign option must be specified and the value specified will be used.	Both
	inheritDNSRestrictions = The possible values are ***True*** or ***False***. If ***True***, the IDs of the DNS zone or View to restrict the IPv4 blocks to be used in will be inherited from the parent object. If ***False***, the DNSRestrictions option must be specified and the value specified will be used.	Both
	inheritDefaultDomains = The possible values are ***True*** or ***False***. If ***True***, the IDs of the default domain will be inherited from the parent object. If ***False***, the DefaultDomains option must be specified and the value specified will be used.	Both
	inheritDefaultView = The possible values are ***True*** or ***False***. If ***True***, the ID of the default View for the block will be inherited from the parent object. If ***False***, the DefaultView option must be specified and the value specified will be used.	Both

Object Type	Properties	Read-only/Updatable
IP4Network	CIDR = CIDR value of the block. (if it forms a valid CIDR.)	Read-only
	template = ID of the linked template.	Read-only
	gateway = Gateway of the network.	Both
	defaultDomains = Comma separated IDs of the default domains.	Both
	defaultView = ID of the default view for the block.	Both
	dnsRestrictions = Comma separated IDs of the DNS zones or views to restrict the IPv4 networks to be used in.	Both
	allowDuplicateHost = Duplicate host names check option property. The possible values are ***Enable*** or ***Disable***.	Both
	pingBeforeAssign = Ping check option property. The possible values are ***Enable*** or ***Disable***.	Both
	inheritAllowDuplicateHost = Duplicate host names inheritance check option property. The possible values are ***True*** or ***False***. If ***True***, the AllowDuplicateHost option set at the parent object level will be used. If ***False***, the allowDuplicateHost option must be specified and the value specified will be used.	Both
	inheritPingBeforeAssign = PingBeforeAssign option inheritance check option property. The possible values are ***True*** or ***False***. If ***True***, the PingBeforeAssign option set at the parent object level will be used. If ***False***, the PingBeforeAssign option must be specified and the value specified will be used.	Both
	inheritDNSRestrictions = The possible values are ***True*** or ***False***. If ***True***, the IDs of the DNS zone or View to restrict the IPv4 blocks to be used in will be inherited from the parent object. If ***False***, the DNSRestrictions option must be specified and the value specified will be used.	Both
	inheritDefaultDomains = The possible values are ***True*** or ***False***. If ***True***, the IDs of the default domain will be inherited from the parent object. If ***False***, the DefaultDomains option must be specified and the value specified will be used.	Both
	inheritDefaultView = The possible values are ***True*** or ***False***. If ***True***, the ID of the default View for the block will be inherited from the parent object. If ***False***, the DefaultView option must be specified and the value specified will be used.	Both

Object Type	Properties	Read-only/Updatable
	assignDefaultGateway = The possible values are ***True*** or ***False***. If ***True***, a gateway will be created by using the default gateway value which is the first IP address in the network. If ***False***, no gateway will be created.	Both
	overwriteConflicts = The possible values are ***True*** or ***False***. If ***True***, any conflicts within the split IPv4 network will be removed.	Both
IP4Address	address = Address string.	Read-only
	state = state of the address. For possible values, refer to ***IP address states*** on page 265.	Read-only
	macAddress = MAC address of the IP4Address.	Both
IP4DHCPRange	start = Start of the range.	Both
	end = End of the range.	Both
	offset = IPv4 address from which the range should begin.	Both
	size = the size of the range to be created.	Both
	defineRangeBy = the possible values are OFFSET_AND_SIZE and OFFSET_AND_PERCENTAGE.	Both
IP4NetworkTemplate	gateway = gateway of the network.	Both
	reservedAddress = the list of reserved addresses being set on the network template.	Both

IPv6Objects

Object Type	Properties	Read-only/Updatable
IP6Network	prefix = Prefix of the Network	Read-only
IP6Address	address = Address string	Read-only
	macAddress = MAC address of the IP6Address	Read-only
	state = State of the address. For possible values, refer to ***IP address states*** on page 265.	Read-only
IP6DHCPRange	start = Start of the range	Read-only
	end = End of the range	Read-only

DeploymentRoles

Object Type	Properties	Read-only/Updatable
DNSDeploymentRole	view	Both
	zoneTransServerInterface	Both
	inherited	Read-only
DHCPDeploymentRole	inherited	Read-only

Access right

Object Type	Properties	Read-only/Updatable
Access right	workflowLevel = valid values are None, Recommend, and Approve.	Both
	deploymentAllowed = true to perform a full deployment of data to a managed server, else false.	Both
	quickDeploymentAllowed = ture to instantly deploy changed DNS resource records, else false.	Both

IP address states

The following tables list the available values of the *state* parameter of the IP address.

IPv4

State	Description
UNALLOCATED	Available and unassigned IP address for DNS or DHCP.
STATIC	Statically assigned hosts and only used for DNS purposes.
DHCP_ALLOCATED	Dynamically assigned through DHCP to the given MAC address.
DHCP_FREE	Dynamically assigned through DHCP but are now in a free or unallocated state.
DHCP_RESERVED	Represent DHCP reservations, and may yet be assigned to a host. These addresses can be inside or outside of a DHCP range.
DHCP_LEASED	Used in a DHCP lease.

State	Description
RESERVED	Reserved for future use. While reserved, the address cannot be assigned a DNS hot name and cannot be deployed to DHCP.
GATEWAY	Network gateway (router) addresses.

IPv6

State	Description
UNALLOCATED	Available and unassigned IP address for DNS or DHCP.
STATIC	Statically assigned hosts and only used for DNS purposes.
DHCP_ALLOCATED	Dynamically assigned through DHCP to the given MAC address.
DHCP_FREE	Dynamically assigned through DHCP but are now in a free or unallocated state.
DHCP_RESERVED	Represent DHCP reservations, and may yet be assigned to a host. These addresses can be inside or outside of a DHCP range.
DHCP_LEASED	Used in a DHCP lease.
RESERVED	Reserved for future use. While reserved, the address cannot be assigned a DNS hot name and cannot be deployed to DHCP.
GATEWAY	Network gateway (router) addresses.

For safe operating procedures, ensure compliance with the guidelines below.

CAUTION

Do not remove the cover from the appliance. The cover is to be removed only by qualified personnel. There are no serviceable parts provided inside.

CAUTION

To prevent the unit from overheating, never install the appliance in an enclosed rack or room that is not properly ventilated or cooled. For proper air flow, keep the front and back sides of the appliance clear of obstructions and away from the exhaust of other equipment.

CAUTION

Before servicing, power off the appliance by using the rear panel switch. If the appliance does not have an On/Off switch, then unplug the power cord.

CAUTION

Electrostatic Discharge (ESD) precautions are required before handling the appliance. Wear a wrist strap with an appropriate ground connection.

CAUTION

There is danger of an explosion if the battery is replaced incorrectly. Replace only with the same or equivalent type recommended by the appliance manufacturer. Contact technical support if you need to replace a battery.

CAUTION

Failure to properly ground the appliance, either by circumventing the 3-wire grounding-type plug or by using a power outlet that is improperly grounded, can create a potentially hazardous electrical situation.

FCC Notice

This device complies with part 15 of the FCC Rules. Operation is subject to the following conditions:

- This device may not cause harmful interference.
- This device must accept any interference received, including interference that may cause undesired operation.

No (Telecommunications Network Voltage) TNV-connected PCBs shall be installed.

Warning

This is a Class A product. In a domestic environment, the product may cause radio interference in which case the user may be required to take adequate measures.

BlueCat Networks (USA), Inc.

www.bluecatnetworks.com

Toll Free: 1.866.895.6931

Document #: Proteus API_4.0.6

Published in Canada

Date: June 2014