



Cisco Prime Network Registrar 9.0 DHCP User Guide

First Published: 2016-12-22

Last Modified: 2016-12-22

Americas Headquarters

Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <http://www.cisco.com/go/trademarks>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

© 2016 Cisco Systems, Inc. All rights reserved.



CONTENTS

CHAPTER 1

Introduction to Dynamic Host Configuration 1

How DHCP Works 1

Related Topics 1

Sample DHCP User 2

Typical DHCP Administration 2

Leases 3

Scopes and Policies 3

Links and Prefixes 4

Related Topics 4

Cisco Prime Network Registrar DHCP Implementations 5

Related Topics 5

Virtual Private Networks 5

Prefix Delegation 6

DNS Update 7

Related Topics 7

Effect on DNS of Obtaining Leases 7

Effect on DNS of Reacquiring Leases 8

Effect on DNS of Releasing Leases 8

DHCP Failover 9

Allocating Addresses Through Failover 9

Client-Classes 10

Related Topics 10

DHCP Processing Without Client-Classes 11

DHCP Processing with Client-Classes 11

Defining Scopes for Client-Classes 12

Choosing Networks and Scopes 12

CHAPTER 2	Managing DHCP Server	15
	Configuring DHCP Servers	15
	General Configuration Guidelines	15
	Configuring DHCP Server Interfaces	16
	Local Advanced Web UI	16
	CLI Commands	16
	Defining Advanced Server Attributes	16
	Related Topics	17
	Setting Advanced DHCP Server Attributes	17
	Local Basic or Advanced Web UI	21
	CLI Commands	21
	Deferring Lease Extensions	22
	Setting DHCP Forwarding	23
	Editing DHCPv6 Server Attributes	24
	Local Basic or Advanced Web UI	24
	CLI Commands	24
	Integrating Windows System Management Servers	25
	Using Extensions to Affect DHCP Server Behavior	26
	Related Topics	27
	Writing Extensions	27
	Preventing Chatty Clients by Using an Extension	28
	Tuning the DHCP Server	31
	Listing Related Servers for DHCP - Failover, DNS, LDAP, and TCP Listener Servers	34
	Local Web UI	35
	CLI Commands	43
	Configuring Virtual Private Networks	43
	Related Topics	43
	Configuring Virtual Private Networks Using DHCP	43
	Related Topics	44
	Typical Virtual Private Networks	44
	Creating and Editing Virtual Private Networks	45
	VPN Usage	46
	Configuring Subnet Allocation	48

Related Topics	48
Configuring DHCP Subnet Allocation	49
VPN and Subnet Allocation Tuning Parameters	50
Configuring BOOTP	50
Related Topics	51
About BOOTP	51
Enabling BOOTP for Scopes	52
Moving or Decommissioning BOOTP Clients	52
Using Dynamic BOOTP	52
BOOTP Relay	53

CHAPTER 3

Managing DHCP Failover	55
How DHCP Failover Works	55
DHCP Simple Failover	56
DHCPv6 Failover	56
Setting Up Failover Server Pairs	57
Related Topics	57
Adding Failover Pairs	57
58	
CLI Commands	60
Related Topics	60
Synchronizing Failover Pairs	61
CLI Commands	63
Failover Checklist	63
Configuring Failover Parameters Based on Your Scenario	64
Setting Backup Percentages	64
Related Topics	65
Setting the Maximum Client Lead Time	65
Using the Failover Safe Period to Move Servers into PARTNER-DOWN State	67
Setting DHCP Request and Response Packet Buffers	69
Setting Load Balancing	69
Related Topics	70
Configuring Load Balancing	70
Recovering from a DHCP Failover	70

Confirming Failover	70
Related Topics	71
Monitoring DHCP Failover	71
Failover States and Transitions	71
State Transitions During Integration	73
Setting Advanced Failover Attributes	75
Setting Backup Allocation Boundaries	75
DHCPLEASEQUERY and Failover	75
Maintaining Failover Server Pair	76
Changing Failover Pair Names	76
Restarting the Failover Servers	76
Related Topics	76
Recovering Failover Configuration	76
Using PARTNER-DOWN State to Allow a Failover Server to Operate for Extended Periods without Its Failover Partner	77
Reintegrating the Returning Failover Partner	77
Restoring a Standalone DHCP Failover Server - Tutorial	78
Related Topics	79
Background	79
Repair Procedure	79
Reversing the Failover Role on Backup Server	80
Starting with Server A Powered Off	80
Starting with Server A Powered On and DHCP Server Stopped	81
Starting with Server A Replaced	82
Transferring Current Lease State to Server A	82
Repairing Partners to Their Original Roles	83
Changing Failover Server Roles	84
Related Topics	84
Establishing Failover Using Standalone Server as Main	84
Replacing Servers Having Defective Storage	85
Removing Backup Servers and Halting Failover Operation	86
Adding Main Servers to Existing Backup Servers	86
Configuring Failover on Multiple Interface Hosts	86
Troubleshooting Failover	86

Related Topics	87
Monitoring Failover Operations	87
Detecting and Handling Network Failures	87
Things to Avoid When Troubleshooting Issues Related to Failover	88
Supporting BOOTP Clients in Failover	88
Related Topics	88
Static BOOTP	88
Dynamic BOOTP	89
Configuring BOOTP Relays	89
BOOTP Backup Percentage	89

CHAPTER 4
Managing Address Space 91

Address Block Administrator Role	91
Related Topics	91
Required Permissions	91
Role Functions	92
Address Blocks and Subnets	92
Related Topics	93
Subnet Allocation and DHCP Address Blocks	93
Knowing When to Add Address Blocks	94
Adding Address Blocks	95
Configuring Private Networks in VPN for a Tenant	96
Regional Advanced Web UI	96
CLI Commands	96
Delegating Address Blocks	97
Creating Reverse Zones from Subnets	97
Related Topics	97
Reclaiming Subnets	97
Adding Children to Address Blocks	98
Adding Address Ranges to Subnets	99
Pulling and Pushing	99
Pulling Replica Address Space from Local Clusters	99
Pushing Subnets to Local DHCP Servers and Routers	100
Viewing Address Space	100

- Viewing Address Utilization for Address Blocks, Subnets, and Scopes 101
- Viewing Address Blocks, Subnets, and Address Types 102
- Viewing IPv6 Address Space 103
- Viewing Address Utilization for Prefixes 103
- Generating Subnet Utilization History Reports 105
 - Related Topics 106
 - Enabling Subnet Utilization History Collection at the Local Cluster 106
 - Querying Subnet Utilization History Data 106
 - Regional Web UI 106
 - Trimming and Compacting Subnet Utilization History Data 107
 - Regional Web UI 108
 - Viewing Subnet Utilization History Data 108

CHAPTER 5

Managing Scopes, Prefixes, Links, and Networks 111

- Managing Scopes 111
 - Related Topics 111
 - Creating Scopes 112
 - Local Basic Web UI 112
 - Local Advanced Web UI 112
 - Configuring Multiple Scopes 113
 - Related Topics 113
 - Configuring Multiple Scopes for Round-Robin Address Allocation 113
 - Configuring Multiple Scopes Using Allocation Priority 114
 - Editing Scopes 118
 - Local Advanced Web UI 119
 - CLI Commands 119
 - Staged and Synchronous Mode 119
 - Local Basic or Advanced Web UI 120
 - CLI Commands 120
 - Getting Scope Counts on the Server 120
 - CLI Commands 120
 - Configuring Embedded Policies for Scopes 121
 - Local Advanced Web UI 121
 - CLI Commands 121

Configuring Multiple Subnets on a Network	121
Local Advanced Web UI	122
CLI Commands	122
Enabling and Disabling BOOTP for Scopes	122
Local Advanced Web UI	122
CLI Commands	123
Setting Scopes to Renew-Only	123
Local Advanced Web UI	123
CLI Commands	123
Setting Free Address SNMP Traps on Scopes	123
Local Advanced Web UI	123
CLI Commands	124
Disabling DHCP for Scopes	124
Local Advanced Web UI	124
CLI Commands	124
Deactivating Scopes	124
Local Advanced Web UI	124
CLI Commands	124
Removing Scopes	125
Removing Scopes if Not Reusing Addresses	125
Removing Scopes if Reusing Addresses	125
DHCPv6 Addresses	126
IPv6 Addressing	126
Determining Links and Prefixes	127
Generating Addresses	127
Generating Delegated Prefixes	128
Prefix Stability	128
CMTS Prefix Stability	129
Universal Prefix Stability	129
Prefix Allocation Groups	130
Configuring Prefixes and Links	130
Creating and Editing Prefixes	131
Local Advanced and Regional Web UI	132
CLI Commands	134

Creating and Editing Links	134
Local Advanced and Regional Web UI	135
CLI Commands	135
Managing DHCP Networks	136
Related Topics	136
Listing Networks	136
Editing Networks	136
Local Basic or Advanced Web UI	137
<hr/>	
CHAPTER 6	Managing Scopes, Prefixes, and Link Templates
Creating and Applying Scope Templates	139
Local Advanced and Regional Web UI	139
Related Topics	139
CLI Commands	140
Additional Scope Template Attributes	140
Editing Scope Templates	140
Applying Scope Templates to Scopes	140
Local Advanced Web UI	140
CLI Commands	141
Cloning a Scope Template	141
Creating and Editing Prefix Templates	141
Local Advanced and Regional Web UI	143
CLI Commands	144
Creating and Editing Link Templates	144
Local Advanced and Regional Web UI	145
CLI Commands	145
Using Expressions in Templates	146
Using Expressions in Scope Templates	146
Local Advanced and Regional Web UI	149
CLI Commands	150
Scope Name Expression Example	150
Range Expression Example	150
Embedded Policy Option Expression Example	151
Using Expressions in Prefix Templates	151

Using Expressions in Link Templates 155

CHAPTER 7

Managing Policies and Options 161

Configuring DHCP Policies 161

Related Topics 161

Configuring DHCPv6 Policies 162

Reconfigure Support (DHCPv6) 162

Types of Policies 163

Policy Hierarchy 164

DHCPv4 Policy Hierarchy 164

DHCPv6 Policy Hierarchy 165

Creating and Applying DHCP Policies 166

Local Basic or Advanced and Regional Web UI 167

CLI Commands 168

Related Topics 168

Cloning a Policy 168

Setting DHCP Options and Attributes for Policies 168

Related Topics 169

Adding Option Values 169

Local Basic or Advanced and Regional Web UI 169

CLI Commands 169

Adding Complex Values for Suboptions 170

Creating and Editing Embedded Policies 170

Local Advanced Web and Regional UI 170

CLI Commands 171

Creating DHCP Option Definition Sets and Option Definitions 171

Related Topics 171

Using Standard Option Definition Sets 172

Local Advanced and Regional Web UI 172

CLI Commands 172

Creating Custom Option Definitions 173

Creating Vendor-Specific Option Definitions 173

Local Advanced and Regional Web UI 174

Local Advanced and Regional Web UI 174

- Example: Creating Vendor Option Set for Cisco AP Devices 176
- Example: Creating Vendor Option Set for SunRay Devices 177
- Example: Creating Option Set for Cisco 79xx IPPhones 178
- Setting Option Values for Policies 179
 - Local Advanced and Regional Web UI 179
 - CLI Commands 179
- Setting DHCPv6 Options 179
 - Local Advanced Web UI 179
 - CLI Commands 180
- Option Definition Data Types and Repeat Counts 180
- Adding Suboption Definitions 181
- Option Definition Set 181
 - Importing and Exporting Option Definition Sets 181
 - Pushing Option Definition Sets to Local Clusters 182
 - Pulling Option Definition Sets from Replica Data 182

CHAPTER 8

- Managing Leases 185**
 - Lease States 185
 - IPv4 Lease States 185
 - IPv6 Lease States 186
 - Guidelines for Lease Times 186
 - Restricting Lease Dates 187
 - DHCPv6 Clients and Leases 189
 - Related Topics 189
 - DHCPv6 Bindings 189
 - Lease Affinity 190
 - Lease Life Cycle 190
 - Configuring Leases in Scopes 191
 - Viewing Leases 191
 - Local Basic Web UI 191
 - Local Advanced Web UI 191
 - CLI Commands 192
 - Importing and Exporting Lease Data 192
 - Import Prerequisites 192

Import and Export Commands	192
Lease Times in Import Files	193
Pinging Hosts Before Offering Addresses	194
Deactivating Leases	195
Local Basic or Advance Web UI	195
CLI Commands	195
Excluding Leases from Ranges	195
Local Basic Web UI	195
Local Advanced Web UI	196
CLI Commands	196
Removing Orphaned Leases	196
Searching Server-Wide for Leases	197
Local Advanced Web UI	197
CLI Commands	199
Using Client Reservations	199
Local Advanced Web UI	201
Differences Between Client Reservations And Lease Reservations	202
Creating Lease Reservations	202
DHCPv4 Reservations	203
Local Basic Web UI	203
Local Advanced Web UI	203
CLI Commands	203
DHCPv6 Lease Reservations	204
Local Advanced Web UI	205
CLI Commands	206
Setting Advanced Lease and Reservation Properties	207
Reserving Currently Leased Addresses	207
Local Advanced Web UI	207
Example of Reserving an Existing Lease	207
Unreserving Leases	208
Local Advanced Web UI	209
CLI Commands	209
Extending Reservations to Non-MAC Addresses	209
Overriding Client IDs	209

Local Advanced Web UI	210
CLI Commands	210
Reservation Override Example	210
Reconfiguring IPv6 Leases	211
CLI Commands	211
Forcing Lease Availability	211
Local Advanced Web UI	211
CLI Commands	212
Inhibiting Lease Renewals	212
Local Advanced Web UI	213
Moving Leases Between Servers	213
Handling Leases Marked as Unavailable	214
Setting Timeouts for Unavailable Leases	215
Querying Leases	216
Related Topics	216
Leasequery Implementations	216
Pre-RFC Leasequery for DHCPv4	217
RFC 4388 Leasequery for DHCPv4	218
Leasequery for DHCPv6	218
Leasequery Statistics	219
Leasequery Example	220
Difference between TCP bulk leasequery and UDP leasequery	222
Running Address and Lease Reports	223
Running Address Usage Reports	223
Local Advanced Web UI	223
CLI Commands	223
Running IP Lease Histories	223
Enabling Lease History Recording at the Local Cluster	224
Local Advanced Web UI	224
CLI Commands	224
Querying IP Lease History	224
Local and Regional Advanced Web UI	225
Using the iphist Utility	226
Trimming Lease History Data	228

Regional Web UI	229
Running Lease Utilization Reports	229
Local Advanced Web UI	229
CLI Commands	229
Receiving Lease Notification	229
Related Topics	230
Running Lease Notification Automatically in Linux	230
Running Lease Notification Automatically in Windows	231
Specifying Configuration Files for Lease Notification	231
Dynamic Lease Notification	231
Using Dynamic Lease Notification	231
Sample Lease Notification Client	232
Requirements for Sample Java Client	235
Local Basic or Advanced Web UI	236
CLI Command	237
DHCP Listener Configuration	237
Local Advanced Web UI	237
CLI Commands	237
Lease History Database Compression Utility	238
General Comments on Running <code>cnr_leasehist_compress</code>	239
Running Compression on Linux	240
Running Compression on Windows	243

CHAPTER 9
Managing DNS Update 245

DNS Update Process	245
Special DNS Update Considerations	246
DNS Updates for DHCPv6	246
DNS Updates for Non-Temporary Stateful Addresses	246
DNS Updates for Delegated Prefixes	247
Related Topics	247
DHCPv6 Upgrade Considerations	247
Generating Synthetic Names in DHCPv4 and DHCPv6	248
Determining Reverse Zones for DNS Updates	248
Using the Client FQDN	249

Configuring Access Control Lists and Transaction Security	249
Related Topics	250
Assigning ACLs on DNS Caching Servers or Zones	250
Local Advanced Web UI	250
CLI Commands	250
Configuring Zones for ACLs	251
Transaction Security	251
Related Topics	251
Creating TSIG Keys	251
Local Advanced Web UI	251
CLI Commands	252
Generating Keys	252
Considerations for Managing Keys	253
Adding Supporting TSIG Attributes	253
GSS-TSIG	253
Creating DNS Update Configurations	256
Local Advanced and Regional Web UI	256
CLI Commands	258
Related Topics	258
Configuring DNS Update Policies	258
Related Topics	258
Compatibility with Cisco Network Registrar Releases	258
Creating and Editing Update Policies	259
Local Advanced Web UI	259
CLI Commands	259
Defining and Applying Rules for Update Policies	259
Related Topics	259
Defining Rules for Named Update Policies	259
Local Advanced Web UI	259
CLI Commands	261
Applying Update Policies to Zones	262
CLI Commands	262
Creating DNS Update Maps	263
Local and Regional Web UI	263

CLI Commands	263
Confirming Dynamic Records	264
Local Advanced Web UI	264
CLI Commands	264
Scavenging Dynamic Records	264
Local Advanced Web UI	265
CLI Commands	266
Transitioning to DHCID RR for DHCPv4	266
Local Advanced and Regional Web UI	267
Configuring DNS Update for Windows Clients	267
Related Topics	267
Client DNS Updates	268
Dual Zone Updates for Windows Clients	270
DNS Update Settings in Windows Clients	270
Windows Client Settings in DHCP Servers	270
SRV Records and DNS Updates	271
Issues Related to Windows Environments	273
Example: Output Showing Invisible Dynamically Created RRs	276
Frequently Asked Questions About Windows Integration	276
Configuring GSS-TSIG	279
Cisco Prime Network Registrar DNS Configuration to integrate with AD	279
Bring Cisco Prime Network Registrar DNS and AD under the same domain in the windows environment:	279
Integrating the DNS server to AD-KDC	279
Primary DNS Server on Linux Integrated to MIT-KDC	281
Troubleshooting GSS-TSIG Configuration	281
Troubleshooting DNS Update	282

CHAPTER 10**Managing Client-Classes and Clients 283**

Configuring Client-Classes	283
Related Topics	283
Client-Class Process	284
Defining Client-Classes	284
Local Basic Web UI	284

Local Advanced Web UI	285
CLI Commands	285
Configuring DHCPv6 Client-Classes	286
Local Advanced Web UI	286
CLI Commands	286
Setting Selection Tags on Scopes and Prefixes	286
Local Basic or Advanced Web UI	287
CLI Commands	287
Defining Client-Class Hostname Properties	287
Related Topics	288
Editing Client-Classes and Their Embedded Policies	288
Local Advanced Web UI	288
CLI Commands	288
Processing Client Data Including External Sources	289
Related Topics	289
Processing Order to Determine Client-Classes	289
Processing Order to Determine Selection Tags	290
Troubleshooting Client-Classes	290
Configuring Clients	291
Local Basic or Advanced Web UI	291
CLI Commands	292
Related Topics	292
Editing Clients and Their Embedded Policies	293
Local Basic or Advanced Web UI	293
CLI Commands	293
Configuring DHCPv6 Clients	293
Local Advanced Web UI	293
CLI Commands	294
Setting Windows Client Properties	294
Settings in Windows Clients	294
Settings in DHCP Servers	294
Allocating Provisional Addresses	294
Provisional Addresses for Unknown Clients	294
Using One-Shot Action	295

Skipping Client Entries for Client-Classing	295
Limiting Client Authentication	295
Setting Client Caching Parameters	296
Subscriber Limitation Using Option 82	297
Related Topics	297
General Approach to Subscriber Limitation	297
Typical Limitation Scenario	298
Calculating Client-Classes and Creating Keys	298
Client-Class Lookup Expression Processing	298
Limitation Processing	299
Expression Processing for Subscriber Limitation	299
Configuring Option 82 Limitation	299
Lease Renewal Processing for Option 82 Limitation	300
Administering Option 82 Limitation	300
Troubleshooting Option 82 Limitation	301
Expression Examples	301
Configuring Cisco Prime Network Registrar to Use LDAP	301
Related Topics	301
About LDAP Directory Servers	302
Adding and Editing LDAP Remote Servers	302
Local Advanced Web UI	302
CLI Commands	302
Configuring DHCP Client Queries in LDAP	303
Configuring DHCP-Server-to-LDAP Client Queries	303
Unprovisioning Client Entries	305
Configuring Embedded Policies in LDAP	305
Configuring DHCP LDAP Update and Create Services	306
Related Topics	307
Lease State Attributes	307
Configuring DHCP to Write Lease States to LDAP	308
Storing Lease State Data as Part of Existing Entries	308
Storing Lease State Data Independently	309
Using LDAP Updates	309
Configuring LDAP State Updates	309

Option 1: Using the update-search-path Option	309
Option 2: Using the dn-format Option	310
Configuring LDAP Entry Creation	311
Troubleshooting LDAP	312
Related Topics	312
LDAP Connection Optimization	312
Recommended Values for LDAP	313

CHAPTER 11
Using Expressions 315

Using Expressions	315
Entering Expressions	316
Creating Expressions	317
Expression Syntax	318
Expression Datatypes	319
Literals in Expressions	319
Expressions Return Typed Values	320
Expressions Can Fail	320
Datatype Conversions	320
Expression Functions	322
+, -, *, /, %	322
and	322
as-blob	323
as-sint	323
as-string	323
as-uint	324
ash	324
bit	325
bit-not	325
byte	326
comment	326
concat	326
datatype	327
dotimes	327
environmentdictionary	328

equal, equali	328
error	329
if	329
ip-string	330
ip6-string	330
is-string	330
length	331
let	331
log	332
mask-blob	332
mask-int	332
not	333
null	333
or, pick-first-value	333
progn, return-last	334
regex	334
request	335
request dump	336
request option	337
requestdictionary	338
response	338
response dump	339
response option	339
responsedictionary	339
search	339
setq	340
starts-with	340
substring	340
synthesize-host-name	341
to-blob	341
to-ip, to-ip6	342
to-lower	342
to-sint	342
to-string	343

to-uint	343
translate	344
try	344
validate-host-name	345
Expression Examples	345
Related Topics	346
Limitation Example 1: DOCSIS Cable Modem	346
Limitation Example 2: Extended DOCSIS Cable Modem	347
Limitation Example 3: DSL over Asynchronous Transfer Mode	348
Debugging Expressions	349

CHAPTER 12

Using Extension Points	351
Using Extensions	351
Related Topics	352
Creating, Editing, and Attaching Extensions	352
Local Advanced Web UI	352
CLI Command	352
Related Topics	353
Determining Tasks	353
Deciding on Approaches	353
Choosing Extension Languages	354
Language-Independent API	354
Related Topics	354
Routine Signature	354
Dictionaries	355
Utility Methods in Dictionaries	355
Configuration Errors	355
Communicating with External Servers	356
Recognizing Extensions	356
Multiple Extension Considerations	357
Tel Extensions	357
Related Topics	357
Tel Application Program Interface	358
Dealing with Tel errors	358

Dealing with Tel errors	358
Configuring Tel Extensions	359
Handling Boolean Variables in Tel	359
Init-Entry Extension Point in Tel	359
C/C++ Extensions	359
Related Topics	359
C/C++ API	360
Using Types in C/C++	360
Building C/C++ Extensions	360
Using Thread-Safe Extensions in C/C++	360
Configuring C/C++ Extensions	361
Debugging C/C++ Extensions	361
Related Topics	362
Pointers into DHCP Server Memory in C/C++	362
Init-Entry Entry Point in C/C++	362
DHCP Request Processing Using Extensions	362
Related Topics	364
Enabling DHCPv6 Extensions	364
Receiving Packets	365
Decoding Packets	365
Determining Client-Classes	365
Modifying Client-Classes	365
Processing Client-Classes	366
Building Response Containers	366
Determining Networks and Links	366
Finding Leases	367
Serializing Lease Requests	367
Determining Lease Acceptability	368
DHCPv6 Leasing	369
Related Topics	369
DHCPv6 Prefix Usability	369
DHCPv6 Lease Usability	370
DHCPv6 Lease Allocation	370
Gathering Response Packet Data	370

Encoding Response Packets	371
Updating Stable Storage	371
Sending Packets	371
Processing DNS Requests	371
Tracing Lease State Changes	371
Controlling Active Leasequery Notifications	372
Extension Dictionaries	373
Related Topics	374
Environment Dictionary	374
Related Topics	375
General Environment Dictionary Data Items	375
Initial Environment Dictionary	376
Request and Response Dictionaries	376
Related Topics	377
Decoded DHCP Packet Data Items	377
Using Parameter List Option	378
Extension Point Descriptions	378
Related Topics	379
init-entry	379
pre-packet-decode	380
post-packet-decode	381
Related Topics	381
Extension Description	381
Overriding Client Identifiers	382
post-class-lookup	382
pre-client-lookup	383
Related Topics	383
Environment Dictionary for pre-client-lookup	383
post-client-lookup	385
Environment Dictionary for post-client-lookup	385
generate-lease	386
check-lease-acceptable	387
lease-state-change	388
Related Topics	388

Environment Dictionary for lease-state-change	388
pre-packet-encode	388
post-packet-encode	389
pre-dns-add-forward	389
post-send-packet	389
environment-destructor	390

CHAPTER 13
DHCP Server Status Dashboard 391

Opening the Dashboard	391
Display Types	392
General Status Indicators	392
Graphic Indicators for Levels of Alert	393
Magnifying and Converting Charts	393
Legends	393
Tables	393
Line Charts	394
Stacked Area Charts	394
Other Chart Types	395
Getting Help for the Dashboard Elements	395
Customizing the Display	395
Refreshing Displays	396
Setting the Polling Interval	396
Displaying Charts as Tables	397
Exporting to CSV Format	397
Displaying or Hiding Chart Legends	397
Selecting Dashboard Elements to Include	397
Configuring Server Chart Types	398
DHCP Metrics	399
DHCP Server Request Activity	399
How to Interpret the Data	400
Troubleshooting Based on the Results	400
DHCP Server Response Activity	400
How to Interpret the Data	400
Troubleshooting Based on the Results	401

DHCP Buffer Capacity	401
How to Interpret the Data	401
Troubleshooting Based on the Results	401
DHCP Response Latency	401
How to Interpret the Data	402
Troubleshooting Based on the Results	402
DHCP DNS Updates	402
How to Interpret the Data	402
Troubleshooting Based on the Results	402
DHCP Address Current Utilization	402
How to Interpret the Data	403
Troubleshooting Based on the Results	403
DHCP Failover Status	403
How to Interpret the Data	403
Troubleshooting Based on the Results	404
DHCP General Indicators	404
How to Interpret the Data	404
Troubleshooting Based on the Results	404
DHCP Server Lease Data	404
<hr/>	
APPENDIX A	DHCP Options 407
	Option Descriptions 407
	RFC 1497 Vendor Extensions 407
	IP Layer Parameters Per Host 409
	IP Layer Parameters Per Interface 410
	Link Layer Parameters Per Interface 411
	TCP Parameters 411
	Application and Service Parameters 411
	DHCPv4 Extension Options 417
	Microsoft Client Options 419
	DHCPv6 Options 420
	Option Tables 428
	Options by Number 428
	Options by Cisco Prime Network Registrar Name 435

Option Validation Types 445

APPENDIX B**DHCP Extension Dictionary 449**

Extension Dictionary Entries 449

Decoded DHCP Packet Data Items 449

Request Dictionary 461

Response Dictionary 469

Extension Dictionary API 482

Tel Attribute Dictionary API 482

Tel Request and Response Dictionary Methods 483

Tel Environment Dictionary Methods 486

DEX Attribute Dictionary API 487

DEX Request and Response Dictionary Methods 488

DEX Environment Dictionary Methods 492

Handling Objects and Options 499

Using Object and Option Handling Methods 499

Options and Suboptions in C/C++ 499

Examples of Option and Object Method Calls 501

Handling Vendor Class Option Data 501

Handling Object Data 501



CHAPTER 1

Introduction to Dynamic Host Configuration

All hosts seeking Internet access must have an IP address. As Internet administrator, you must perform the following for every new user and for every user whose computer was moved to another subnet:

1. Choose a legal IP address.
2. Assign the address to the individual device.
3. Define device configuration parameters.
4. Update the DNS database, mapping the device name to the IP address.

These activities are time consuming and error prone, hence the Dynamic Host Configuration Protocol (DHCP). DHCP frees you from the burden of individually assigning IP addresses. It was designed by the Internet Engineering Task Force (IETF) to reduce the amount of configuration required when using TCP/IP. DHCP allocates IP addresses to hosts. It also provides all the parameters that hosts require to operate and exchange information on the Internet network to which they are attached.

DHCP localizes TCP/IP configuration information. It also manages allocating TCP/IP configuration data by automatically assigning IP addresses to systems configured to use DHCP. Thus, you can ensure that hosts have Internet access without having to configure each host individually.

This chapter contains the following sections:

- [How DHCP Works, on page 1](#)
- [Links and Prefixes, on page 4](#)
- [Cisco Prime Network Registrar DHCP Implementations, on page 5](#)
- [Prefix Delegation, on page 6](#)
- [DNS Update, on page 7](#)
- [DHCP Failover, on page 9](#)
- [Client-Classes, on page 10](#)

How DHCP Works

DHCP makes dynamic address allocation possible by shifting device configuration to global address pools at the server level. DHCP is based on a client/server model. The client software runs on the device and the server software runs on the DHCP server.

Related Topics

[Sample DHCP User, on page 2](#)

[Typical DHCP Administration, on page 2](#)

[Leases, on page 3](#)

[Scopes and Policies, on page 3](#)

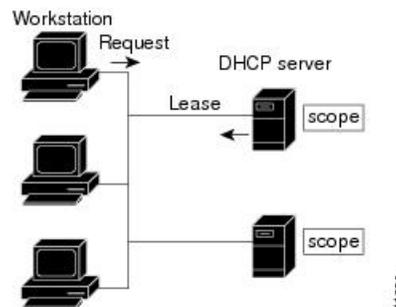
Sample DHCP User

After Beth's workstation (bethpc) is configured with DHCP, these actions occur when she first starts up:

1. Her pc automatically requests an IP address from a DHCP server on the network.
2. The DHCP server offers her a lease that is an IP address with the configuration data necessary to use the Internet. Nobody else uses the leased address, and it is valid only for her pc.
3. Before the address lease expires, bethpc renews it, thereby extending the expiration time. It continues to use the lease right up to its expiration or if it cannot reach the server.
4. If Beth relocates to another department and her pc moves to a different subnet, her current address expires and becomes available for others. When Beth starts her pc at its new location, it leases an address from an appropriate DHCP server on the subnet (see the image below).

As long as the DHCP server has the correct configuration data, none of the workstations or servers using DHCP will ever be configured incorrectly. Therefore, there is less chance of incurring network problems from incorrectly configured devices and servers that are difficult to trace.

Figure 1: Hosts Request an IP Address



The example shows the DHCP protocol with a set of DHCP servers that provide addresses on different subnets. To further simplify the administration of address pools, network routers are often configured as DHCP relay agents to forward client messages to a central DHCP server. This server is configured with address pools for a group of subnets.

Typical DHCP Administration

To use DHCP, you must have at least one DHCP server on the network. After you install the server:

- Define a scope of IP addresses that the DHCP server can offer to DHCP clients. You no longer need to keep track of which addresses are in use and which are available.
- Configure a secondary server to share the distribution or handle leases if the first DHCP server goes down. This is known as DHCP failover. For information on Managing DHCP Failover, see [Managing DHCP Failover, on page 55](#).

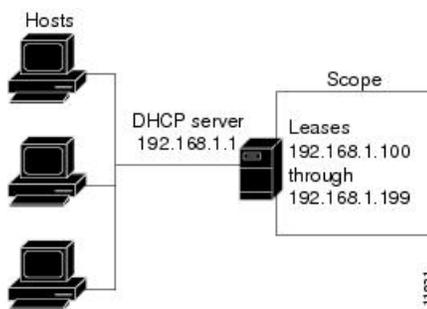
Leases

One of the most significant benefits of DHCP is that it can dynamically configure devices with IP addresses and associate leases with the assigned addresses. DHCP uses a lease mechanism that offers an automated, reliable, and safe method for distributing and reusing addresses in networks, with little need for administrative intervention. As system administrator, you can tailor the lease policy to meet the specific needs of your network.

Leases are grouped together in an address pool, called a scope, which defines the set of IP addresses available for requesting hosts. A lease can be reserved (the host always receives the same IP address) or dynamic (the host receives the next available, unassigned lease in the scope). The DHCP server of the site is configured to lease addresses 192.168.1.100 through 192.168.1.199 (see the image below).

If you plan not to have more network devices than configured addresses for the scope, you can define long lease times, such as one to two weeks, to reduce network traffic and DHCP server load.

Figure 2: DHCP Hosts Requesting Leases from a DHCP Server



Scopes and Policies

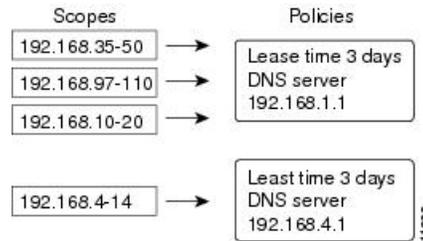
A scope contains a set of addresses for a subnet, along with the necessary configuration parameters. You must define at least one scope for each subnet for which you want dynamic addressing.

A policy includes lease times and other configuration parameters that a DHCP server communicates to clients. Use policies to configure DHCP options that the DHCP server supplies to a client upon request. Policies ensure that the DHCP server supplies all the correct options for scopes without having to do so separately for each scope (see the image below).

The difference between scopes and policies is that scopes contain server information about addresses, such as which address is leasable and whether to ping clients before offering a lease. Policies contain client configuration data, such as the lease duration and address of the local DNS server.

Policies are especially useful if you have multiple scopes on a server. You can create policies that apply to all or selected scopes. The Cisco Prime Network Registrar policy hierarchy is a way to define policies from least to most specific. For example, you usually specify a router option for each policy, which means that you would need a policy for each scope. Scope-specific policies like this can be defined in a scope-embedded policy. More general policies, such as those referring to lease times, can be applied in a system-wide policy (see [Configuring DHCP Policies, on page 161](#)). You can also write extensions to handle policy assignments (see [Using Extensions to Affect DHCP Server Behavior, on page 26](#)).

Figure 3: Scopes and Policies



Links and Prefixes

The explicit DHCPv6 configuration objects are links and prefixes:

- **Link**—Network segment that can have one or more prefixes, and adds an additional layer at which policies can be applied for DHCPv6 clients.
- **Prefix**—Equates to a scope in IPv4. The link associated with a prefix is similar to a primary scope, except that it names a link and not another prefix.

Just as with scopes, you can create multiple prefix objects for the same IPv6 prefix. However, rather than supporting multiple ranges with explicit start and end addresses, prefixes support only a single range that must be an IPv6 prefix with a length the same as, or longer than, the prefix object. For example, if you define a 2001::/64 prefix with a 2001::/96 range, the server can assign addresses from 2001:0:0:0:0:0:0 through 2001:0:0:0:0:0:ffff:ffff only. The range:

- Is limited to powers of 2.
- Must be unique (cannot be duplicated by any other range, except in a different VPN).
- Cannot be contained in, or contain, another range, except for prefix delegation prefixes, as explained below.
- Is the full IPv6 prefix if not specified, except for prefix delegation prefixes, as explained below.

If a prefix delegation prefix object is defined with an unspecified range, it may contain non prefix-delegation prefixes, and the effective range is either:

- The full IPv6 prefix if no other prefixes exist with the same IPv6 prefix, or
- The prefixes that remain when all other ranges for prefix objects with the same IPv6 prefix are removed from the IPv6 prefix.

You create a link only if more than one prefix object with a different IPv6 prefix exists on a link. When the server loads the configuration, if a prefix has no explicit link, the server searches for or creates an implicit link with the name `Link-[vpn.name]/prefix`. All prefix objects with the same IPv6 prefix must either not specify a link or explicitly specify the same link.

The DHCPv6-enabled server supports VPN address spaces for DHCPv6. Both the link and prefix objects may be assigned to a VPN. But all prefixes on a link must use the same VPN ID. Because there is presently no DHCPv6 VPN option, clients can only be assigned addresses from a VPN by using the `client` or `client-class` `override-vpn` attribute.

Related Topics

[Determining Links and Prefixes, on page 127](#)

[Generating Addresses, on page 127](#)

[Generating Delegated Prefixes, on page 128](#)

[Prefix Stability, on page 128](#)

Cisco Prime Network Registrar DHCP Implementations

The Cisco Prime Network Registrar DHCP server provides a reliable method for automatically assigning IP addresses to hosts on your network. You can define DHCP client configurations, and use the Cisco Prime Network Registrar database to manage assigning client IP addresses and other optional TCP/IP and system configuration parameters. The TCP/IP assignable parameters include:

- IP addresses for each network adapter card in a host.
- Subnet masks for the part of an IP address that is the physical (subnet) network identifier.
- Default gateway (router) that connects the subnet to other network segments.
- Additional configuration parameters you can assign to DHCP clients, such as a domain name.

Cisco Prime Network Registrar automatically creates the databases when you install the DHCP server software. You add data through the web UI or CLI as you define DHCP scopes and policies.

The Cisco Prime Network Registrar DHCP server also supports allocating addresses in virtual private networks (VPNs) and subnets to pool manager devices for on-demand address pools. These features are described in the following sections.

Related Topics

[Virtual Private Networks, on page 5](#)

[Subnet Allocation and DHCP Address Blocks, on page 93](#)

Virtual Private Networks

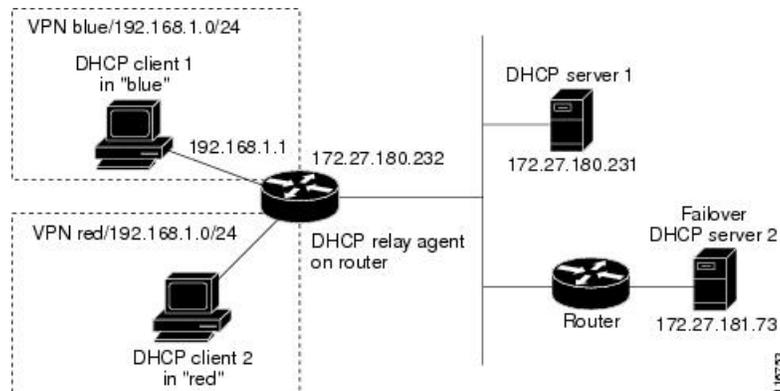
Virtual private networks (VPNs) allow the possibility that two pools in separate networks can have the same address space, with these two pools having overlapping private network addresses. This can save address resources without having to use valuable public addresses. These VPN addresses, however, require a special designator to distinguish them from other overlapping IP addresses. Cisco Prime Network Registrar DHCP servers that are not on the same VPN as their clients can now allocate leases and addresses to these clients, and can distinguish the addresses from one VPN to another.

Through changes made to the Cisco Prime Network Registrar DHCP server and Cisco IOS DHCP Relay Agent, the DHCP server can service clients on multiple VPNs. A VPN distinguishes a set of DHCP server objects, making them independent of otherwise identical objects in other address spaces. You can define multiple VPNs containing the same addresses. You create a VPN based on the VPN identifier configured in the Cisco IOS Relay Agent.

The illustration below shows a typical VPN-aware DHCP environment. The DHCP Relay Agent services two distinct VPNs, blue and red, with overlapping address spaces. The Relay Agent has the interface address 192.168.1.1 on VPN blue and is known to DHCP Server 1 as 172.27.180.232. The server, which services address requests from DHCP Client 1 in VPN blue, can be on a different network or network segment than the client, and can be in a failover configuration with DHCP Server 2 (see [Managing DHCP Failover, on page 55](#)). The Relay Agent can identify the special, distinguished route of the client address request to the DHCP

server, as coordinated between the Relay Agent and Cisco Prime Network Registrar administrators. The DHCP servers can now issue leases based on overlapping IP addresses to the clients on both VPNs.

Figure 4: Virtual Private Network DHCP Configuration

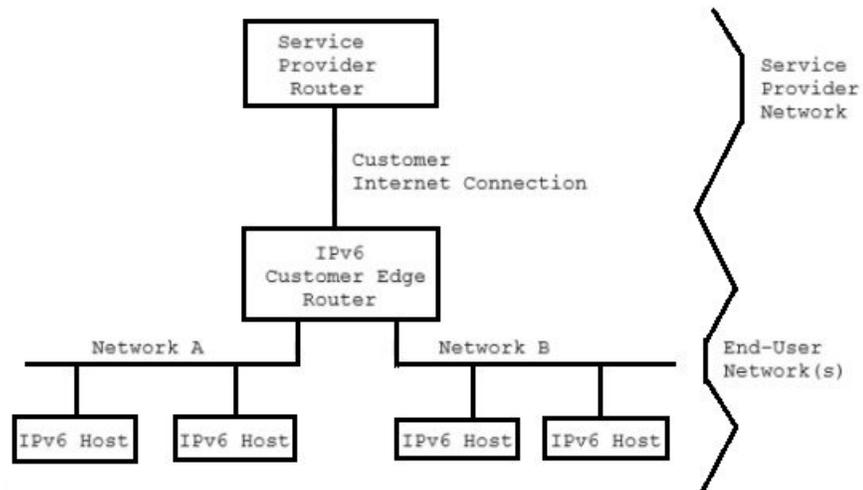


Prefix Delegation

Prefix delegation enables delegation of prefixes from a DHCPv6 server to a requesting device. Prefix Delegation is used by service providers to assign a prefix to a Customer Premise Equipment (CPE) device. It is also used by an ISP to delegate a prefix to a subscriber.

During operation, a DHCPv6 server is provided IPv6 prefixes to be delegated to the requesting device. The requesting device requests prefix(es) from the DHCPv6 server. The DHCPv6 server chooses prefix(es) for delegation, and responds with prefix(es) to the requesting device. The requesting device is then responsible for the delegated prefix(es). For example, the requesting device might assign a subnet from a delegated prefix to one of its interfaces, and begin sending advertisements for the prefix on that link. Each prefix has an associated valid and preferred lifetime, which constitutes an agreement about the length of time over which the requesting device is allowed to use the prefix. A requesting device can request an extension of the lifetimes on a delegated prefix and is required to terminate the use of a delegated prefix if the valid lifetime of the prefix expires.

Figure 5: Model Topology for the end-user network.



DNS Update

Although DHCP frees you from the burden of distributing IP addresses, it still requires updating the DNS server with DHCP client names and addresses. DNS update automates the task of keeping the names and addresses current. With the Cisco Prime Network Registrar DNS update feature, the DHCP server can tell the corresponding DNS server when a name-to-address association occurs or changes. When a client gets a lease, Cisco Prime Network Registrar tells the DNS server to add the host data. When the lease expires or when the host gives it up, Cisco Prime Network Registrar tells the DNS server to remove the association.

In normal operation, you do not have to manually reconfigure DNS, no matter how frequently clients' addresses change through DHCP. Cisco Prime Network Registrar uses the hostname that the client device provides. You also can have Cisco Prime Network Registrar synthesize names for clients who do not provide them, or use the client lookup feature to use a preconfigured hostname for the client.

Different use-cases for DHCPv4 and DHCPv6 DNS update made server design different to handle hostname updates. So, the difference of behavior in DHCPv4 and DHCPv6 DNS updates for hostname is expected.

Related Topics

[Effect on DNS of Obtaining Leases, on page 7](#)

[Effect on DNS of Releasing Leases, on page 8](#)

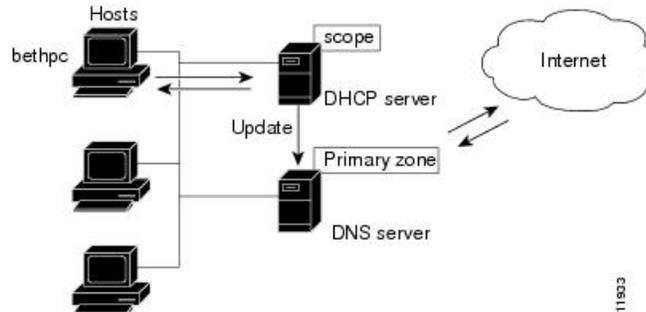
[Effect on DNS of Reacquiring Leases, on page 8](#)

Effect on DNS of Obtaining Leases

For ExampleCo, the administrator creates a scope on the DHCP server and allocates 100 leases (192.168.1.100 through 192.168.1.199). Each device gets its owner name. The administrator also configures the DHCP server to use DNS update and associates it with the correspondingly configured DNS server. The administrator does not need to enter the names in the DNS server database.

Monday morning, Beth (user of bethpc) tries to log into a website without having an address. When her host starts up, it broadcasts an address request (see the image below).

Figure 6: DNS Update at ExampleCo Company



The DHCP server then:

1. Gives bethpc the next available (unassigned) IP address (192.168.1.125).
2. Updates her DNS server with the hostname and address (bethpc 192.168.1.125).

Beth can now access the website. In addition, programs that need to translate the name of Beth's machine to her IP address, or the other way around, can query the DNS server.

Effect on DNS of Reacquiring Leases

When Beth returns from her trip to start up her host again:

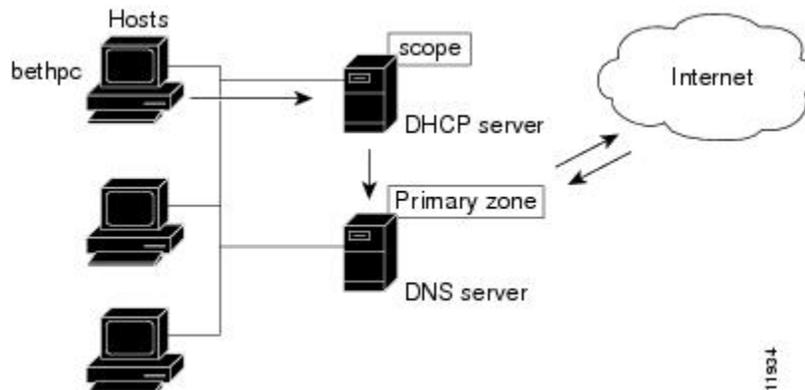
1. Her pc broadcasts for an IP address.
2. The DHCP server checks if the host is on the correct network. If so, the server issues an address. If not, the server on the correct network issues the address.
3. The DHCP server updates the DNS server again with the host and address data.

Effect on DNS of Releasing Leases

Later that day, Beth learns that she needs to travel out of town. She turns off her host, which still has a leased address that is supposed to expire after three days. When the lease is released, the DHCP server:

1. Acknowledges that the IP address is now available for other users (see the figure below).
2. Updates the DNS server by removing the hostname and address. The DNS server no longer stores data about bethpc or its address.

Figure 7: Relinquishing a Lease



DHCP Failover

Cisco Prime Network Registrar failover protocol is designed to allow a backup DHCP server to take over for a main server if the main server is taken offline for any reason. Prior to 8.2, this protocol was UDP based, only operated over IPv4, and only supported DHCPv4. Starting with 8.2, this protocol is TCP based, can be configured to use either IPv4 or IPv6, and supports both DHCPv4 and DHCPv6 over a single connection. With 9.0, this protocol now will try both IPv4 and IPv6 transports if configured to use both, and will use whichever connection comes up first. The existing DHCP clients can keep and renew their leases without the need to know which server is responding to their requests.

You can create and synchronize failover pairs at the local and regional clusters in Cisco Prime Network Registrar. For details, see [Managing DHCP Failover, on page 55](#).

Allocating Addresses Through Failover

In order to keep the failover pair operating in spite of a network partition, in which both can communicate with clients but not with each other, you must make available more addresses than the addresses needed to run a single server. Configure the main server to allocate a percentage of the currently available (unassigned) addresses in each scope or prefix delegation address pool to its partner. These addresses become unavailable to the main server. The partner uses them when it cannot talk to the main server and does not know if it is down. However, when the failover partners are in communication, they periodically rebalance these pools.

The backup server needs enough addresses from each scope or prefix to satisfy the requests of all new DHCP clients that arrive during the period in which the backup does not know if the main server is down. In Cisco Prime Network Registrar 8.2 or later, the default backup percentage for a failover pair is 50%. This ensures that during the failover the other partner has equal number of addresses.

Even during PARTNER-DOWN state, the backup server waits for the lease expiration and the maximum client lead time (MCLT), a small additional time buffer, before reallocating any leases. When these times expire, the backup server offers:

- Leases from its private pool of addresses.
- Leases from the main server pool of addresses.
- Expired leases to new clients.

During the working hours, if the administrative staff can respond within two hours to a COMMUNICATIONS INTERRUPTED state to determine if the main server is working, the backup server needs enough addresses to support a reasonable upper bound on the number of new DHCP clients that might arrive during those two hours.

During off-hours, if the administrative staff can respond within 12 hours to the same situation, and considering that the arrival rate of previously unheard from DHCP clients is also less, the backup server then needs enough addresses to support a reasonable upper bound on the number of DHCP clients that might arrive during those 12 hours.

Consequently, the number of addresses over which the backup server requires sole control would be the greater of the numbers of addresses given out during peak and non-peak times, expressed as a percentage of the currently available (unassigned) addresses in each scope or prefix.



Note Starting in 8.2, the default use-safe-period is enabled for the DHCP failover pair and the default safe period is 4 hours. This ensures that if the failover partner is in COMMUNICATIONS-INTERRUPTED state for 4 hours, it will enter PARTNER-DOWN state automatically after the safe period elapses.

Client-Classes

You can use the Cisco Prime Network Registrar client and client-class facility to provide differentiated services to users that are connected to a common network. You can group your user community based on administrative criteria, and then ensure that each user receives the appropriate class of service.

Although you can use the Cisco Prime Network Registrar client-class facility to control any configuration parameter, the most common uses are for:

- **Lease periods**—How long a set of clients should keep their addresses.
- **IP address ranges**—From which lease pool to assign clients addresses.
- **DNS server addresses**—Where clients should direct their DNS queries.
- **DNS hostnames**—What name to assign clients.
- **Denial of service**—Whether unauthorized clients should be offered leases.

One way to use the client-class facility is to allow visitors access to some, but not all, of your network. For example, when Joe, a visitor to ExampleCo, tries to attach his laptop to the example.com network, Cisco Prime Network Registrar recognizes the laptop as being foreign. ExampleCo creates one class of clients known as having access to the entire network, and creates another visitor class with access to a subnet only. If Joe needs more than the standard visitor access, he can register his laptop with the Cisco Prime Network Registrar system administrator, who adds him to a different class with the appropriate service.

The following sections describe how DHCP normally processes an address assignment, and then how it would handle it with the client-class facility in effect.

Related Topics

[DHCP Processing Without Client-Classes, on page 11](#)

[DHCP Processing with Client-Classes, on page 11](#)

[Defining Scopes for Client-Classes, on page 12](#)

[Choosing Networks and Scopes, on page 12](#)

DHCP Processing Without Client-Classes

To understand how you can apply client-class processing, it is helpful to know how the DHCP server handles client requests. The server can perform three tasks:

- Assign an IP address.
- Assign the appropriate DHCP options (configuration parameters).
- Optionally assign a fully qualified domain name (FQDN) and update the DNS server with that name.

The DHCP server:

1. Assigns an address to the client from a defined scope—To choose an address for the client, the DHCP server determines the client subnet, based on the request packet contents, and finds an appropriate scope for that subnet.

If you have multiple scopes on one subnet or several network segments, which is known as multinetting, the DHCP server may choose among these scopes in a round-robin fashion, or you can change the priority of the scope choice by using the DHCP server address allocation priority feature (see [Configuring Multiple Scopes Using Allocation Priority, on page 114](#)). After the server chooses a scope, it chooses an available (unassigned) address from that scope:

1. It assigns DHCP option values from a defined policy. Cisco Prime Network Registrar uses policies to group options. There are two types of policies: scope-specific and system default. For each DHCP option the client requests, the DHCP server searches for its value in a defined sequence.
 2. If the scope-specific policy contains the option, the server returns its value to the client and stops searching.
 3. If not found, the server looks in the system default policy, returns its value, and stops searching.
 4. If neither policy contains the option, the server returns no value to the client and logs an error.
 5. The server repeats this process for each requested option.
2. With DNS update in effect, the server assigns an FQDN to the client. If you enabled DNS update, Cisco Prime Network Registrar enters the client name and address in the DNS host table. See [DNS Update, on page 7](#). The client name can be:
 - Its name as specified in the client lease request (the default value).
 - Its MAC address (hardware address; for example, 00:d0:ba:d3:bd:3b).
 - A unique name using the default prefix *dhcp* or a specified prefix.

DHCP Processing with Client-Classes

When you enable the client-class facility for your DHCP server, the request processing performs the same three tasks of assigning IP addresses, options, and domain names as described in [DHCP Processing Without Client-Classes, on page 11](#), but with added capability. The DHCP server:

1. **Considers the client properties and client-class inclusion before assigning an address**—As in regular DHCP processing, the DHCP server determines the client subnet. The server then checks if there is a client-class defined or a MAC address for this client in its database. If there is:
 1. A client-class defined by a client-class lookup ID expression, the client is made a member of this client-class.

2. No MAC address, it uses the default client. For example, the default client could have its client-class name set to Guest, and that client-class could limit (using options and address selection) what network operations such clients are permitted.
3. No MAC address and no default client, the server handles the client through regular DHCP processing.
4. No client-specifier, but a MAC address, the MAC address is converted into a client-specifier. An unknown client is mapped to the default client, if the default client is defined.

The scopes must have addresses on client-accessible subnets. That is, they must have a selection tag that associates them with a client-class. To assign the same clients to different address pools, you must use separate scopes.

For example, a scope would either have a selection tag of Employee or Guest, but not both. In this case, there are two scopes for each subnet; one with the selection tag Employee, and the other with Guest. Each scope has a different associated policy and address range that provides the appropriate access rights for the user group.

2. **Checks for client-class DHCP options**—In regular DHCP processing, the server checks the scope-specific and system default DHCP options. With client-class, it also first checks the client-specific and client-class-specific options.
3. **Provides additional FQDN assignment options**—Beyond the usual name assignment process of using the hostname the client requests, the server can:
 - Provide an explicit hostname that overrides it.
 - Drop the client-requested hostname and not replace it.
 - Synthesize a hostname from the client MAC address.

Defining Scopes for Client-Classes

The motivating factor for using client-classes is often to offer an address from one or another address pool to a client. Another motivating factor might be to provide clients with different option values or lease times. Offering clients addresses from separate pools requires defining more than one scope.

To get more than one scope on a subnet, they must come from the same network segment. Networks are not configured directly in Cisco Prime Network Registrar, but are inferred from scope configurations. Scopes become related (end up in the same network):

- **Implicitly**—Two scopes have the same network number and subnet mask. These scopes naturally end up on the same network without explicit configuration.
- **Explicitly**—One scope is marked as a secondary to another. This is required when the scope marked as a secondary has a network and subnet mask unrelated to the primary. An example is putting a set of 10.0.0.0 network addresses on a normal, routable network segment.

When the Cisco Prime Network Registrar DHCP server reads the scope configuration from its database, it places every scope in a network, and logs this information. Scopes with the same network number and subnet mask end up on the same network, while a secondary scope ends up on the primary scope network.

Choosing Networks and Scopes

When a DHCP packet arrives, the server determines the address from which it came by:

- When a DHCPv4 packet arrives the server determines the gateway address (*giaddr*), if there was one, for packets sent through a BOOTP relay.
- For information on DHCPv6, see [Managing Scopes, Prefixes, Links, and Networks](#).

- Interface address of the interface on which the broadcast packet arrived, if the DHCP client is on a network segment to which the DHCP server is also directly connected.

In all cases, the DHCP server determines a network from the gateway or interface address. Then, if the network has multiple scopes, the server determines from which scope to allocate an address to the DHCP client. It always looks for a scope that can allocate addresses to this type of client. For example, a DHCP client needs a scope that supports DHCP, and a BOOTP client needs one that supports BOOTP. If the client is a DHCP client and there are multiple scopes that support DHCP, each with available (unassigned) addresses, the DHCP server allocates an IP address from any of those scopes, in a round-robin manner, or by allocation priority.

Selection tags and client-classes let you configure the DHCP server to allocate IP addresses from:

- One or more scopes on a network to one class of clients.
- A different set of scopes to a different class of clients.

In the latter case, the gateway or interface address determines the network. The client-class capability, through the mechanism of the selection tags, determines the scope on the network to use.



CHAPTER 2

Managing DHCP Server

This chapter describes how to set up some of the DHCP server parameters. Before clients can use DHCP for address assignment, you must add at least one scope to the server. This is described in [Managing Scopes, Prefixes, Links, and Networks](#), on page 111.

Cisco Prime Network Registrar failover protocol is designed to allow a backup DHCP server to take over for a main server if the main server is taken offline for any reason. To set up DHCP failover, read [Managing DHCP Failover](#), on page 55.

- [Configuring DHCP Servers](#), on page 15
- [Defining Advanced Server Attributes](#), on page 16
- [Setting DHCP Forwarding](#), on page 23
- [Editing DHCPv6 Server Attributes](#), on page 24
- [Integrating Windows System Management Servers](#), on page 25
- [Using Extensions to Affect DHCP Server Behavior](#), on page 26
- [Tuning the DHCP Server](#), on page 31
- [Listing Related Servers for DHCP - Failover, DNS, LDAP, and TCP Listener Servers](#), on page 34
- [Configuring Virtual Private Networks](#), on page 43
- [Configuring Subnet Allocation](#), on page 48
- [Configuring BOOTP](#), on page 50

Configuring DHCP Servers

When configuring a DHCP server, you must configure the server properties, policies, and associated DHCP options. Cisco Prime Network Registrar needs:

- The DHCP server IP address.
- One or more scopes (see the [Managing Scopes](#), on page 111) and/or prefixes.

General Configuration Guidelines

Here are some guidelines to consider before configuring a DHCP server:

- **Separate the DHCP server from secondary DNS servers used for DNS updating**—To ensure that the DHCP server is not adversely affected during large zone transfers, it should run on a different cluster than your secondary DNS servers.
- **Lease times**—See the [Guidelines for Lease Times](#), on page 186.

Configuring DHCP Server Interfaces

To configure the DHCP server, accept the Cisco Prime Network Registrar defaults or supply the data explicitly:

- **Network interface**—Ethernet card IP address, which must be static and not assigned by DHCP.
- **Subnet mask**—Identifies the interface network membership. The subnet mask is usually based on the network class of the interface address, in most cases 255.255.255.0.

By default, the DHCP server uses the operating system support to automatically enumerate the active interfaces on the machine and listens on all of them. You can also manually configure the server interface. You should statically configure all the IP addresses assigned to NIC cards on the machine where the DHCP server resides. The machine should not be a BOOTP or DHCP client.



Note Unless you have a specific need to restrict the interfaces used for DHCP, it is recommended that you do not configure specific DHCP Server interfaces. Allow the server to automatically discover the available interfaces.

Local Advanced Web UI

-
- Step 1** From the **Operate** menu, choose **Manage Servers** under the **Servers** submenu to open the Manage Servers page.
 - Step 2** Select the **Local DHCP server** on the Manager Servers pane.
 - Step 3** Click the **Network Interfaces** tab to view the available network interfaces that you can configure for the server. By default, the server uses all of them.
 - Step 4** To configure an interface, click the **Edit** icon in the Configure column for the interface. This adds the interface to the Configured Interfaces table, where you can edit or delete it.
 - Step 5** Clicking the name of the configured interface opens the Edit DHCP Server Network Interface page, where you can change the address and ports (in Expert mode) of the interface.
 - Step 6** Click **Save** when you are done editing.
 - Step 7** Click **Revert** to return to the Manage Servers page.
-

CLI Commands

Use **dhcp-interface** to manually control which network interface cards' IP addresses the DHCP server will listen on for DHCP clients. By default, the DHCP server automatically uses all your server network interfaces, so use this command to be more specific about which ones to use.

To troubleshoot and confirm validity of the configuration changes.

- Reload the DHCP server.
- Check the `dhcp_startup_log` and/or `name_dhcp_1_log` file.

For information on Log Settings, see [Tuning the DHCP Server, on page 31](#).

Defining Advanced Server Attributes

You can set advanced DHCP server attributes, including custom DHCP options.

To set up the DHCP server.

1. Configure a scope or prefix.
2. Reload the server.

Related Topics

[Setting Advanced DHCP Server Attributes, on page 17](#)

[Enabling BOOTP for Scopes, on page 52](#)

[Moving or Decommissioning BOOTP Clients, on page 52](#)

[Using Dynamic BOOTP, on page 52](#)

[BOOTP Relay, on page 53](#)

Setting Advanced DHCP Server Attributes

The table below describes the advanced DHCP server attributes that you can set in the local cluster web UI and CLI.

Table 1: DHCP Advanced Attributes

Advanced Parameter	Action	Description
<i>max-dhcp-requests</i>	set/ unset	<p>Controls the number of buffers that the DHCP server allocates for receiving packets from DHCP clients and failover partners. If this setting is too large, a burst of DHCP activity can clog the server with requests that become stale before being processed. This results in an increasing processing load that can severely degrade performance as clients try to obtain a new lease, and affects the ability to handle bursts. A low buffer setting throttles requests and could affect server throughput. If the server runs out of buffers, packets are dropped.</p> <p>A good rule of thumb is to increase the buffers if you expect a high load (in a steady state or when experiencing frequent stress times) or you have a fast multiprocessor system.</p>

Advanced Parameter	Action	Description
		<p>In a nonfailover deployment, the default setting (500) is sufficient. In a failover deployment, you can increase it to 1000 if the DHCP logs indicate a consistently high number of request buffers. You should then also modify the number of DHCP responses (see the <i>max-dhcp-responses</i> parameter) to four times the request buffers.</p> <p>When using LDAP client lookups, buffers should not exceed the LDAP lookup queue size defined by the total number of LDAP connections and the maximum number of requests allowed for each connection. Set the LDAP queue size to match the capacity of the LDAP server to service client lookups.</p> <p>If the following logs messages occur frequently and are not related to short term traffic spikes (such as after a power recovery), you may want to consider increasing the value of the attribute:</p> <pre>4493 DHCP ERROR "DHCP has used xx of its yy request buffers: the server is dropping a request." 4494 DHCP WARNING "DHCP has used xx of yy request packets. Requests will be ignored if no packet buffers are available." 5270 DHCP WARNING "DHCP has used xx of its yy request buffers: the server is congested -- will not keep the client last-transaction-time to within value but will keep it to within value seconds."</pre> <p>Required. The default is 500.</p>

Advanced Parameter	Action	Description
<i>max-dhcp-responses</i>	set/ unset	<p>Controls the number of response buffers that the DHCP server allocates for responding to DHCP clients and performing failover communication between DHCP partners.</p> <p>In a non-failover deployment, the default setting of twice the number of request buffers is sufficient. In a failover deployment, you can increase this so that it is four times the number of request buffers. In general, increasing the number of response buffers is not harmful, while reducing it to below the previously recommended ratios might be harmful to server responsiveness.</p> <p>If the following logs messages occur frequently and are not related to short term traffic spikes (such as after a power recovery), you may want to consider increasing the value of the attribute:</p> <pre>4721 DHCP ERROR "DHCP has used all xx response packets. A request was dropped and they will continue to be dropped if no responses are available." 5289 DHCP WARNING "DHCP has used xx of yy response packets. Requests will be dropped if no responses are available."</pre> <p>Required. The default is 1000.</p>
<i>max-ping-packets</i>	set/ unset	<p>Controls the number of buffers that the server has available to initiate Ping requests to clients. If you enable the <i>Ping address before offering it</i> option at the scope level, packet buffers are used to send and receive ICMP messages. If you enable pinging, you should have enough ping packets allocated to handle the peak load of possible ping requests. The default is 500 ping packets.</p>
<i>hardware-unicast</i>	enable/ disable	<p>Controls whether the DHCP server sends unicast rather than broadcast responses when a client indicates that it can accept a unicast. This feature is only available on Windows platform; other operating systems broadcast instead. The default is enabled.</p>

Advanced Parameter	Action	Description
<i>defer-lease-extensions</i>	enable/ disable	Controls whether the DHCP server extends leases that are less than half expired. This is a performance tuning attribute that helps minimize the number of disk writes to the lease state database. The default is checked or true. This means that a client renewing a lease less than halfway through can get the remaining part of it only and not be extended. See Deferring Lease Extensions, on page 22 .
<i>last-transaction-time-granularity</i>	set/ unset	<p>The default value of the last-transaction-time-granularity attribute has changed from 60 seconds to one week. This new default means that the client-last-transaction-time may not accurately reflect the last time the client communicated with the server.</p> <p>If your deployment depended on this attribute being updated whenever the client communicated with the server, you need to explicitly set the last-transaction-time-granularity attribute to a value appropriate for the deployment.</p> <p>The last-transaction-time-granularity attribute is effectively not used when you have disabled defer-lease-extensions. Therefore, if you have disabled defer-lease-extensions, this change in the default value does not impact you.</p> <p>When the server is heavily loaded and has run low on request or response buffers, the server temporarily sets the last-transaction-time-granularity value to one year to reduce its load.</p>

Advanced Parameter	Action	Description
<i>discover-queue-limit</i>	set/ unset	<p>Specifies the percentage limit of the request buffers that may be used for DHCPDISCOVER and SOLICIT client requests at any time. Once the configured percentage of the request buffers is exceeded, additional DHCPDISCOVER and SOLICIT client requests are discarded. By restricting the requests buffers that can be used by DHCPDISCOVER/SOLICIT requests, the server assures it has request buffers available to process DHCPREQUEST/REQUEST requests and this can greatly reduce the time needed to get clients online during spikes in activity, such as after a power recovery or CMTS reboot.</p> <p>The DRL (Discriminating Rate Limiter) attribute controls the discriminating rate limiter capability. The Discriminating Rate Limiter is enabled by default and assures that the DHCP server prefers completing DHCP transaction over starting too many new ones. In many situations, this should expedite bringing all clients online. If activity summary logging is enabled, the number of DHCPDISCOVER (DHCPv4) and SOLICIT (DHCPv6) packets dropped because of rate limiting is reported as DRL:number.</p> <p>The DHCPv4 statistics includes a new queue-limited-discovers-dropped counter and the DHCPv6 statistics includes a new queue-limited-solicits-dropped counter. These counters are used to monitor the packets that are dropped.</p>

Local Basic or Advanced Web UI

-
- Step 1** From the **Deploy** menu, choose **DHCP Server** under the **DHCP** submenu to open the Manage DHCP Server page.
 - Step 2** Select the server from the DHCP Server pane.
 - Step 3** Add or modify attributes on the Edit Local DHCP Server page.
 - Step 4** Click **Save** after making the changes.
-

CLI Commands

Use **dhcp show** and **dhcp get attribute** to show the current server parameters, then use **dhcp set attribute=value [attribute=value ...]**, **dhcp unset attribute**, **dhcp enable attribute**, and **dhcp disable attribute** to change them (see the table above).

Deferring Lease Extensions

Enabling the *defer-lease-extensions* attribute (which is its preset value) allows the DHCP server to optimize response to a sudden flood of DHCP traffic. An example of a network event that could result in such a traffic spike is a power failure at a cable internet service provider (ISP) data center that results in all of its cable modem termination systems (CMTS) rebooting at once. If this happens, the devices attached to the CMTSs produce a flood of DHCP traffic as they quickly come back online.

With the *defer-lease-extensions* attribute enabled, the DHCP server might defer extending the lease expiration time for a client's renewal request, which typically occurs before T1 (usually before halfway through the lease). Instead of giving the client the full configured lease time, the server grants the remaining time on the existing lease. Because the absolute lease expiration time does not change, the server can avoid database updates that result in a significantly higher server throughput. Another benefit is avoiding having to update the failover partner with an extended lease expiration time.

If a client is at or beyond T1 (typically halfway to its expiration), enabling or disabling this attribute has no effect, and the server always tries to extend the lease expiration time. However, failover and other protocol restrictions can prevent the server from extending the lease for the full configured time.



Note Deferring lease extensions significantly increases the server performance while remaining in compliance with the DHCP RFC, which stipulates that client binding information is committed to persistent storage when the lease changes.

When deferring lease extensions, it is advisable to leave the policy attribute *allow-lease-time-override* to its default of disabled, or to change it to disabled if it is enabled.

These three specific situations are described from the server point of view:

- **Client retries**—When the server gets behind, it is possible for a client to retransmit requests. The DHCP server does not maintain enough information to recognize these as retransmissions, and processes each to completion, granting a full lease duration again and updating the database. When the server is already behind, doing extra work worsens the situation. To prevent this, the DHCP server does not extend leases that are less than 30 seconds old, regardless of the state of the *defer-lease-extensions* attribute.
- **Client reboots**—The effective renew time for a client lease is really the minimum of the configured renew time and the time between client reboots. In many installations this may mean that clients get fresh leases one (in a typical enterprise) or two (in a typical cable network) times per day, even if the renew time is set for many days. Setting the *defer-lease-extensions* attribute can prevent these early renews from causing database traffic.
- **Artificially short renewal times**—Because there is no way for a DHCP server to proactively contact a DHCP client with regard to a lease, you might configure short lease times on the DHCP server to provide a means of doing network renumbering, address reallocation, or network reconfiguration (for example, a change in DNS server address) in a timely fashion. The goal is to allow you to do this without incurring unacceptable database update overhead.

As a complication, the server also keeps track of the time when it last heard from the client. Known as the last transaction time, sites sometimes use this information as a debugging aid. Maintaining this time robustly requires a write to the database on every client interaction. The *last-transaction-time-granularity* attribute is the one to set. (See the attribute description in [Table 1: DHCP Advanced Attributes](#).) Because it is primarily a debugging aid, the value need not be entirely accurate. Furthermore, because the in-memory copy is always accurate, you can use **export leases –server** to display the current information, even if the data is not up to date in the database.

Setting DHCP Forwarding

The Cisco Prime Network Registrar DHCP server supports forwarding DHCP packets to another DHCP server on a per-client basis. For example, you might want to redirect address requests from certain clients, with specific MAC address prefixes, to another DHCP server. This can be useful and important in situations where the server being forwarded to is not one that you manage. This occurs in environments where multiple service providers supply DHCP services for clients on the same virtual LAN.

Enabling DHCP forwarding requires implementing an extension script. The DHCP server intercepts the specified clients and calls its forwarding code, which checks the specified list of forwarded server addresses. It then forwards the requests rather than processing them itself. You attach and detach extensions to and from the DHCP server by using **dhcp attachExtension** and **dhcp detachExtension**.

The DHCP forwarding feature works like this:

1. When DHCP is initialized, the server opens a UDP socket, which it uses to send forwarded packets. To support servers with multiple IP addresses, the socket address pair consists of `INADDR_ANY` and any port number. This enables clients to use any one of the server IP addresses.
2. When the DHCP server receives a request from a client, it processes these extension point scripts:
 - **post-packet-decode**
 - **pre-client-lookup**
 - **post-client-lookup**

As the DHCP server processes these scripts, it checks the environment dictionary for this string:

```
cnr-forward-dhcp-request
```

3. When it finds that string and it has the value *true* (enabled), the server calls its forwarding code.
4. The forwarding code checks the environment dictionary for a string with this key:

```
cnr-request-forward-address-list
```

It expects a list of comma-separated IP addresses with an optional colon-delimited port number, as in this example:

```
192.168.168.15:1025,  
192.168.169.20:  
1027
```

By default, the server forwards to `server-port` for DHCPv4 and `v6-server-port` for DHCPv6. It sends a copy of the entire client request to each IP address and port in turn. If any element in the list is invalid, the server stops trying to parse the list.

5. After the forwarding code returns, the server stops processing the request. In the `post-client-lookup` extension point script, however, this action might create an optional log message with client-entry details.

The following example of a portion of a TCL extension script tells the DHCP server to forward a request to another server based on the information in the request. You can use such a script if there are multiple device provisioning systems in the same environment. In this case, you would run the extension script on the DHCP server to which routers forward broadcast requests. The script would determine which (if any) other server or servers should handle the request, and tell the original server to forward the request.

The sample script uses a static mapping of MAC address prefix to send modems from a specific vendor to a specific system:

```
proc postPktDecode {req resp env} {
  set mac [$req get chaddr]
  set addr ""
  ;# Very simple, static classifier that forwards all requests from devices
  ;# with a vendor-id of 01:0c:10 to the DHCP servers at 10.1.2.3 and 10.2.2.3:
  switch -glob -- $mac {
    01:0c:10* {
      set addr "10.1.2.3,10.2.2.3"
    }
  }
  ;# If we decide to forward the packet, the $addr var will have the IP addresses
  ;# where to forward the packet:
  if {$addr != ""} {
    ;# Tell the DHCP server to forward the packet...
    $env put cnr-forward-dhcp-request true
    ;# ...and where to forward it:
    $env put cnr-request-forward-address-list $addr
    ;# No more processing is required.
    return
  }
}
```

A more flexible script could use a per-client configuration object, such as the Cisco Prime Network Registrar client entry, to indicate which DHCP server should get the request.



Note DHCP forwarding is available only for DHCPv4; not for DHCPv6.

Editing DHCPv6 Server Attributes

You can edit DHCP server attributes related to DHCPv6. These attributes are:

- **v6-client-class-lookup-id**—Expression that determines a client-class based on the DHCPv6 client request and returns a string with either the name of a configured client-class or <none> (if the expression does not wish to provide a client-class). The attribute has no preset value.
- **max-client-leases**—Maximum number of leases a DHCPv6 client can have on a link. Do not use this attribute to limit clients to one lease only. The preset is 50.

Local Basic or Advanced Web UI

From the **Deploy** menu, choose **DHCP Server** under the **DHCP** submenu to open the Manage DHCP Server page. Click the **Local DHCP Server** link to open the Edit DHCP Server page, modify the aforementioned DHCPv6 attribute values, then click **Save**.

CLI Commands

Use **dhcp** to show the aforementioned DHCPv6 server attributes, then modify them by using **dhcp set attribute=value [attribute=value ...]**.

Integrating Windows System Management Servers

You can have the DHCP server interact with the Microsoft System Management Server (SMS) so that SMS is current with DHCP changes. Normally, SMS pulls updated data through a DHCPDISCOVER request from the server about any new clients that joined the network. Cisco Prime Network Registrar, however, pushes these updates to SMS when you use **dhcp updateSms**. Before you do, verify that:

- SMS client installation and initialization step is complete.
- Cisco Prime Network Registrar Server Agent is set to run under a login account with sufficient privileges.
- SMS site ID is correct and matches that of the SMS server.

These steps describe how to integrate Windows SMS into Cisco Prime Network Registrar.

Step 1 Install the Microsoft BackOffice 4.5 Resource Kit on the same machine as the Cisco Prime Network Registrar DHCP server. Follow the installation instructions and choose the default settings.

Step 2 After the installation, modify the User Variable search path on the Environment tab of the System control panel to:

```
\program files\ResourceKit\SMS\diagnose
```

Step 3 If the DHCP and SMS servers are on different machines, install the SMS client on the same machine as the DHCP server. The SMS library has the necessary API calls to communicate with the SMS server. You must assign the correct site code from the DHCP server machine. In your Network Neighborhood, go to the path `\\SMS-servername\SMSLOGON\x86.bin\0000409\smsman.exe`.

Run the program and follow the instructions, using the default settings. The program creates two icons that you can use later from the control panel, marked SMS and Remote Control.

Step 4 Stop and then restart the Cisco Prime Network Registrar server agent under a trusted domain account with sufficient privileges. Both the DHCP and SMS servers must be aware of this account. Use this short procedure:

- a) Stop the local cluster server agent process.
- b) Configure the account under which the Cisco Prime Network Registrar services run. Create an account name that is a member of both the trusted SMS site server group and a member of the DHCP server administrator group, with the corresponding password.
- c) Restart the local cluster server agent process.

Step 5 Use **dhcp set sms-library-path** (or the *sms-library-path* attribute under the Microsoft Systems Management Server category on the Edit DHCP Server page) to configure the DHCP server to push lease information to SMS. Include the full path to the SMSRsGen.dll. If you omit a value, the path defaults to the internal server default location of this file. For example:

```
nrcmd> dhcp set sms-library-path /conf/dll
```

When you install the Microsoft BackOffice Resource Kit, the system path is not updated to reflect the location of the SMS data link library (DLL). Use one of these methods to configure this attribute:

a) Set the *sms-library-path* attribute to a relative path:

- First, modify the system PATH variable to append the path of the directory where the DLL is installed:

```
sms-install-directory\diagnose
```

- Then, set *sms-library-path* to the name of the DLL, such as smsrsgen.dll. You can also accept the system default by unsetting the attribute.

- b) Set *sms-library-path* to an absolute path. If you do not want to change the system path, set this attribute to the absolute path of the DLL location:

```
"\\Program Files\\Resource Kit\\sms\\diagnose\\smsgen.dll"
```

Step 6 Set the *sms-network-discovery* DNS attribute to 1 to turn SMS network discovery on.

If you use the default of 0, you disable SMS network discovery.

Step 7 Set the *sms-site-code* DHCP server attribute by entering the SMS site code from **Step 3**.

The default string is empty, but for data discovery to be successful, you must provide the site code.

Step 8 Set the *sms-lease-interval* attribute to the SMS lease interval.

The lease interval is the time between sending addresses to SMS, or how long, in milliseconds, the DHCP server should wait before pushing the next lease to the SMS server when you run **server dhcp updateSms**. Early versions of the SMSRsGen.dll file (SMS Version 2.0) did not allow SMS to reliably receive multiple updates within a one-second window (1000 ms); the default value, therefore, was set to 1.1 second (1100 ms). If you install a future version of the Microsoft BackOffice Resource Kit, which might contain an enhanced version of the SMSRsGen.dll file, then reduce this interval or set it to 0 to increase performance.

Step 9 Reload the DHCP server and check the *dhcp_startup_log* and/or *name_dhcp_1_log* file.

Step 10 In the CLI, use **server dhcp updateSms** to initiate SMS processing. (This command can take an optional **all** keyword to send all leased addresses from the DHCP server to SMS. If you omit this keyword, the DHCP server sends only new leases activated since the last time the command ran.) Then, verify that both the DHCP and SMS logs indicate successful completion. Note that a server reload during SMS updating interrupts the process, but the process resumes (or restarts) after the server is back up.

Using Extensions to Affect DHCP Server Behavior

Cisco Prime Network Registrar provides the ability to alter and customize the operation of the DHCP server through *extensions*, programs that you can write in TCL or C/C++. Extensions interact with the server in two ways: by modifying request or response packets, and through environment variables stored in the environment dictionary (see [Using Extension Points, on page 351](#) for details).

For example, you might have an unusual routing hub that uses BOOTP configuration. This device issues a BOOTP request with an Ethernet hardware type (1) and MAC address in the *chaddr* field. It then sends out another BOOTP request with the same MAC address, but with a hardware type of Token Ring (6). The DHCP server normally distinguishes between a MAC address with hardware type 1 and one with type 6, and considers them to be different devices. In this case, you might want to write an extension that prevents the DHCP server from handing out two different addresses to the same device.

You can solve the problem of the two IP addresses by writing either of these extensions:

- One that causes the DHCP server to drop the Token Ring (6) hardware type packet.
- One that changes the Token Ring packet to an Internet packet and then switches it back again on exit. Although this extension would be more complex, the DHCP client could thereby use either return from the DHCP server.

Related Topics

[Writing Extensions, on page 27](#)

[Preventing Chatty Clients by Using an Extension, on page 28](#)

Writing Extensions

You can write extensions in TCL or C/C++:

- **TCL**—Makes it a bit easier and quicker to write an extension. If the extension is short, the interpreted nature of TCL does not have a serious effect on performance. When you write an extension in TCL, you are less likely to introduce a bug that can crash the server.
- **C/C++**—Provides the maximum possible performance and flexibility, including communicating with external processes. However, the complexity of the C/C++ API is greater and the possibility of a bug in the extension crashing the server is more likely than with TCL.

You create extensions at specific extension points. Extension points include three types of dictionaries—request, response, and environment. One or more of these dictionaries are available for each of the following extension points:

1. **init-entry**—Extension point that the DHCP server calls when it configures or unconfigures the extension. This occurs when starting, stopping, or reloading the server. This entry point has the same signature as the others for the extension. It is required for DHCPv6 processing. Dictionary: environment only.
2. **pre-packet-decode**—First extension point that the DHCP server encounters when a request arrives, and calls it before decoding the packet. Dictionaries: request and environment.
3. **post-packet-decode**—Rewrites the input packet. Dictionaries: request and environment.
4. **post-class-lookup**—Evaluates the result of a *client-class-lookup-id* operation on the client-class. Dictionaries: request and environment.
5. **pre-client-lookup**—Affects the client being looked up, possibly by preventing the lookup or supplying data that overrides the existing data. Dictionaries: request and environment.
6. **post-client-lookup**—Reviews the operation of the client-class lookup process, such as examining the internal server data structures filled in from the client-class processing. You can also use it to change any data before the DHCP server does additional processing. Dictionaries: request and environment.
7. **generate-lease**—Generates and controls a DHCPv6 address or prefix. Dictionaries: request, response, and environment.
8. **check-lease-acceptable**—Changes the results of the lease acceptability test. Do this only with extreme care. Dictionaries: request, response, and environment.
9. **lease-state-change**—Determines when the lease state changes this only with extreme care. Dictionaries: response and environment.
10. **pre-packet-encode**—Changes the data sent back to the DHCP client in the response, or change the address to which to send the DHCP response. Dictionaries: request, response, and environment.
11. **post-packet-encode**—Allows the server to examine and alter the packet before it sends the packet to the client, or drops the packet. Dictionaries: request, response, and environment.
12. **pre-dns-add-forward**—Alters the name used for the DNS forward (A record) request. Dictionary: environment only.
13. **post-send-packet**—Used after sending a packet for processing that you want to perform outside of the serious time constraints of the DHCP request-response cycle. Dictionaries: request, response, and environment.
14. **environment-destroyer**—Allows an extension to clean up any context that it might be holding. Dictionary: environment.

To extend the DHCP server, do the following:

Step 1 Write the extension in Tcl, C or C++ and install it in the server extensions directory, on:

- **UNIX:**
 - **Tcl**—`/opt/nwreg2/local/extensions/DHCP/tcl`
 - **C or C++**—`/opt/nwreg2/local/extensions/DHCP/dex`
- **Windows:**
 - **Tcl**—`\program files\Cisco Prime Network Registrar\extensions\dhcp\tcl`
 - **C or C++**—`\program files\Cisco Prime Network Registrar\extensions\dhcp\dex`

It is best to place these extensions in the appropriate directory for TCL or C/C++ extensions. Then, when configuring the filename, just enter the filename itself, without slash (/) or backslash (\).

If you want to place extensions in subdirectories, enter the filename with a path separator. These are different depending on the operating system on which your DHCP server is running.

Note When entering a filename that contains a backslash (\) character in Windows, you must enter it with a double-backslash (\\), because backslash (\) is an escape character in the CLI. For example, enter the filename `debug\myextension.tcl` as **`debug\\myextension.tcl`**.

Step 2 Use the List/Add DHCP Extensions page in the web UI (In the Advanced mode, from the Deploy menu, choose **Extensions** from the DHCP submenu to open the List/Add DHCP Extensions page) or the **extension** command in the CLI to configure the DHCP server to recognize this extension.

Step 3 Attach the configured extension to one or more DHCP extension points by using **dhcp attachExtension**.

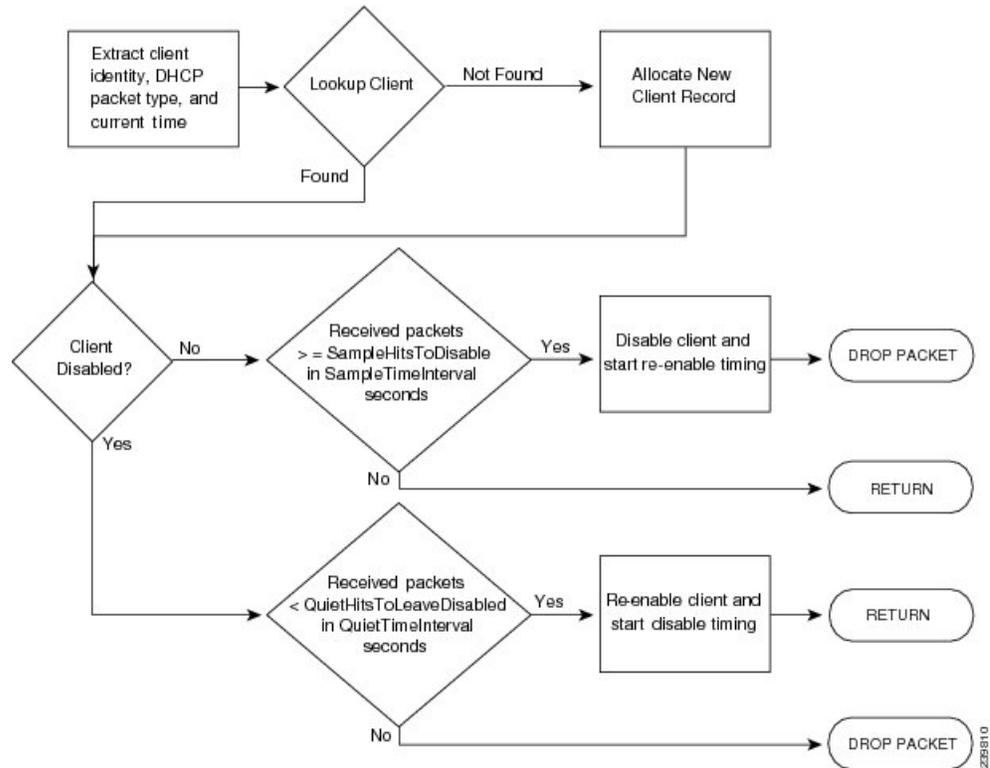
Step 4 Reload the server.

Preventing Chatty Clients by Using an Extension

One example of an effective use of an extension is to protect against clients flooding the server with unnecessary traffic. You can use the ChattyClientFilter extension to keep the server from having to do much of the work of processing these chatty client packets. If you have large numbers of clients in your network, you might want to consider implementing this extension.

The ChattyClientFilter extension is available in the `/examples/dhcp/dex` directory of the Cisco Prime Network Registrar installation, and compiled and ready to use in `/extensions/dhcp/dex/dextension.so` or `/extensions/dhcp/dex/dextension.dll`. The extension monitors client requests, based on the MAC address, and disables the client if it generates more than a certain number of packets in a time interval. Disabling a client means that the server discards packets from it. However, the server does not ignore the client entirely, because it continues to monitor traffic from it. If the server detects that the client starts to generate fewer than a certain number of packets in a time interval, it reenables the client and begins to allow packets from it again.

Figure 8: Chatty Client Filter Flow



The criteria for disabling and reenabling are set through arguments to the ChattyClientFilter extension. By default, the server disables a client when it receives more than 15 packets within 30 seconds; the server reenables the client when it sends fewer than 5 packets within 10 seconds. Note that these defaults are conservative and do not protect against all situations. For example, the server does not disable a client that sends packets every three seconds. Even allowing for a few retransmissions, a client should never need to send more than a half dozen packets in a short interval.

If you suspect chatty clients, review the DHCP server logs to determine incoming rates, then set the arguments described in the table below in the ChattyClientFilter code appropriately.

Table 2: ChattyClientFilter Arguments

ChattyClientFilter Argument	Description
-c	Ignores the packets when the “drop” attribute of the environment dictionary is set to “true”; default is not to ignore.
-d <i>packet-count seconds</i>	Drops DHCPRELEASE packets if more than the specified count are received in the specified time interval; default disabled. The server keeps dropping DHCPRELEASE packets until the client suspends sending them for the <i>specified</i> interval. (DHCPv4 clients only.) The basic formula is that the time interval should be at least $(packet-count + 2) * 30$ seconds.

ChattyClientFilter Argument	Description
<code>-h packet-count</code>	SampleHitsToDisable; default 15 packets.
<code>-i seconds</code>	SampleTimeInterval; default 30 seconds.
<code>-l packet-count</code>	QuietHitsToLeaveDisabled; default 5 packets.
<code>-m seconds</code>	Sets the maximum time a client is disabled, in seconds; default 0 - unlimited.
<code>-n</code>	<p>NAKs the client if renewing or rebinding; default off. If the client exceeding the SampleHitsToDisable rate does a DHCPREQUEST, the server sends it a DHCPNAK instead of discarding the packet.</p> <p>This can resolve problems with clients (such as cable modems) that cannot renew leases for some reason. Sending the DHCPNAK causes the client to restart its DHCP state machine and send a DHCPDISCOVER.</p> <p>If you use this argument, you must attach the ChattyClientFilter to the check-lease-acceptable extension point. (DHCPv4 clients only.)</p>
<code>-q seconds</code>	QuietTimeInterval; default 10 seconds.
<code>-r seconds</code>	StatisticsInterval; default 300 seconds (5 minutes). This argument controls the frequency of periodic logging of the number of clients disabled and reenabled.
<code>-s</code>	Silently discards dropped packets; default off.
<code>-w port</code>	Enables web access on specified port (only enabled over IPv4; specify negative port to not bind to 127.0.0.1).
<code>-4</code>	Filters only DHCPv4 packets; default is both.
<code>-6</code>	Filters only DHCPv6 packets; default is both.



Note The `-h`, `-i`, `-l`, and `-q` defaults are unlikely to be appropriate to most situations as these were designed to address a single type of misbehaving client. Using a longer interval and packet hit count for normal conditions will produce reasonable results. Values such as `-i 120 -h 8 -q 120 -l 8` would allow a client 8 packets over a 120 second period. A normal DHCPDISCOVER/OFFER/REQUEST/ACK is only 2 packets from a client. That is, the proper use of the ChattyClientFilter requires tuning these values for your particular network conditions. Use of the logscan tool which is available from the Cisco Prime Network Registrar download section on the Cisco website can help in analyzing client activity.

Review the comments in the ChattyClientFilter.cpp file for details on setting the arguments and enabling the extension. In most cases, you would attach it to the **post-packet-decode** extension point (along with **check-lease-acceptable** if you use the `-n` argument).

A sample use for the ChattyClientFilter is to drop DHCPRELEASE packets sent from a DHCPv4 client to prevent the lease history database from growing out of bounds, which can be the case with certain router configurations.

This scenario uses the `-d` argument. The setup on a Linux system might be:

```
nrcmd> extension dexChattyClientFilter create dex libdexextension.so
dexChattyClientFilter
init-entry=dexChattyClientFilterInitEntry
init-args="-d 2 120"
```

```
nrcmd> dhcp attachextension post-packet-decode dexChattyClientFilter
```

For Windows, replace libdexextension.so with dexextension.dll.

This setup results in the server dropping DHCPRELEASE packets if it receives more than two of these packets from the same client in a 120-second interval, and resuming DHCPRELEASEs processing when the client does not send a DHCPRELEASE for at least 120 seconds.

Cisco Prime Network Registrar 8.2 or later supports the mini-web server that can be used to obtain information about the clients being monitored or disabled (traffic being dropped) by the Chatty Client Filter. A typical request might be `http://127.0.0.1:<port>/report` entered in a web browser.

The web server supports the following requests:

- **status**—Returns a statistics report.
- **report**—Returns a statistics report and a full client report. The client report includes all clients currently being monitored and those that are disabled.
- **disabled-report**—Same as report except only the disabled clients are returned.
- **flush**—Same as report but all clients are REMOVED from the internal monitored and disabled list.
- **csv-client-list**—Returns the client list using CSV format (includes monitored and disabled clients).
- **csv-disabled-client-list**—Same as csv-client-list but only includes clients currently disabled.
- **xml-client-list**—Returns the client list using XML (includes monitored and disabled clients).
- **xml-disabled-client-list**—Returns the disabled client list using XML.



Note This web server is a very basic server implementation. It only supports the requests mentioned above.

Tuning the DHCP Server

Other helpful hints in tuning your DHCP performance include:

- Set the request (*max-dhcp-requests*) and response (*max-dhcp-responses*) buffers for optimal throughput. See [Table 1: DHCP Advanced Attributes](#) for details.
- Keep the *defer-lease-extensions* attribute enabled. This reduces writes to the database.
- Set the *last-transaction-time-granularity* attribute to at least 60 seconds, optimally a value greater than half your lease interval.
- Disable the *allow-lease-time-override* attribute for policies offering production leases.
- Minimize your logging and debugging settings. If you require logging, use the *log-settings* attribute for the DHCP server with a controlled number of attributes, as described in the table below.

Table 3: DHCP Log Settings

Log Setting (Numeric Equivalent)	Description
default (1)	Gives a low level of logging in several parts of the DHCP server. If you unconfigure the default, even this logging will not appear.
activity-summary (20)	Causes a summary message to appear every 1 minute. It is useful when many of the no-xxx log settings are enabled, to give some idea of the activity in the server without imposing the load required for a log message corresponding to each DHCP message. The time period for these messages can be configured with the DHCP server property <i>activity-summary-interval</i> .
client-criteria-processing (9)	Causes a log message to be output whenever a scope is examined to find an available lease or whenever a scope is examined to determine if a lease is still acceptable for a client who already has one. It can be very useful when configuring or debugging client-class scope criteria processing. It causes moderate amount of information to be logged and should not be left enabled as a matter of course.
client-detail (8)	Causes a single line to be logged at the conclusion of every client-class client lookup operation. This line will show the composite of the data found for the client as well as the data that found in the client's client-class. It is useful when setting up a client-class configuration and for debugging problems in client-class processing.
dns-update-detail (7)	Causes the server to log a message as it sends each DNS update and as it receives replies to update messages.
dropped-waiting-packets (15)	If the value of <i>max-waiting-packets</i> is non-zero, packets may be dropped if the queue length for any IP address exceeds the value of <i>max-waiting-packets</i> . If <i>dropped-waiting-packets</i> is set, the server will log a message whenever it drops a waiting packet from the queue for an IP address.
failover-detail (10)	Causes the server to log a single message for most failover transactions. The information logged is very useful for understanding how failover is operating, and should be included if at all possible when sending requests for support regarding failover issues.
incoming-packet-detail (4)	Causes the contents of every DHCP packet received by the DHCP server to be interpreted in a human readable way and printed in the log file. This enables the built-in DHCP packet sniffer for input packets. The log files will fill up (and turn over) very rapidly when this setting is enabled. This setting also causes a significant performance impact on the DHCP server and should not be left enabled as a matter of course.
incoming-packets (2)	This setting (on by default) causes a single line message to be logged for every incoming packet. This is especially useful when initially configuring a DHCP server or a BOOTP relay, in that an immediate positive indication exists that the DHCP server is receiving packets.

Log Setting (Numeric Equivalent)	Description
ldap-create-detail (13)	Causes log messages to appear whenever the dhcp server initiates a lease state entry create or delete to LDAP server, receives response and retrieves result or error messages.
ldap-query-detail (11)	Causes log messages to appear whenever the DHCP server initiates a query to LDAP server, receives response and retrieves result or error messages.
ldap-update-detail (12)	Causes log messages to appear whenever the DHCP server initiates an update lease state to LDAP server, receives response and retrieves result or error messages.
leasequery (14)	Causes log messages to appear when leasequery packets are processed without internal errors and result in an ACK or a NAK.
minimal-config-info (24)	Reduces the number of configuration messages printed when the server starts or reloads. In particular, it will not log a message for every scope.
missing-options (3)	This setting (on by default) causes a message to be logged whenever an option requested by a DHCP client has not been configured in a policy and therefore cannot be supplied by the DHCP server.
no-dropped-bootp-packets (18)	Causes the single line message normally logged for every BOOTP packet that is dropped to not appear.
no-dropped-dhcp-packets (17)	Causes a single line message normally logged for every DHCP packet that is dropped due to DHCP configuration to not appear. (See <i>no-invalid-packets</i> for messages associated with packets dropped because they are invalid.)
no-failover-activity (19)	Causes normal activity and some warning messages logged for failover to not appear. Serious error log messages will continue to appear independent of this log-setting.
no-failover-conflict (25)	Causes conflicts between failover partners to not be logged.
no-invalid-packets (21)	Causes a single line message normally logged for every DHCP packet that is dropped due to being invalid to not appear. (See <i>no-dropped-dhcp-packets</i> for messages associated with packets dropped due to DHCP server configuration.)
no-reduce-logging-when-busy (22)	Normally, the DHCP server will reduce logging when it becomes very busy (i.e., when it has used over 2/3 of the available receive buffers (itself a configurable value)). It will set <i>no-success-messages</i> , <i>no-dropped-dhcp-packets</i> , <i>no-dropped-bootp-packets</i> , <i>no-failover-activity</i> , <i>no-invalid-packets</i> , and clear everything else except <i>activity-summary</i> . If <i>no-reduce-logging-activity</i> is set, then the server will not do this. It will restore the previous settings when the server becomes unbusy (i.e., when it has used only 1/3 of the available receive buffers).
no-success-messages (16)	Causes the single line message that is normally logged for every successful outgoing DHCP response packet to not appear. It affects logging only for successful outgoing DHCP response packets.

Log Setting (Numeric Equivalent)	Description
no-timeouts (23)	Causes messages associated with timeout of leases or offers not to appear in the log file.
outgoing-packet-detail (5)	Causes the contents of every DHCP packet transmitted by the DHCP server to be interpreted in a human readable way and printed in the log file. This enables the built-in DHCP packet sniffer for output packets. The log files will fill up (and turn over) very rapidly when this setting is enabled. This setting also causes a significant performance impact on the DHCP server and should not be left enabled as a matter of course.
unknown-criteria (6)	Causes a single line log message to appear whenever a client entry is found which specifies selection criteria that is not found in any scope appropriate for that client's current network location.
v6-lease-detail (27)	Causes the server to log individual messages regarding DHCPv6 leasing activity (in addition to or in place of a single message per client transaction depending on no-success-messages, or client timeout event depending on no-timeouts).

- Consider setting client caching (see [Setting Client Caching Parameters, on page 296](#)).
- Check the server statistics to aid in monitoring server performance (see the *"Displaying Statistics"* section in *Cisco Prime Network Registrar 9.0 Administrator Guide*).
- Consider setting the scope allocation priority (see [Configuring Multiple Scopes Using Allocation Priority, on page 114](#)).
- If pinging hosts before offering addresses, consider adjusting the ping timeout period (see [Pinging Hosts Before Offering Addresses, on page 194](#)).
- To boost performance, consider limiting the number of selection tags.
- If using Lightweight Directory Access Protocol (LDAP) servers, consider the performance issues described in [Configuring Cisco Prime Network Registrar to Use LDAP, on page 301](#).
- If using DHCP failover, consider using the load balancing feature (see [Setting Load Balancing, on page 69](#)).



Tip Be sure to follow any DHCP server attribute changes with a server reload.

Listing Related Servers for DHCP - Failover, DNS, LDAP, and TCP Listener Servers

If you have related failover, DNS, LDAP, or TCP Listener servers (see [Setting Up Failover Server Pairs, on page 57](#)), you can access the attributes for these servers.

Local Web UI

On the Failover Pairs page, click the **Manage Failover Servers** tab and then click the **Related Servers** tab or click the **Related Servers** tab on the Manage DHCP Server page (**Operate > Servers > Manage Servers**) to open the DHCP Related Server Attributes page. This page shows the communication and failover states the servers are in. The following table describes the attributes on this page. (For this page to appear, you must be assigned the central-cfg-admin role with the dhcp-management subrole.)

Table 4: Attributes for Related Servers

Related Server Attribute	Description
<i>Related Server Type</i>	Type of related server: DHCP, DNS, or LDAP.
<i>Related Server IP Address</i>	IP address of the related server. For DHCP failover partners, click this link to open the View Failover Related Server page.
<i>Communications</i>	State of the communication—None, OK, or Interrupted.
<i>Requests</i>	Applies to DNS or LDAP related servers only, the number of requests from these servers.
<i>State</i>	For DHCP failover—None, Startup, Normal, Communications-interrupted, Partner-down, Potential-conflict, Recover, Paused, Shutdown, or Recover-done. For High-Availability (HA) DNS—Send-Update, Probe, or ha-state-unknown. Only the server that is successfully updating can be in Send-Update state. The partner server not sending updates is then always in Probe or unknown state. When the DHCP server comes up if there is no client activity, both DNS servers are often in the unknown state. This changes when the DHCP server tries to do DNS updates.
<i>Partner Role</i>	For DHCP failover only, the failover role of the partner—Main or Backup.
<i>Partner State</i>	For DHCP failover only, the partner's state—None, Startup, Normal, Communications-interrupted, Partner-down, Potential-conflict, Recover, Paused, Shutdown, or Recover-done.
<i>Update Response Complete</i>	For DHCP failover only, the percentage of completed update responses, valid only if there are outstanding update responses.

Table 5: Attributes for DHCP Related Failover Servers

DHCP Related Failover Server Attribute	Description
General attributes	
<i>failover-pair-name</i>	The name of the failover pair object used to manage this server.
<i>current-time</i>	Current time on the server returning this object.
<i>comm-state</i>	None, OK, or Interrupted.

DHCP Related Failover Server Attribute	Description
<i>smoothed-time-delta</i>	The time difference between the local server and the partner server. If the local server time is ahead of the partner server time, the attribute value is positive. If the local server time is behind the partner server time, the attribute value is negative. If the servers are not communicating, the last known attribute value is recorded.
<i>maximum-client-lead-time</i>	Current maximum client lead time (MCLT) on this system.
<i>sequence-number</i>	Sequence number unique across failover objects, if different from the sequence in the lease, the lease is considered “not up to date” independent of the sf-up-to-date lease flag.
<i>load-balancing-backup-pct</i>	The current failover load balancing backup percentage. If the backup percentage is zero, failover load balancing is not in use (disabled).
Local server information	
<i>our-ipaddr</i>	IPv4 address of the interface to this server.
<i>our-ip6address</i>	IPv6 address of the interface to this server.
<i>role</i>	Failover role of the server returning this object—None, Main, or Backup.
<i>state</i>	State of the local server—None, Startup, Normal, Communications-interrupted, Partner-down, Potential-conflict, Recover, Paused, Shutdown, or Recover-done.
<i>start-time-of-state</i>	Time at which the current failover state began.
<i>start-of-comm-interrupted</i>	Time at which this partner most recently went into communications-interrupted state. This is valid across reloads, while the start-time-of-state never has a time earlier than the most recent server reload.
<i>est-end-recover-time</i>	Valid if <i>update-request-in-progress</i> is not set to None. If it appears, the time at which the server enters the recover- done state if the update request outstanding is complete. If it does not appear, then the server enters recover-done whenever update-request is completed.
<i>use-other-available</i>	If false or unset, then this server cannot use other-available leases. If true, then the server can use other-available leases. Valid at all times, but should only be true if in partner-down state.
<i>use-other-available-time</i>	If, in partner-down state, the <i>use-other-available</i> is false or unset, the time when <i>use-other-available</i> will go to true.
<i>safe-period-remaining</i>	Duration in seconds remaining in safe-period. If not set to 0, then this server is currently running down a safe period with respect to its partner.

DHCP Related Failover Server Attribute	Description
<i>load-balancing-local-hba</i>	The current hash bucket assignment of the local server, usually shown as a range of the hash bucket numbers. (See RFC 3074.)
<i>request-buffers-in-use</i>	The number of failover request buffers the DHCP server is using at the time the statistics are calculated.
<i>decaying-max-request-buffers-in-use</i>	The maximum number of failover request buffers that have recently been in use.
<i>request-buffers-allocated</i>	The number of request buffers that the server has allocated to support the failover capability.
<i>connection-start-time</i>	The time at which the most recent connection started. This value is set whenever a connection is started, and it not cleared when a connection ended.
<i>connection-end-time</i>	The time at which the most recent connection ended. This value is set whenever a connection is ended, and it not cleared when a new connection starts.
Partner server information	
<i>ipaddr</i>	IP address of the partner server.
<i>ip6address</i>	IPv6 address of the partner server.
<i>partner-role</i>	Failover role of the partner of the server returning this object—None, Main, or Backup.
<i>partner-state</i>	Last known state which the partner end of the failover relationship is in—None, Startup, Normal, Communications-interrupted, Partner-down, Potential-conflict, Recover, Paused, Shutdown, or Recover-done.
<i>start-time-of-partner-state</i>	Time at which the partner current failover state began.
<i>est-partner-end-recover-time</i>	If the <i>partner-state</i> is Recover, an estimated prediction of when the partner will time out its MCLT and finish being in recover state.
<i>last-comm-ok-time</i>	Time at which this server last found communications to be OK.
<i>load-balancing-partner-hba</i>	The current hash bucket assignment of the partner server, usually shown as a range of the hash bucket numbers. (See RFC 3074.)
<i>partner-vendor-major-version</i>	The vendor ID major version from the partner server.
<i>partner-vendor-minor-version</i>	The vendor ID minor version from the partner server.
Update requests sent to partner	

DHCP Related Failover Server Attribute	Description
<i>update-request-outstanding</i>	If None or unset, then the server does not have an update request queued for its partner. If not set to None, then it does have an update request queued for its failover partner. Valid values are None, Update, and Update-all.
<i>update-request-start-time</i>	Time at which any <i>update-request-outstanding</i> request was started.
<i>update-request-done-time</i>	Time at which the last of any update request completed.
<i>v6-update-response-in-progress</i>	The type and origin of the response.
<i>v6-update-response-percent-complete</i>	The percent complete of the current IPv6 update response.
<i>v6-update-response-start-time</i>	The time that the IPv6 update response mentioned in <i>v6-update-response-in-progress</i> was started.
<i>v6-update-response-done-time</i>	The time that the most recent IPv6 update response sent an update done to the partner server.
Update requests processed for partner	
<i>update-response-in-progress</i>	If this server is processing an update response, gives information about the type and origin of the response.
<i>update-response-percent-complete</i>	If <i>update-response-outstanding</i> appears, the percent complete of the current update response.
<i>update-response-start-time</i>	Time that the update response mentioned in <i>update-response-in-progress</i> was started.
<i>update-response-done-time</i>	Time that the most recent update response sent an update done to the partner server.
Load Balancing Counters	
<i>load-balancing-processed-requests</i>	The number of server processed requests, both IPv4 and IPv6, subject to load balancing. This counter includes only the requests made after the latest transition of server to normal state.
<i>load-balancing-dropped-requests</i>	The number of server dropped requests, both IPv4 and IPv6, subject to load balancing. This counter includes only the requests made after the latest transition of server to normal state.
<i>load-balancing-processed-total</i>	The number of server processed requests, both IPv4 and IPv6, subject to load balancing. This counter includes the requests since this server was last started or reloaded.
<i>load-balancing-dropped-total</i>	The number of server dropped requests, both IPv4 and IPv6, subject to load balancing. This counter includes the requests since this server was last started or reloaded.

DHCP Related Failover Server Attribute	Description
Binding Update or Ack Counters (this connection)	
<i>binding-updates-sent</i>	The number of binding update (BNDUPD) messages sent to the failover partner.
<i>binding-acks-received</i>	The number of binding acknowledgement (BNDACK) messages received from the failover partner.
<i>binding-updates-received</i>	The number of binding update (BNDUPD) messages received from the failover partner.
<i>binding-acks-sent</i>	The number of binding acknowledgement (BNDACK) messages sent to the failover partner.
<i>v6-binding-updates-sent</i>	The number of IPv6 binding updates (BNDUPD6) messages received from the failover partner since the start of the most recently established connection.
<i>v6-binding-acks-received</i>	The number of IPv6 binding acknowledgements (BNDACK6) messages received from the failover partner since the start of the most recently established connection.
<i>v6-binding-updates-received</i>	The number of IPv6 binding updates (BNDUPD6) messages received from the failover partner since the start of the most recently established connection.
<i>v6-binding-acks-sent</i>	The number of IPv6 binding acknowledgements (BNDACK6) messages sent to the failover partner since the start of the most recently established connection.
<i>Binding Update/Ack Counters Totals</i>	
<i>binding-updates-sent-total</i>	The number of IPv4 binding updates (BNDUPD) messages sent to the failover partner since the most recent statistics reset.
<i>binding-acks-received-total</i>	The number of IPv4 binding acknowledgements (BNDACK) messages received from the failover partner since the most recent statistics reset.
<i>binding-updates-received-total</i>	The number of IPv4 binding updates (BNDUPD) messages received from the failover partner since the most recent statistics reset.
<i>binding-acks-sent-total</i>	The number of IPv4 binding acknowledgements (BNDACK) messages sent to the failover partner since the most recent statistics reset.
<i>v6-binding-updates-sent-total</i>	The number of IPv6 binding updates (BNDUPD6) messages sent to the failover partner since the most recent statistics reset.
<i>v6-binding-acks-received-total</i>	The number of IPv6 binding acknowledgements (BNDACK6) messages received from the failover partner since the most recent statistics reset.

DHCP Related Failover Server Attribute	Description
<i>v6-binding-updates-received-total</i>	The number of IPv6 binding updates (BNDUPD6) messages received from the failover partner since the most recent statistics reset.
<i>v6-binding-acks-sent-total</i>	The number of IPv6 binding acknowledgements (BNDACK6) messages sent to the failover partner since the most recent statistics reset.
<i>Flow Control Counters (this connection)</i>	
<i>current-binding-updates-in-flight</i>	The current number of binding updates (both IPv4 and IPv6) that are currently in-flight (sent).
<i>current-binding-updates-queued</i>	The current number of binding updates (both IPv4 and IPv6) that are queued at present.
<i>maximum-binding-updates-in-flight</i>	The maximum number of binding updates (both IPv4 and IPv6) that were in-flight (sent) at one time.
<i>maximum-binding-updates-queued</i>	The maximum number of binding updates (both IPv4 and IPv6) that were queued at one time.
<i>last-binding-update-sent-time</i>	The time the last binding update (either IPv4 or IPv6) was sent.
<i>last-binding-ack-received-time</i>	The time the last IPv4 or IPv6 binding acknowledgement (whether NAKed or not) was received.
<i>last-binding-update-received-time</i>	The time the last binding update (either IPv4 or IPv6) was received.
<i>last-binding-ack-sent-time</i>	The time the last IPv4 or IPv6 binding acknowledgement (whether NAKed or not) was sent.

Table 6: Attributes for DNS Related Failover Servers

DNS Related Server Attribute	Description
General attributes	
<i>current-time</i>	Current time on the server returning this object.
<i>ipaddr</i>	IP address
<i>comm-state</i>	There are three possible values: None, OK, or Interrupted. Status of communication between the DHCP and remote server. An 'OK' indicates DHCP server succeeded in communicating to the remote server. An 'Interrupted' indicates DHCP server was unsuccessful in communicating to the remote server.

DNS Related Server Attribute	Description
<i>dns-server-state</i>	There are two possible values: PROBE or SEND-UPDATE. PROBE means that the DHCP server has either not yet tried to communicate to this server, or it was determined to be down and hence it is actively being probed (this means that the DHCP server only sends one update request to it). SEND-UPDATE means that the server appears to be communicating and that the DHCP server can send it many requests.
<i>probe-polling-event-id</i>	Zero.
<i>requests</i>	Number of requests currently outstanding with the remote server.
HA DNS Configuration information	
<i>ha-dns-role</i>	Role played by this DNS server. The value can be STANDALONE-DNS, HA-MAIN, or HA-BACKUP. A DNS server can be a standalone DNS, or HA-Main or HA-Backup if HA-DNS is in use.
<i>dns-timeout</i>	Number of milliseconds that the DHCP server will wait for a response from the DNS server for a dynamic dns update, before retrying dynamic dns update.
<i>max-dns-retries</i>	Number of times that the DHCP server will try to send dynamic updates to a DNS server.
<i>ha-dns-failover-timeout</i>	Maximum time period, in seconds, the DHCP server will wait for a reply from a DNS server, before the DHCP will failover to use next DNS Server to perform the dynamic-update. Default value is 30 seconds.
<i>ha-dns-probe-timeout</i>	If cnr-ha-dns is enabled, DHCP server will use this timer to co-ordinate and reduce latency in failing over between HA-DNS servers, when HA-DNS servers are in COMMUNICATION-INTERRUPTED state or SYNCHRONIZING. Default value is 3 seconds.
<i>ha-dns-probe-retry</i>	If cnr-ha-dns is enabled, DHCP server will use this retry count and ha-dns-probe-timeout to co-ordinate and reduce latency in failing over between HA-DNS servers, when HA-DNS servers are in COMMUNICATION-INTERRUPTED state or SYNCHRONIZING. Default value is 1 retry attempt.

Table 7: Attributes for TCP Listener and Connection Related Servers

TCP Listener and Connection Related Server Attribute	Description
TCP Listener Related Server Attributes	
<i>ipaddr</i>	Address to which the listener is bound. This may be 0.0.0.0.
<i>comm-state</i>	Status of communication. This will always be none.

TCP Listener and Connection Related Server Attribute	Description
<i>ip6address</i>	IPv6 address to which the listener is bound. This may be 0::0.
<i>name</i>	Name of the service.
<i>port</i>	Port number to which the listener is bound. Incoming connections to this port will be processed.
<i>total-connections</i>	Total number of incoming connections.
<i>current-connections</i>	Number of currently active connections.
<i>rejected-connections</i>	Total number of incoming connections that were rejected, such as the maximum number of active connections were exceeded.
TCP Connection Related Server Attributes	
<i>ipaddr</i>	Address of the remote end of the connection.
<i>comm-state</i>	Status of communication. This will always be ok.
<i>ip6address</i>	IPv6 address of the remote end of the connection.
<i>name</i>	Name of the service on which this connection was accepted.
<i>port</i>	Port number for the remote end of the connection.
<i>total-requests</i>	Total number of request messages received.
<i>current-requests</i>	Number of active requests.
<i>current-state</i>	Current state of the connection.
<i>total-replies</i>	Total number of reply messages sent.
<i>start-time</i>	Time when the connection was established.
<i>last-receive-time</i>	Time of the last byte received.
<i>last-send-time</i>	Time of the last byte sent.
<i>total-bytes-received</i>	Total number of bytes received over the connection.
<i>total-bytes-sent</i>	Total number of bytes sent over the connection.
<i>our-ipaddr</i>	Address of the local end of the connection.
<i>our-ip6address</i>	IPv6 address of the local end of the connection.
<i>our-port</i>	Port number for the local end of the connection.

Other controls are available on these pages:

- To refresh the data on the Related Server tab, click **Refresh Data**.

- On the Related Server tab, if the partner is in the Communications-interrupted failover state, you can click **Set Partner Down** in association with an input field for the partner-down date setting. This setting is initialized to the value of the *start-of-communications- interrupted* attribute. (In Normal web UI mode, you cannot set this date to be an earlier value than the initialized date. In Expert web UI mode, you can set this value to any date.) After clicking **Set Partner Down**, you return to the List Related Servers for DHCP Server page to view the result of the partner-down action. Never set both partners to Partner Down mode.
- To return from the List Related Servers for DHCP Server page or View Failover Related Server page, click **Return**.

CLI Commands

To list the related servers for a DHCP server in a brief table form, with a subset of the values, use **dhcp getRelatedServers**. To report the full details (normal object form display, not table), use **dhcp getRelatedServers full**.

Configuring Virtual Private Networks

This section describes how to configure the Cisco Prime Network Registrar DHCP server to support virtual private networks (VPNs).

Configuring VPNs involves an adjustment to the usual DHCP host IP address designation. VPNs use private address spaces that might not be unique across the Internet. Because of this, Cisco Prime Network Registrar supports IP addresses that are distinguished by a VPN identifier. Relay agents on routers must support this capability as well. The VPN identifier selects the VPN to which the client belongs. VPN for DHCP is currently only supported by Cisco IOS software, the newest versions of which can include VPN IDs in the relayed DHCP messages.

Related Topics

[Configuring Virtual Private Networks Using DHCP, on page 43](#)

[VPN and Subnet Allocation Tuning Parameters, on page 50](#)

Configuring Virtual Private Networks Using DHCP

VPNs that you create provide a filtering mechanism for:

- Viewing the unified address space (see [Viewing Address Space, on page 100](#))
- Listing address blocks (see [Adding Address Blocks, on page 95](#))
- Listing subnets (see [Address Blocks and Subnets, on page 92](#))
- Querying subnet utilization (see [Generating Subnet Utilization History Reports, on page 105](#))
- Querying lease history (see [Running IP Lease Histories, on page 223](#))

If you do not configure a VPN, Cisco Prime Network Registrar uses the global VPN of 0 on each scope.

To configure a VPN whereby a client can request IP addresses from a DHCP server using a relay agent, you must define the VPN and associate a scope with it. Specifically:

1. Ensure that the relay agents that handle DHCP VPN traffic are configured with a version of Cisco IOS software that supports the *vpn-id* suboption of the *relay-agent-info* option (82) in DHCP.

2. Coordinate with the Cisco IOS relay agent administrator that the VPN is identified either by a VPN ID or a VPN Routing and Forwarding instance (VRF) name.
3. Create a scope for the VPN.

Related Topics

[Typical Virtual Private Networks, on page 44](#)

[Creating and Editing Virtual Private Networks, on page 45](#)

[VPN Usage, on page 46](#)

Typical Virtual Private Networks

[Figure 4: Virtual Private Network DHCP Configuration](#) shows a typical VPN scenario with DHCP client 1 as part of VPN blue and DHCP client 2 in VPN red. For example, both DHCP client 1 in VPN blue and client 2 in VPN red have the same private network address: 192.168.1.0/24. The DHCP relay agent has gateway addresses that are in the two VPNs as well as a global one (172.27.180.232). There are two failover DHCP servers, both of which know the relay agent through its external gateway address.

Here is the processing that takes place for the server to issue a VPN-supported address to a client:

1. DHCP client 1 broadcasts a DHCPDISCOVER packet, including its MAC address, hostname, and any requested DHCP options.
2. DHCP relay agent at address 192.168.1.1 picks up the broadcast packet. It adds a *relay-agent-info* option (82) to the packet and includes the *subnet-selection* suboption that identifies 192.168.1.0 as the subnet. The packet also includes the *vpn-id* suboption that identifies the VPN as *blue*. Because the DHCP server cannot communicate directly with the requesting client, the *server-id-override* suboption contains the address of the relay agent as known by the client (192.168.1.1). The relay agent also includes in the packet its external gateway address (*giaddr*), 172.27.180.232.
3. The relay agent unicasts the DHCPDISCOVER packet to the configured DHCP server on its subnet.
4. DHCP server 1 receives the packet and uses the *vpn-id* and *subnet-selection* suboptions to allocate an IP address from the proper VPN address space. It finds the available address 192.168.1.37 in the subnet and VPN, and places it in the *yiaddr* field of the packet (the address offered to the client).
5. The server unicasts a DHCPOFFER packet to the relay agent that is identified by the *giaddr* value.
6. The relay agent removes the *relay-agent-info* option and sends the packet to DHCP client 1.
7. DHCP client 1 broadcasts a DHCPREQUEST message requesting the same IP address that it was offered. The relay agent receives this broadcast message.
8. The relay agent forwards the DHCPREQUEST packet to DHCP server 1, which replies with a unicast DHCPACK packet to the client.
9. For a lease renewal, the client unicasts a DHCPRENEW packet to the IP address found in the *dhcp-server-identifier* option of the DHCPACK message. This is 192.168.1.1, the address of the relay agent. The relay agent unicasts the packet to the DHCP server. The server does its normal renewal processing, without necessarily knowing whether it was the server that gave out the original address in the first place. The server replies in a unicast DHCPACK packet. The relay agent then forwards the DHCPACK packet to the client IP address identified by the *ciaddr* field value.

If the *server-id-override* suboption of the *relay-agent-info* option (82) exists, the DHCP server uses its value to compare to that of the *dhcp-server-identifier* option in the reply packet. Any packet that the DHCP client unicasts then goes directly to the relay agent and not to the server (which may, in fact, be inaccessible from the client). Both partners in a failover environment can renew a lease if the packet includes the *server-id-override* suboption.

Creating and Editing Virtual Private Networks

To set up the VPN and its index:

- Step 1** Coordinate with the Cisco IOS relay agent administrator that the VPNs are configured either by VPN ID or VRF name on the relay agent. This will determine how to identify the VPN in Cisco Prime Network Registrar.
- Step 2** Create a VPN to allow provisioning DHCP clients onto the VPN that is configured in the IOS switch or router.
- Step 3** Enter a VPN index, which can be any unique text string except the reserved words **all** or **global**. Its associated ID must also be unique.

To add an index at the:

- **Local cluster (Advanced)**—From the **Design** menu, choose **VPNs** under the **DHCP Settings** submenu to open the List/Add VPNs page. Give the VPN a numerical key identifier and a unique name in the cluster.
- **Regional cluster**—Add the local cluster containing the VPN (from the **Operate** menu, choose **Manage Clusters** under the **Servers** submenu). Then choose **VPNs** from the **Design** menu. This opens the List/Add VPNs page. You can create the VPN on this page or pull the VPN from the local clusters:
 - If creating the VPN, give it a numerical key identifier and a unique name.
 - If pulling the VPN from the local clusters, click the **Pull Data** icon in the VPNs pane on the List/Add VPNs page, then pull a specific VPN or all the VPNs from the selected cluster.

You can also push VPNs to the clusters by clicking the **Push** or **Push All** icon in the List/Add VPNs page. Then choose the synchronization mode and the clusters to which to push the VPNs on the Push VPN Data to Local Clusters page.

- **In the CLI** —Use `vpn name create key`. For example:

```
nrcmd> vpn blue create 99
```

- Step 4** Specify the appropriate VPN identifier, either by VPN ID or VRF name. It is rarely both.
 - If you use a VPN ID, set the `vpn-id` attribute value for the VPN. The value is usually in hexadecimal, in the form `oui:index`, per IETF RFC 2685. It consists of a three-octet VPN Organizationally Unique Identifier (OUI) that corresponds to the VPN owner or ISP, followed by a colon. It is then followed by a four-octet index number of the VPN itself. Add the VPN ID value to the List/Add VPNs page. In the CLI, set the `vpn-id` attribute. For example:


```
nrcmd> vpn blue set vpn-id=a1:3f6c
```
 - If you use a VPN Routing and Forwarding (VRF) instance name, set the `vrf-name` attribute value for the VPN. Cisco routers frequently use VRF names. Add the VRF Name value to the List/Add VPNs page. In the CLI, set the `vrf-name` attribute. For example:


```
nrcmd> vpn blue set vrf-name=framus
```

- Step 5** Add a description for the VPN (optional).

- Step 6** Click **Add VPN**. You can edit the VPN to change the values on the Edit VPN page.

- Step 7** Create a scope for the VPN.

You must keep the VPN name and scope name as similar as possible for identification purposes.

1. In the web UI, from the **Design** menu, choose **Scopes** under the **DHCPv4** submenu to open the List/Add DHCP Scopes page.

2. Choose the VPN from the username drop-down list on the top right of the window. You cannot change the VPN once you set it at the time of creation of the scope

In the CLI, identify to which VPN the scope belongs in one of three ways:

- Its VPN name, through the *vpn* attribute (which applies the VPN ID to the scope).
- The VPN ID itself, through the *vpn-id* attribute.
- The current session VPN name, by omitting the VPN or its ID on the command line.

You set the default VPN for the current session using *session set current-vpn*. You can then set the usual address range and necessary option properties for the scope. For example:

```
nrcmd> scope blue-1921681 create 192.168.1.0 255.255.255.0 vpn=blue
```

Or

```
nrcmd> scope blue-1921681 create 192.168.1.0 255.255.255.0 vpn-id=99
```

Or

```
nrcmd> session set current-vpn=blue
nrcmd> scope blue-1921681 create 192.168.1.0 255.255.255.0
```

Then

```
nrcmd> scope blue-1921681 addRange 192.168.1.101 192.168.1.200
nrcmd> scope-policy blue-1921681 setOption routers 192.168.1.1
```

If you are in the staged dhcp edit mode, reload the DHCP server after you create all the VPNs and scopes.

VPN Usage

The VPN name is used to qualify many DHCP objects in Cisco Prime Network Registrar, such as IP addresses (leases), scopes, and subnets. For example, lease names can have this syntax:

vpn/ipaddress

For example, red/192.168.40.0

A VPN can be any unique text string except the reserved words **global** and **all**. You can use **global** and **all** when you lease data. The **global** VPN maps to the [none] VPN; the **all** VPN maps to both the specific VPN and the [none] VPN.

In the CLI, if you omit the VPN or its ID in defining an object, the VPN defaults to the value set by **session set current-vpn**. In the web UI, if the current VPN is not defined, it defaults to the [none] VPN, which includes all addresses outside of any defined VPNs.

These objects have associated VPN properties:

- **Address blocks**—Define the VPN for an address block. Choose **Address Blocks** from the **Design > DHCPv4** menu to open the List/Add DHCP Address Blocks page. Choose the VPN from the username > VPN menu at the top right of the window. In the CLI, use the **dhcp-address-block** creation and attribute setting commands. For example:

```
nrcmd> dhcp-address-block red create 192.168.50.0/24
```

```
nrcmd> dhcp-address-block red set vpn=blue
```

```
nrcmd> dhcp-address-block red set vpn-id=99
```



Note Before creating the objects, set the `vpn-id` value to the VPN in which the `dhcp-address-block` has to be created. Do not assume that `vpn-id` is always the current VPN.

- **Clients and client-classes**—In some cases it is best to provision a VPN inside of Cisco Prime Network Registrar instead of externally, where it might have to be configured for every Cisco IOS device. To support this capability, you can specify a VPN for a client or client-class. Two attributes are provided:
 - *default-vpn* —VPN that the packet gets if it does not already have a *vpn-id* or *vrf-name* value in the incoming packet. You can use the attribute with clients and client-classes.
 - *override-vpn* —VPN the packet gets no matter what is provided for a *vpn-id* or *vrf-name* value in the incoming packet. You can use the attribute with clients and client-classes. Note that if you specify an override VPN on the client-class, and a default VPN for the client, the override VPN on the client-class takes precedence over the default VPN on the client.

At the local cluster—Choose **Clients** or **Client Classes** (available in Advanced mode) from the **Design > DHCP Settings** menu. Create or edit a client-class or client and enter the *default-vpn* and *override-vpn* attribute values.

At the regional cluster—Choose **Client Classes** (available in Advanced mode) from the **Design > DHCP Settings** menu. Create or pull, and then edit a client-class to enter the *default-vpn* and *override-vpn* attribute values.

In the CLI—Use the **client-class** creation and attribute setting commands. For example:

```
nrcmd> client 1,6,00:d0:ba:d3:bd:3b set default-vpn=blue
```

```
nrcmd> client-class CableModem set override-vpn=blue
```

In a cable modem deployment, for example, you can use the *override-vpn* attribute to provision the cable modems. The client-class would determine the scope for the cable modem, and the scope would determine the VPN for the uBR. User traffic through the cable modem would then have the *vpn-id* suboption set and use the specific VPN. The *override-vpn* value also overrides any *default-vpn* set for the client.

- **Leases**—List leases, show a lease, or get lease attributes.

In the CLI—To import leases, use **import leases filename**. Each lease entry in the file can include the VPN at the end of the line. If it is missing, Cisco Prime Network Registrar assigns the [none] VPN. (See also [Importing and Exporting Lease Data, on page 192](#).)

```
nrcmd> import leases leaseimport.txt
```

To export the address or lease data to include the VPN, use **export addresses** with the *vpn* attribute, or **export leases** with the `-vpn` option. The VPN value can be the reserved word **global** or **all**:

- **Global**—Any addresses outside the defined VPNs (the [none] VPN).
- **All**—All VPNs, including the [none] VPN.

If you omit the VPN, the export uses the current VPN as set by **session set current-vpn**. If the current VPN is not set, the server uses the [none] VPN.

```
nrcmd> export addresses file=addrexport.txt vpn=red
nrcmd> export leases -server -vpn red leaseexport.txt
```

- **Scopes**—Scopes can include the VPN name or its ID, as described in [Configuring Virtual Private Networks Using DHCP, on page 43](#)



Note You cannot change the VPN once you set it at the time of creation of the scope.

- **Subnets**—Listing subnets, showing a subnet, or getting the *vpn* or *vpn-id* attribute for a subnet shows the VPN. See [Configuring DHCP Subnet Allocation, on page 49](#).
- **DHCP server**—If the *vpn-communication* attribute is enabled (which it is by default), the DHCP server can communicate with DHCP clients that are on a different VPN from that of the DHCP server by using an enhanced DHCP relay agent capability. This capability is indicated by the *server-id-override* suboption in the relay agent information option (82).



Note The DHCP server does not try to ping the clients residing in VPNs.

Configuring Subnet Allocation

This section describes how to configure the Cisco Prime Network Registrar DHCP server to support subnet allocation for on-demand address pools.

Subnet allocation is a way of leasing subnets to clients (usually routers or edge devices) so that they can, in turn, provide DHCP services. This can occur along with or instead of managing individual client addresses. Subnet allocation can vastly improve IP address provisioning, aggregation, characterization, and distribution by relying on the DHCP infrastructure to dynamically manage subnets. Subnet allocation through DHCP is currently only supported by Cisco IOS software, the newest versions of which incorporate the on-demand address pools feature.



Note DHCP failover does not include DHCPv4 subnet allocation.

Related Topics

[Configuring DHCP Subnet Allocation, on page 49](#)

[VPN and Subnet Allocation Tuning Parameters, on page 50](#)

Configuring DHCP Subnet Allocation

The following section provides an example of setting up subnet allocation using the DHCP server. [Figure 10: Sample DHCP Subnet Allocation Configuration](#) shows a sample subnet allocation configuration with subnets assigned to provisioning devices, along with the conventional DHCP client/server configuration.

Before allocating subnets, the DHCP server first determines what VPN the client is on, in the following order:

1. The server looks for incoming VPN options and uses the value for the VPN.
2. If no VPN options are found, the server uses the relay agent suboption value, then combines the VPN with the subnet address to form the unique identifier.
3. If no relay agent suboption is found, the server looks for client-class information (selection tags).

To configure DHCP subnet allocation:

Step 1

Create a DHCP address block for a subnet, set the initial subnet mask and its increment, and set other subnet allocation request attributes. Also, associate a policy or define an embedded policy.

- If you use VPNs, you can specify a *vpn* or *vpn-id* attribute (see [Configuring Virtual Private Networks Using DHCP, on page 43](#)).
- The server uses the presence of the *subnet-alloc* DHCP option (220) in the request packet to determine that the packet is a subnet allocation request. You can configure the server to use the *subnet-name* suboption (3) as a selection tag if you set the *addr-blocks-use-selection-tags* attribute for the server or VPN.
- You can optionally set a default selection tag by setting the *addr-blocks-default-selection-tags* attribute for the DHCP server or VPN object. This identifies one or more subnets from which to allocate the addresses. If the relay agent sends a VPN string (via a VPN option or relay agent suboption), associated with a subnet, any address block with that string as one of its *addr-blocks-default-selection-tags* values uses that subnet.
- The default behavior on the server and for VPNs is that the DHCP server tries to allocate subnets to clients using address blocks that the clients already used. Disabling the *addr-blocks-use-client-affinity* attribute causes the server to supply subnets from any suitable address block, based on other selection data in the clients' messages.
- If you want to support configurations of multiple address blocks on a single LAN segment (analogous to using primary and secondary scopes), add a *segment-name* attribute string value to the DHCP address block. When the relay agent sends a single subnet selection address, it selects address blocks tagged with that *segment-name* string value. However, you must also explicitly enable the LAN segment capability (*addr-blocks-use-lan-segments*) at the server or VPN level.
- Instead of associating a policy, you can set properties for the address block embedded policy. As in embedded policies for clients, client-classes, and scopes, you can enable, disable, set, unset, get, and show attributes for an address block policy. You can also set, unset, get, and list any DHCP options for it, as well as set, unset, and list vendor options. Note that deleting an address block embedded policy unsets all the embedded policy properties.

Step 2

Note that the server allocates subnets based on the relay agent request. If not requested, the default subnet size is a 28-bit address mask. You can change this default, if necessary, by setting the *default-subnet-size* attribute for the DHCP address block.

For example:

```
nrcmd> dhcp-address-block red set default-subnet-size=25
```

- Step 3** You can control any of the subnets the DHCP server creates from the address blocks. Identify the subnet in the form *vpn-name/netipaddress/mask*, with the *vpn-name* optional. Subnet control includes activating and deactivating the subnet as you would a lease. Likewise, you can force a subnet to be available, with the condition that before you do so, that you check that the clients assigned the subnet are no longer using it. First, show any subnets created.
- Step 4** Reload the DHCP server.
-

VPN and Subnet Allocation Tuning Parameters

Consider these tuning parameters for VPNs and on-demand address pools.

- **Keep orphaned leases that have nonexistent VPNs**—Cisco Prime Network Registrar usually maintains leases that do not have an associated VPN in the Cisco Prime Network Registrar state database. You can change this by enabling the DHCP attribute *delete-orphaned-leases*. The server maintains a lease state database that associates clients with leases. If a scope modification renders the existing leases invalid, the lease database then has orphaned lease entries. These are typically not removed even after the lease expires, because the server tries to use this data in the future to reassociate a client with a lease. One downside to this is that the lease database may consume excessive disk space. When you enable the *delete-orphaned-leases* attribute, such lease database entries are removed during the next server reload. However, be cautious when enabling this attribute, because rendering leases invalid can result in clients using leases that the server believes to be free. This can compromise network stability.
- **Keep orphaned subnets that have nonexistent VPNs or address blocks**—This is the default behavior, although you can change it by enabling the DHCP attribute *dhcp enable delete-orphaned-subnets*. As the DHCP server starts up, it reads its database of subnets and tries to locate the parent VPN and address block of each subnet. With the attribute enabled, if a subnet refers to a VPN that is no longer configured in the server, or if the server cannot locate a parent address block that contains the subnet, the server permanently deletes the subnet from the state database.
- **Keep the VPN communication open**—This is the default behavior, although you can change it by disabling the DHCP attribute *vpn-communication*. The server can communicate with clients that reside on a different VPN from that of the server by using an enhanced DHCP relay agent capability. This is signaled by the appearance of the *vpn-id* suboption of the *relay-agent-info* option (82). You can disable the *vpn-communication* attribute if the server is not expected to communicate with clients on a different VPN than the server. The motivation is typically to enhance network security by preventing unauthorized DHCP client access.

Configuring BOOTP

BOOTP (the BOOTstrap Protocol) was originally created for loading diskless computers. It was later used to allow a host to obtain all the required TCP/IP information to use the Internet. Using BOOTP, a host can broadcast a request on the network and get information required from a BOOTP server. The BOOTP server is a computer that listens for incoming BOOTP requests and generates responses from a configuration database for the BOOTP clients on that network. BOOTP differs from DHCP in that it has no concept of lease or lease expiration. All IP addresses that a BOOTP server allocates are permanent.

You can configure Cisco Prime Network Registrar to act like a BOOTP server. In addition, although BOOTP normally requires static address assignments, you can choose to either reserve IP addresses (and, therefore, use static assignments) or have IP addresses dynamically allocated for BOOTP clients.

Related Topics

- [About BOOTP, on page 51](#)
- [Enabling BOOTP for Scopes, on page 52](#)
- [Moving or Decommissioning BOOTP Clients, on page 52](#)
- [Using Dynamic BOOTP, on page 52](#)
- [BOOTP Relay, on page 53](#)

About BOOTP

When you configure the DHCP server to return a BOOTP packet, be aware that BOOTP requires information in the DHCP packet in fields other than the option space. BOOTP devices often need information in the boot file (*file*), server IP address (*siaddr*), and server hostname (*sname*) fields of the DHCP packet (see RFC 2131).

Every Cisco Prime Network Registrar DHCP policy has attributes with which you can configure the information you want returned directly in the *file*, *siaddr*, or *sname* fields. The Cisco Prime Network Registrar DHCP server also supports a configuration parameter with which you can configure the policy options and determine which of the *file*, *sname*, or *siaddr* values you want returned to the BOOTP device.

Cisco Prime Network Registrar supports an analogous configuration parameter with which you can configure the options and *file*, *sname*, or *siaddr* values you want returned to the DHCP client. This is in addition to any options requested by the DHCP clients in the *dhcp-parameter-request* option in the DHCP request. Thus, you can configure both the BOOTP and DHCP response packets appropriately for your devices.

-
- Step 1** Decide which values you want for the BOOTP attributes:
- *file*—Name of the boot file
 - *siaddr*—Server IP address
 - *sname*—Optional server hostname
- Step 2** Decide the list of options and their values that you want returned to the BOOTP client.
- Step 3** Set these values in the policy you want associated with the BOOTP request:
- Attributes (*packet-siaddr*, *packet-file-name*, *packet-server-name*) to send to the BOOTP client.
 - Option values, such as the server addresses and domain name to return to the BOOTP client.
 - List of fields and options you want returned to the BOOTP client.
- Step 4** Enable the associated scope or scopes for BOOTP processing.
- Step 5** Enable dynamic BOOTP processing if you want to have this scope provide an address for any BOOTP client that requests one. If you do not enable dynamic BOOTP, you must make reservations for each BOOTP client for which you want this scope to provide an address.
-

Enabling BOOTP for Scopes

You can enable BOOTP processing for a scope. Set certain attributes and BOOTP reply options for a created policy in the local cluster web UI, or use **policy name create** and **policy name set** in the CLI, to configure BOOTP. Set the policy attributes and options as a comma-separated list. The attributes are entities to use in a client boot process:

- **packet-siaddr**—IP address of the next server
- **packet-file-name**—Name of the boot file
- **packet-server-name**—Hostname of the server

The server looks through the policy hierarchy for the first instances of these attribute values.

In the CLI, **policy name setOption** requires spaces (not equal signs) before values.

Also, enable BOOTP and dynamic BOOTP, if desired, and ensure that the DHCP server updates the DNS server with BOOTP requests. The options are:

- Set the option *dhcp-lease-time*.
- Enable the *dynamic-bootp* attribute.
- Enable the *update-dns-for-bootp* attribute.
- Enable the *update-dns-for-bootp* attribute.

Moving or Decommissioning BOOTP Clients

When you move or decommission a BOOTP client, you can reuse its lease. To decommission a BOOTP client, you must remove its lease reservation from the scope and force its lease to be available.

Force the lease to be available in the local cluster web UI, or set **scope name removeReservation** and **lease ipaddr force-available** in the CLI.

Using Dynamic BOOTP

When you use dynamic BOOTP, there are additional restrictions placed on the address usage in scopes, because BOOTP clients are allocated IP addresses permanently and receive leases that never expire.

If you are using DHCP failover, when a server whose scope does not have the *dynamic-bootp* option enabled goes into PARTNER-DOWN state, it can allocate any available IP address from that scope, no matter whether it was initially available to the main or backup server. However, when the *dynamic-bootp* option is enabled, the main server and backup servers can only allocate their own addresses. Consequently scopes that enable the *dynamic-bootp* option require more addresses to support failover.

When using dynamic BOOTP:

1. Segregate dynamic BOOTP clients to a single scope. Disable DHCP clients from using that scope. In the local cluster web UI, under the BOOTP attributes for the scope, disable the *dhcp* attribute. In the CLI, use **scope name disable dhcp**.
2. If you are using DHCP failover, set the *failover-dynamic-bootp-backup-percentage* attribute for the DHCP server to allocate a greater percentage of addresses to the backup server for this scope. This percentage can be as much as 50 percent higher than a regular backup percentage.

BOOTP Relay

Any router that supports BOOTP relay usually has an address that points to the DHCP server. For example, if you are using a Cisco router, it uses the term *IP helper-address*, which contains an address for a specific machine. In this case, use this address to forward all BOOTP (and therefore DHCP) broadcast packets. Be sure that you configure this address on the router closest to your host.



Tip If your DHCP clients are not receiving addresses from the DHCP server, check the network configuration, particularly the router or relay agent configuration, to verify that your network devices are set up to point to your Cisco Prime Network Registrar DHCP server address.



CHAPTER 3

Managing DHCP Failover

Cisco Prime Network Registrar failover protocol is designed to allow a backup DHCP server to take over for a main server if the main server is taken offline for any reason. Prior to 8.2, this protocol was UDP based, only operated over IPv4, and only supported DHCPv4. Starting with 8.2, this protocol is TCP based, can be configured to use either IPv4 or IPv6, and supports both DHCPv4 and DHCPv6 over a single connection. With 9.0, this protocol now will try both IPv4 and IPv6 transports if configured to use both, and will use whichever connection comes up first. The DHCP failover supports the following features:

- DHCPv4 addresses
- DHCPv6 addresses (non-temporary and temporary)
- DHCPv6 prefix delegation

DHCP failover is not applicable to DHCPv4 subnet allocation (on-demand address pools).

- [How DHCP Failover Works, on page 55](#)
- [DHCP Simple Failover, on page 56](#)
- [DHCPv6 Failover, on page 56](#)
- [Setting Up Failover Server Pairs, on page 57](#)
- [Configuring Failover Parameters Based on Your Scenario, on page 64](#)
- [Recovering from a DHCP Failover, on page 70](#)
- [Setting Advanced Failover Attributes, on page 75](#)
- [Maintaining Failover Server Pair, on page 76](#)
- [Recovering Failover Configuration, on page 76](#)
- [Using PARTNER-DOWN State to Allow a Failover Server to Operate for Extended Periods without Its Failover Partner, on page 77](#)
- [Restoring a Standalone DHCP Failover Server - Tutorial , on page 78](#)
- [Changing Failover Server Roles, on page 84](#)
- [Troubleshooting Failover, on page 86](#)
- [Supporting BOOTP Clients in Failover, on page 88](#)

How DHCP Failover Works

DHCP failover is based on a server-partner relationship. The partner must have identical DHCPv4 scopes, DHCPv6 prefixes, DHCPv6 links, reservations, policies, and client-classes, as the server. After the servers start up, they contact each other. The main server provides its partner with a DHCPv4 addresses and DHCPv6 delegated prefixes, and updates its partner with every client operation. If the main server fails, then the partner takes over offering and renewing leases, using its DHCPv4 addresses and DHCPv6 delegated prefixes. When

the main server becomes operational again, it re-integrates with its partner without administrative intervention. These servers are in a relationship known as a failover pair.

The failover protocol keeps DHCP operational, if:

- **The main server fails**—The partner takes over services during the time the main server is down. The servers cannot generate duplicate addresses, even if the main server fails before updating its partner.
- **Communication fails**—A partner can operate correctly even though it cannot tell whether it was the other server or the communication with it that failed. The servers cannot issue duplicate addresses, even if they are both running and each can communicate with only a subset of clients.

After a failover pair is configured:

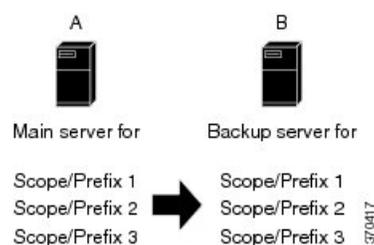
1. The partners connect.
2. The main server supplies data about all existing leases to its partner.
3. The backup server requests a pool of backup addresses from the main server.
4. The main server replies with a percentage of available addresses from each scope or prefix to its partner.
5. The backup server ignores all DHCPDISCOVER and Solicit requests, unless it senses that the main server is down or load balancing is enabled for the failover pair. In normal operation, the backup server handles only some renewal and rebinding requests.
6. The main server updates its partner with the results of all client operations.

You can automatically synchronize the configuration of the servers in a failover pair. The two servers dynamically rebalance the available leases; if the main server hands out a large portion of its available leases, it can reclaim leases from its partner.

DHCP Simple Failover

Starting from release 8.2, Cisco Prime Network Registrar supports only simple failover configuration. Simple failover involves a single main server and a single backup server pair (see the image below). In the example, main server A has three scopes or prefixes that must be configured identically on backup server B.

Figure 9: Simple Failover Example



DHCPv6 Failover

DHCPv6 failover works very similar to the DHCPv4 simple failover configuration. The DHCPv6 failover partners keep each other updated on stateful address and delegated prefix leases that are granted, perform synchronization when communication is restored, and generally follow and adhere to the DHCPv4 failover protocol requirements (except the differences between DHCPv4 and DHCPv6).

The maximum client lead time (MCLT) and lease time restrictions are applied to DHCPv6 leases and both the valid lifetime and preferred lifetime of leases are limited to MCLT defined for the failover pair. Only

when the longest lease time allowed by the failover pair exceeds the configured preferred lifetime and if the configured preferred lifetime is less than the configured valid lifetime, the preferred lifetime and valid lifetime of the lease can be different. The delegated prefixes are managed and balanced similar to DHCPv4 addresses.

The most significant difference is that the DHCPv6 failover servers do not balance the available addresses on each prefix but instead use an algorithm to determine which new addresses each server can lease. The algorithm uses the least significant bit of the address and the main server assigns odd addresses whereas the backup server assigns even addresses. This applies to client requested and randomly generated addresses and is not applicable if:

- A lease is already assigned to the client.
- A reservation exists for the client.
- The allocation-algorithms interface-identifier is set and is used. In this case, the interface-identifier (EUI-64) bit is assumed to be unique, and as the global bit is set, these addresses do not conflict with randomly generated addresses as these never have the global bit set.
- Client reservations are configured on the prefix.
- An extension supplies the address.

Setting Up Failover Server Pairs

You can create and synchronize failover pairs at the local and regional clusters.

A failover pair has two main elements, its configuration and the state information that the servers maintain. The key configuration attributes are the name of the failover pair, the role of the local server (main or backup), and the address of the partner. The failover state is defined when you reload the server and the server processes this state data at startup.



Note Cisco Prime Network Registrar 8.2 or later DHCP failover does not interoperate with Cisco Prime Network Registrar 8.1 or earlier releases DHCP failover. You must upgrade both the main and the backup servers in the same maintenance window.

Related Topics

[Adding Failover Pairs, on page 57](#)

[Synchronizing Failover Pairs, on page 61](#)

[Failover Checklist, on page 63](#)

Adding Failover Pairs

Create the DHCP failover pair based on the cluster of the main and backup servers. Then synchronize the configuration of the failover pair so that the scopes, prefixes, policies, and other DHCP properties match between the servers.

To add a failover pair:

Local and Regional Web UI

-
- Step 1** From the **Deploy** menu, choose **Failover Pairs** under the **DHCP** submenu to open the List/Add DHCP Failover Pairs page.
- Step 2** Click the **Add Failover Pair** icon in the **Failover Pairs** pane.
- Step 3** On the Add DHCP Failover Pair dialog box, add a failover pair name.
This is required and can be any distinguishing name. (See [Changing Failover Pair Names, on page 76.](#))
- Step 4** Choose the cluster for the main DHCP server. This can be localhost or some other cluster you define.
- Step 5** Choose the cluster for the backup DHCP server. This cannot be the same as the main server cluster, but it must be localhost if the main cluster is not localhost.
- Step 6** Click **Add DHCP Failover Pair**. The failover pair is created.
- Step 7** You can set additional attributes, such as the maximum client lead time (*mclt*) or backup percentage (*backup-pct*). Most of the default values are optimized. Leave the *failover* attribute enabled by default unless you want to temporarily disable failover for the pair.

You may specify the *main-server*, *backup-server*, *main-ip6address*, or *backup-ip6address* attributes if you want to override the values that are configured for the main and backup cluster objects, or if you want to disable a specific transport (by specifying 0.0.0.0 or 0::0 for the addresses). If both IPv4 and IPv6 addresses are available, failover will attempt to connect on both transports and use the connection that comes up first. Click **Save** to save these additional changes.

Following attributes can be configured on the Edit DHCP Failover Pair page (in Advanced mode):

Table 8: Failover Pair Attributes

Attribute	Description
Main Server (<i>main</i>)	Identifies the cluster with the main server for a failover pair.
Backup Server (<i>backup</i>)	Identifies the cluster that contains the backup server for a failover pair.
Scope Template (<i>scopetemplate</i>)	Associates a scope template with a specified failover pair.
Failover Settings	
<i>failover</i>	Enables failover configuration. If you disable this attribute, you turn off failover on attached subnets without changing configuration fundamentals.
<i>mclt</i>	Sets the maximum client lead time in seconds. This attribute controls how far ahead of the backup server that you can make the client lease expiration. You must define this value on both the main and backup servers, and make sure the value is identical on both servers.

Attribute	Description
<i>backup-pct</i>	Controls the percentage of available addresses that the main server sends to the backup server. Set this value on the main server. If it is set on a backup server, it is ignored (to enable copying of configurations). Unless you explicitly set this value on a scope and you disable load balancing, the value set here becomes the default value.
<i>dynamic-bootp-backup-pct</i>	<p>Determines the percentage of available addresses that the main server sends to the backup server for scopes on which dynamic BOOTP is enabled. If defined, it must be defined on the main server. If it is defined in a backup server, it is ignored (to enable copying of configurations). If it is not defined at all or the value is 0, the <i>backup-pct</i> is used instead. This parameter is separate from "backup-pct" because if dynamic BOOTP is enabled on a scope, a server will never, even in PARTNER-DOWN state, grant leases on addresses that are available to the other server because they can never safely be assumed to be available again.</p> <p>The MCLT has no meaning for dynamic BOOTP leases.</p>
<i>load-balancing</i>	Determines whether load balancing (RFC 3074) is enabled on a failover pair. The default is disabled. When enabled, the <i>backup-pct</i> is ignored and the main and backup server evenly split the client load and available leases for all scopes in the failover relationship (that is, as if backup-pct were configured at 50%).
<i>safe-period</i>	Controls the safe period, in seconds. It does not have to be the same on both main and backup servers. It only has meaning if <i>use-safe-period</i> is enabled.
<i>use-safe-period</i>	Controls whether a server can enter PARTNER-DOWN state without an operator command. If disabled, a server never enters PARTNER-DOWN without an operator command.
Failover Server Addresses	
<i>main-server</i>	<p>Controls the IPv4 address used for the failover protocol on the main server. If this value is unset, the address specified for the main cluster is used. Cisco recommends setting this value only if the server is configured with different interfaces for configuration management and clients requests.</p> <p>This value may be set to 0.0.0.0 to disable use of IPv4 for failover communication.</p> <p>If both IPv4 and IPv6 addresses are available, the servers will try both transports for the TCP connection and use whichever comes up first.</p>

Attribute	Description
<i>backup-server</i>	<p>Controls the IPv4 address used for the failover protocol on the backup server. If this value is unset, the address specified for the backup cluster is used. Cisco recommends setting this value only if the server is configured with different interfaces for configuration management and clients requests.</p> <p>This value may be set to 0.0.0.0 to disable use of IPv4 for failover communication.</p> <p>If both IPv4 and IPv6 addresses are available, the servers will try both transports for the TCP connection and use whichever comes up first.</p>
<i>main-ip6address</i>	<p>Controls the IPv6 address used for the failover protocol on the main server. If this value is unset, the address specified for the main cluster is used. Cisco recommends setting this value only if the server is configured with different addresses for configuration management and clients requests.</p> <p>This value may be set to 0::0 to disable use of IPv6 for failover communication.</p> <p>If both IPv4 and IPv6 addresses are available, the servers will try both transports for the TCP connection and use whichever comes up first.</p>
<i>backup-ip6address</i>	<p>Controls the IPv6 address used for the failover protocol on the backup server. If this value is unset, the address specified for the backup cluster is used. Cisco recommends setting this value only if the server is configured with different addresses for configuration management and clients requests.</p> <p>This value may be set to 0::0 to disable use of IPv6 for failover communication.</p> <p>If both IPv4 and IPv6 addresses are available, the servers will try both transports for the TCP connection and use whichever comes up first.</p>

CLI Commands

Use **failover-pair** *name* **create** *main-cluster backup-cluster* [*attribute=value ...*]. For example:

```
nrcmd> failover-pair example-fo-pair create Example-cluster Boston-cluster
```

Related Topics

[Failover Checklist, on page 63](#)

[Changing Failover Pair Names, on page 76](#)

[Synchronizing Failover Pairs, on page 61](#)

[Restarting the Failover Servers, on page 76](#)

Synchronizing Failover Pairs

Once you create the failover pairs, you must synchronize the failover pair configuration.

Web UI

Step 1 From the **Deploy** menu, choose **Failover Pairs** under the **DHCP** submenu to open the List/Add DHCP Failover Pairs page.

Step 2 Select the failover pair on the Failover pane.

Step 3 On the List/Add DHCP Failover Pairs page, click the **Synchronize DHCP Failover Pair** tab.

For synchronization in the regional web UI, see the *"Managing DHCP Failover Pairs" section in Cisco Prime Network Registrar 9.0 Administrator Guide*.

Step 4 Choose the direction of synchronization. The direction of synchronization can be either from main to backup server or from backup to main server.

Step 5 Choose the synchronization operation, depending on the degree to which you want the main server objects to replace those of the backup server. The following are the basic synchronization operations that can be performed on the servers:

- **Update operation**—This is the default and least radical operation. It is appropriate for update synchronizations in that it has the least effect on the unique properties of the backup server.
- **Complete operation**—This operation is appropriate for all initial synchronizations. It is more complete than an update operation, while still preserving many of the backup server unique properties.
- **Exact operation**—This operation is appropriate for simple failover configuration.

It makes the two servers mirror images of each other, as much as possible, although this operation retains the unique DHCP server, and extension points on the backup server.

Note For initial failover configurations, use the Exact or Complete operation.

For a better understanding of the functions that are performed on the classes of the objects, consider the following example. Here, we have a main server and its backup server with the following objects:

On the main server	On the backup server
Name1=A	Name2=B
Name2=C	Name3=D

Note In this example, we consider failover synchronization from the main server to the backup server.

Each operation performs a different mix of functions on the classes of objects. The following are the four functions that are performed on the objects based on the operation selected.

- **no change**—Makes no change to the list of properties or their values on the backup server.
For example, the result would be Name2=B, Name3=D.
- **ensure**—Ensures that a copy of the main server object exists on the backup. The target server objects with the same name as main server objects are left unchanged, the objects that are not on the target server are added to it, and the objects only on the target server are left unchanged.

For example, the result would be Name1=A, Name2=B, Name3=D.

- **replace**—Ensures that the existing object in the target server is replaced by the main server object of the same name. Also the objects that are not on the target server are added to it and the objects only on the target server are left unchanged. The only exceptions to this are for policies and option definition sets, where the option lists are extracted to compare the list entries.

For example, the result would be Name1=A, Name2=C, Name3=D.

Note After deleting the client on the main server and performing the failover synchronization Update or Complete operation to remove the entry on the backup, the client is not removed from the backup. The only failover synchronization operation that will delete the client entry on the backup, after it is removed from the main server, is the failover synchronization Exact operation.

- **exact**—Puts an exact copy of the main server object on the backup server and removes the unique ones. That is, the objects of target server are made identical to the objects of main server.

For example, the result would be Name1=A, Name2=C.

For more information, see table below. This table provides the information on the functions (no change, ensure, replace, or exact) that are performed on the objects based on the operations (Update, Complete, Exact) you select.

Table 9: Failover Pair Synchronization Functions

Data Description	Update	Complete	Exact
DHCP Server: Client-Class Properties Client Host Name Processing Properties Dynamic DNS Properties Failover Tuning Properties	replace	replace	replace
All other properties	no change	replace	replace
LDAP Remote Server	ensure	replace	exact
Policy: Option List Properties Packet Boot File Properties All other properties	ensure ensure replace	replace replace replace	replace replace replace
Client	replace	replace	exact
Client-Class	replace	replace	exact
Scopes	exact	exact	exact
Links	exact	exact	exact
Prefixes	exact	exact	exact
Reservations	exact	exact	exact
DNS Update Configuration	replace	replace	exact
Trap Configuration	ensure	replace	exact

Data Description	Update	Complete	Exact
VPN	replace	replace	exact
Key	replace	replace	exact
Extensions (You must copy extension files.)	ensure	replace	exact
Extension Point	replace	replace	replace
Option Information: Custom options list Vendor options list	ensure	replace	exact
DHCP Listener Configuration	ensure	replace	exact

Step 6 Click **Report** on the Synchronize DHCP Failover Pair page:

- When you click **Report**, the resulting View DHCP Failover Pair Sync Report page shows what change entries the synchronization will apply if you run the synchronization. You have the option to choose the direction of synchronization and also the option to check the desired mode of synchronization operation (**Update**, **Complete**, **Exact**). Check the desired values and click **Report**.

Step 7 Click **Save** on the View DHCP Failover Pair Sync Report page.

Step 8 On the List/Add DHCP Failover Pairs page, click the Manage Failover Servers tab.

Step 9 Click the **Restart Server** icon to reload the backup server.

Step 10 Try to get a lease.

Step 11 On the Manage Failover Servers tab, look at the health of the servers. Also, click the Logs tab to view the log entries on the Log for Server page, and ensure that the servers are in NORMAL failover mode. The log file should contain an item similar to the following:

```
06/19/2003 9:41:19 name/dhcp/1 Info Configuration 0 04092 Failover is enabled server-wide. Main
server name: '192.168.0.1',
backup server name: '192.168.0.110', mclt = 3600, backup-pct = 10, dynamic-bootp-backup-pct = 0,
use-safe-period: disabled,
safe-period = 0.
```

CLI Commands

Use **failover-pair** *name* **sync** {**update** | **complete** | **exact**} [{**main-to-backup** | **backup-to-main**}]:

```
nrcmd> failover-pair example-fo-pair sync exact main-to-backup
```

Failover Checklist

Once you create the failover pair, you must synchronize the configuration of the failover servers. Use this checklist to prepare for an effective failover configuration:

- Duplicate the DHCPv4 scope, DHCPv6 prefix, DHCPv6 links, reservations (IPv4 and IPv6), selection tags, policy, DHCP option, IP addresses, client-classes, dynamic DNS updates, dynamic BOOTP, VPN, DHCP extensions, DHCP extensions, LDAP server, and address configurations on the partner servers by synchronizing a failover server pair for a simple failover scenario.
- Ensure that both partners are configured with a wide enough range of addresses so that the backup server can provide leases for a reasonable amount of time while the main server is down.
- If you use BOOTP (DHCP) relay agents (IP helpers), configure all BOOTP relay agents to point to both partners. Cisco Prime Network Registrar does not automatically detect this.

You can detect BOOTP configuration errors only by performing live tests in which you periodically take the main server out of service to verify that the backup server is available to DHCP clients.

Configuring Failover Parameters Based on Your Scenario

Following are the advanced failover properties that are important to set:

- Backup percentage (see [Setting Backup Percentages, on page 64](#))
- Backup allocation boundaries (see [Setting Backup Allocation Boundaries, on page 75](#))
- Maximum client lead time (MCLT) (see [Setting the Maximum Client Lead Time, on page 65](#))
- Safe period (see [Using the Failover Safe Period to Move Servers into PARTNER-DOWN State, on page 67](#))
- Request and response packet buffers (see [Setting DHCP Request and Response Packet Buffers, on page 69](#))
- Load balancing (see [Setting Load Balancing, on page 69](#))

Setting Backup Percentages

To keep failover partners operating despite a network partition (when both servers can communicate with clients, but not with each other), allocate more addresses than the addresses for a single server. Configure the main server to allocate a percentage of the currently available addresses in each scope and prefix delegation prefixes to the backup server. This makes these addresses unavailable to the main server. The backup server uses these addresses when it cannot talk to the main server and cannot tell if it is down.

If the main server detects that the address pool is significantly out of balance or the server has no leases, then the pool of available or other-available leases are rebalanced even when the failover pair is functioning in the Normal state. The failover pair must be carefully monitored during failover and if the failover partner is down for an extended period then operator intervention may be required to move the failover partner to the PARTNER-DOWN state.

You can set the percentage of currently available addresses by setting the *backup-pct* attribute on the failover pair or DHCPv4 scope (**failover-pair name set fail backup-pct** or **scope name set backup-pct** in the CLI). The default backup percentage is 50%. DHCPv6 prefix delegation prefixes are fixed at 50% for the backup-pct equivalent.

Note that setting the backup percentage on the failover pair level sets the value for all scopes not set with that attribute. However, if set at the scope level, the backup percentage overrides the one at the failover pair level. If the *load-balancing* attribute is enabled for the failover pair (**failover-pair name enable load-balancing** in

the CLI), the backup percentage is fixed at 50% and any of the backup percentage attributes (on a failover pair or scope) are ignored.

The backup percentage should be set large enough to allow the backup server to continue serving new clients in the event that the main server fails. The backup percentage is calculated based on the number of available addresses. The backup percentage can safely be set to a larger value, if extended outages are expected, because the main server periodically reclaims addresses (once per hour) if, in the course of normal leasing activity, the main server's available address pool drops below its predefined percentage. For example, if backup percentage is set to 60%, the main server will reclaim addresses if its address pool falls below 60%.



Note When failover load balancing is in effect, the main and backup servers actively move available leases between them to maintain the backup percentage of available leases. See the [Setting Load Balancing, on page 69](#).

The percentage depends on the new client arrival rate and the network operator reaction time. It depends on the arrival rate of new DHCP clients and the reaction time of your network administration staff. The backup server needs enough addresses from each scope to satisfy all new clients requests arriving during the time it does not know if the main server is down. Even during PARTNER-DOWN state, the backup server waits for the maximum client lead time (MCLT) and lease time to expire before reallocating leases. See the [Setting the Maximum Client Lead Time, on page 65](#). When these times expire, the backup server offers:

- Leases from its private pool.
- Leases from the main server pool.
- Expired leases to new clients.

During the working hours, an operator likely responds within two hours to COMMUNICATIONS-INTERRUPTED state to determine if the main server is working. The backup server then needs enough addresses to support a reasonable upper bound on the number of new clients that could arrive during those two hours.

During off-hours, the arrival rate of previously unknown clients is likely to be less. The operator can usually respond within 12 hours to the same situation. The backup server then needs enough addresses to support a reasonable upper bound on the number of clients that could arrive during those 12 hours.

The number of addresses over which the backup server requires sole control is the greater of the two numbers. You can express this number as a percentage of the currently available (unassigned) addresses in each scope. If you use client-classes, remember that some clients can only use some sets of scopes and not others.



Note During failover, clients can sometimes obtain leases whose expiration times are shorter than the amount configured. This is a normal part of keeping the server partners synchronized. Typically this happens only for the first lease period, or during COMMUNICATIONS-INTERRUPTED state.

Related Topics

[BOOTP Backup Percentage, on page 89](#)

Setting the Maximum Client Lead Time

You can set a property for the failover pair that controls an adjustment to the lease period, the maximum client lead time (MCLT). The MCLT adjusts for a potential period of uncertain connectivity between the servers.

It is the maximum time one server can grant (or extend) a lease to a client without first negotiating a longer time with its partner. This time has the following implications:

- Clients may initially (or if the partners are not communicating) only receive leases of the MCLT length. This means that they need to renew leases sooner than they might otherwise without failover. At this renewal, the client should get a full lease time (unless the partners are not communicating).
- If a server enters PARTNER-DOWN state, it must wait until the MCLT after the later of the partner-down time or the latest lease expiration time communicated with the partner gets over. The latest lease expiration time communicated to the partner is typically 1.5 times the lease time from the last client lease request before communication was interrupted.
- If a failover recovery occurs where there is uncertainty about what one partner did (such as when it loses its lease database), the partners may have to restrict leasing activity for the MCLT period after they synchronize before they can resume normal failover operations.

The default MCLT is one hour, the optimum for most configurations. As defined by the failover protocol, the lease period given for a client can never be more than the MCLT plus the most recently received potential expiration time from the failover partner, or the current time, whichever is later. That is why you sometimes see the initial lease period as only an hour, or an hour longer than expected for renewals. The actual lease time is recalculated when the main server comes back.

The MCLT is necessary because of failover use of lazy updates. Using lazy updates, the server can issue or renew leases to clients before updating its partner, which it can then do in batches of updates. If the server goes down and cannot communicate the lease information to its partner, the partner may try to reoffer the lease to another client based on what it last knew the expiration to be. The MCLT guarantees that there is an added window of opportunity for the client to renew. The way that a lease offer and renewal works with the MCLT is:

1. The client sends a DHCPDISCOVER or DHCPv6 Solicit to the server, requesting a desired lease period (for example, three days). The server responds with a DHCPOFFER or DHCPv6 Advertise with an initial lease period of only the MCLT (one hour by default). The client then requests the MCLT lease period and the server acknowledges it.
2. The server sends its partner a bind update containing the lease expiration for the client as the current time plus the MCLT. The update also includes the potential expiration time as the current time plus the client desired period plus half of the client desired period ($3 + 1.5 = 4.5$ days). The partner acknowledges the potential expiration, thereby guaranteeing the transaction.
3. When the client sends a renewal request halfway through its lease (in one-half hour), the server acknowledges with the client desired lease period (3 days). The server then updates its partner with the lease expiration as the current time plus the desired lease period (3 days), and the potential expiration time (4.5 days. See the description in **Step 2**). The partner acknowledges this potential expiration of 4.5 days. In this way, the main server tries to have its partner always lead the client in its understanding of the client lease period so that it can always offer it to the client.

There is no one correct value for the MCLT. There is an explicit trade-off between various factors in choosing one. Most people use the preset value of one hour effectively and it works well in almost all environments. Here are some of the trade-offs between a short and long MCLT:

- **Short MCLT**—A short MCLT value means that after entering PARTNER-DOWN state, a server only has to wait a short time before it can start allocating its partner IP addresses to DHCP clients. Furthermore, it only has to wait a short time after a lease expires before it can reallocate that address to another DHCP client. However, the down side is that the initial lease interval that is offered to every new DHCP client will be short, which causes increased traffic, because those clients need to send their first renewal in a half of a short MCLT time. Also, the lease extensions that a server in COMMUNICATIONS-INTERRUPTED state can give is the MCLT only after the server has been in

that state for around the desired client lease period. If a server stays in that state for that long, then the leases it hands out will be short, increasing the load on that server, possibly causing difficulty.

- **Long MCLT**—A long MCLT value means that the initial lease period will be longer and the time that a server in COMMUNICATIONS-INTERRUPTED state can extend leases (after it being in that state for around the desired client lease period) will be longer. However, a server entering PARTNER-DOWN state must wait the longer MCLT before being able to allocate its partner addresses to new DHCP clients. This may mean that additional addresses are required to cover this time period. Also, the server in PARTNER-DOWN state must wait the longer MCLT from every lease expiration before it can reallocate an address to a different DHCP client.

Using the Failover Safe Period to Move Servers into PARTNER-DOWN State

One or both failover partners could potentially move into COMMUNICATIONS-INTERRUPTED state. They cannot issue duplicate addresses while in this state. However, having a server in this state over longer periods is not a good idea, because there are restrictions on what a server can do. The main server cannot reallocate expired leases and the backup server can run out of addresses from its pool.

COMMUNICATIONS-INTERRUPTED state was designed for servers to easily survive transient communication failures of a few minutes to a few days. A server might function effectively in this state for only a short time, depending on the client arrival and departure rate. After that, it would be better to move a server into PARTNER-DOWN state so it can completely take over the lease functions until the servers resynchronize.

There are two ways a server can move into PARTNER-DOWN state:

- **User action**—An administrator sets a server into PARTNER-DOWN state based on an accurate assessment of reality. The failover protocol handles this correctly. Never set both partners to PARTNER-DOWN.
- **Failover safe period expires**—When the servers run unattended for longer periods, they need an automatic way to enter PARTNER-DOWN state.

Network operators might not sense in time that a server is down or uncommunicative. Hence, the failover safe period, which provides network operators some time to react to a server moving into COMMUNICATIONS-INTERRUPTED state. During the safe period, the only requirement is that the operators determine that both servers are still running, and if so, fix the network communications failure or take one of the servers down before the safe period expires.

The length of the safe period is installation-specific, and depends on the number of unallocated addresses in the pool and the expected arrival rate of previously unknown clients requiring addresses.

In Cisco Prime Network Registrar 8.2 or later, the use-safe-period attribute is enabled by default for a failover pair and the default safe period is 4 hours. This ensures that if the failover partner is in COMMUNICATIONS-INTERRUPTED state for 4 hours, it will enter PARTNER-DOWN state automatically after the safe period elapses. You may need to review if this setting is appropriate for your network and adjust the safe-period based on your network requirements.

In addition, during this safe period, either server allows renewals from any existing client, but there is a major risk of possibly issuing duplicate addresses. This is because one server can suddenly enter PARTNER-DOWN state while the other is still operating. In order to prevent this problem, it is important that you do not change the default settings for use-safe-period and put operational procedures in place to alert operations personnel when the failover pair loses contact with each other. Especially, in the event of network communications failure, operator intervention is required before the safe period elapses. Either one failover server needs to be taken offline or the use-safe-period attribute needs to be disabled on both the servers before the safe period elapses.



Note In Cisco Prime Network Registrar 8.2 or later, use-safe-period is enabled by default. You may want to review if this is appropriate for your network and you may want to disable the use-safe-period or adjust the safe-period based on your network requirements and monitoring.

The number of extra addresses required for the safe period should be the same as the expected total of new clients a server encounters. This depends on the arrival rate of new clients, not the total outstanding leases. Even if you can only afford a short safe period, because of a shortage of addresses or a high arrival rate of new clients, you can benefit substantially by allowing DHCP to ride through minor problems that are fixable in an hour. There is minimum chance of duplicate address allocation, and reintegration after the solved failure is automatic and requires no operator intervention.

In Cisco Prime Network Registrar 8.2 or later, if the failover safe period length is more than the length of the MCLT and the failover server enters into PARTNER-DOWN state because of the safe-period, the server can start allocating its partner other-available leases to DHCP clients immediately. The advantage of this is that the server has additional leases to allocate. However, the disadvantage is that operator intervention is required within the safe period in case of network communications failure. Either the failover server needs to be taken offline or the use-safe-period attribute needs to be disabled on both the servers before the safe period elapses. Without operator intervention, both failover servers will transition to PARTNER-DOWN state and start allocating its partner addresses to new DHCP clients.

Here are some guidelines to follow, to help you decide whether to use manual intervention or the safe period for transitioning to PARTNER-DOWN state:

- If your corporate policy is to have minimal manual intervention, set the safe period. Enable the failover pair attribute *use-safe-period* to enable the safe period. Then, set the DHCP attribute *safe-period* to set the duration (4 hours by default). Set this duration long enough so that operations personnel can explore the cause of the communication failure and assure that the partner is truly down.
- If your corporate policy is to avoid conflict under any circumstances, then never let either server go into PARTNER-DOWN state unless by explicit command. Allocate sufficient addresses to the backup server so that it can handle new client arrivals during periods when there is no administrative coverage. You can set PARTNER-DOWN on the Manage Failover Servers tab of the web UI, if the partner is in the Communications-interrupted failover state, you can click **Set Partner Down** in association with an input field for the PARTNER-DOWN date setting. This setting is initialized to the value of the *start-of-communications-interrupted* attribute. (In Normal web UI mode, you cannot set this date to be an earlier value than the initialized date. In Expert web UI mode, you can set this value to any date.)

Use **failover-pair name setPartnerDown** date in the CLI, specifying the name of the partner server. This moves all the scopes running failover with the partner into PARTNER-DOWN state immediately, unless you specify a date and time with the command. This date and time should be when the partner was last known to be operational.

In Cisco Prime Network Registrar 8.2 or later, if you use setPartnerDown in the CLI and specify the date and time when the partner was last known to be operational then the failover server calculates the MCLT from the time specified in the setPartnerDown command. If the date and time is not specified for the setPartnerDown command, then the failover server calculates the MCLT from the time the failover server moved to the COMMUNICATIONS-INTERRUPTED state. In case of network communications failure, it is important that you specify the actual time the partner was last known to be operational in the setPartnerDown command. Otherwise, it can result in duplicate IP addresses.

There are two conventions for specifying the date:

- `-num unit` (a time in the past), where `num` is a decimal number and `unit` is `s`, `m`, `h`, `d`, or `w` for seconds, minutes, hours, days or weeks respectively. For example, specify `-3d` for three days.
- Month (name or its first three letters), day, hour (24-hour convention), year (fully specified year or last two digits). This example notifies the backup server that its main server went down at 12 o'clock midnight on October 31, 2002:

```
nrcmd> failover-pair dhcp2.example.com. setPartnerDown -3d
```

```
nrcmd> failover-pair dhcp2.example.com. setPartnerDown Oct 31 00:00:00 2001
```



Note Wherever you specify a date and time in the CLI, enter the time that is local to the `nrcmd` process. If the server is running in a different time zone than this process, disregard the time zone where the server is running and use local time instead.

Setting DHCP Request and Response Packet Buffers

DHCP failover allows a limited number of binding updates to be outstanding (set through the (expert mode) `max-unacked-bndupd failover-pair` attribute). The default value of `max-unacked-bndupd` is 1/5th (20%) of the `max-dhcp-requests` value and also it is at least the min of 100 and `max-dhcp-requests`. The server allocates additional request buffers to accommodate failover (as it must have these resources available for failover).

Setting Load Balancing

In normal failover mode, the main DHCP server bears most of the burden of servicing clients when the failover partners are in NORMAL communication mode. The main server not only services all new client requests, but has to handle renewal and rebinding requests and expired leases from the backup partner. To distribute the load more evenly between the two servers in a simple failover configuration scenario, Cisco Prime Network Registrar introduced the load balancing feature (based on RFC 3074).

Failover load balancing allows both servers to actively service clients and determine which unique clients each will serve without running the risk of both servicing the same ones. Failover load balancing applies only while the servers are in NORMAL mode; in other states, both servers can respond to clients.

According to RFC 3074, the servers calculate a hash value for each request that the server receives, based on the client identifier option value or hardware address. The request is serviced if the hash value is assigned to that server.

With failover load balancing enabled, the servers split the client load evenly. The main partner processes 50% of the hash values and the backup partner the other 50%.

While the failover partners periodically balance the available leases on the backup server or do so shortly after a scope or prefix is detected to be out of leases, enabling the `rebalancing-delta-pct` attribute (Expert mode) on the main server to set the percentage difference between the desired and actual available leases on the backup server that will trigger a rebalancing on the scope or prefix.

Each partner responds to all clients whenever a partner is not in NORMAL mode. Each partner responds only to the broadcast DHCPDISCOVER or SOLICIT messages from clients that are in their assigned hash values.

For broadcast DHCPREQUEST or REBIND messages, the server responds only if it is the targeted one (based on the server identifier option); so, if the targeted server is the main server and it is down, the backup does not service the client (unless you release the lease). Broadcast BOOTP, DHCPINFORM, and INFORMATION-REQUEST requests are also load-balanced.

Related Topics

[Configuring Load Balancing, on page 70](#)

Configuring Load Balancing

In the web UI, when setting the failover properties for the pair (see the [Setting Up Failover Server Pairs, on page 57](#)), enable or disable the *load-balancing* attribute in the Failover Settings attributes as desired to enable or disable failover load balancing. In the CLI, use **failover-pair *name* set load-balancing**.



Note You must restart the DHCP server on both main and backup to apply the changes.

Recovering from a DHCP Failover

During normal operation, the failover partners undergo transition between states. If one of the failover server fails, then the partner takes over offering and renewing leases, using its private pool. When the main server becomes operational again, it re-integrates with its partner without administrative intervention.

The following sections describe how to confirm a DHCP failover, monitor DHCP failover event, what happens when servers enter various states, and how the servers integrate.

Confirming Failover

To confirm the failover:

-
- Step 1** Ping from one server to the other to verify TCP/IP connectivity. Make sure that routers are configured to forward clients to both servers.
 - Step 2** Check that the server is in NORMAL mode by clicking the **Related Servers** icon on the Manage DHCP Server or List/Add DHCP Failover Pairs page, or use **dhcp getRelatedServers** in the CLI.
 - Step 3** After startup, have a client attempt to get a lease.
 - Step 4** Set the log settings on the main server to include at least *failover-detail*.
 - Step 5** Confirm that the name_dhcp_1_log log file (in *install-path /logs*) on the main server contains DHCPBNDACK or DHCPBNDUPD messages (for IPv4) and BNDUPD6 or BNDACK6 messages (for IPv6) from each server.
 - Step 6** Confirm that the name_dhcp_1_log log file on the backup server contains messages that the backup server is dropping requests because failover is in NORMAL state.
 - Step 7** Repeat **Step 2**.
-

Related Topics

[State Transitions During Integration, on page 73](#)

[Configuring Failover Parameters Based on Your Scenario, on page 64](#)

Monitoring DHCP Failover

When the main failover server goes down, the backup server moves to COMMUNICATIONS-INTERRUPTED state. The backup server cannot determine whether the main server is down or whether it cannot contact with the backup server. Depending upon the nature of outage you should monitor situation and follow the following steps:

1. Monitor both the failover servers and take action immediately when the main server goes down.
2. When the backup server first takes over, attempt to get the main server operational.
3. If you succeed in getting the main server operational within the MCLT, then nothing more needs be done.
4. If the main server is not operational until the MCLT has expired, then move the backup server to PARTNER DOWN state. On the backup server, use failover-pair name setPartnerDown date in the CLI.
5. When the main server is operational, ensure that it can contact the backup server before it is restarted.

For more information, see [State Transitions During Integration, on page 73](#).

Failover States and Transitions

During normal operation, the failover partners undergo transition between states. They stay in their current state until all the actions for the state transition are completed. If communication fails, they stay in their current state until the conditions for the next state are fulfilled. The states and their transitions are described in [Table 10: Failover States and Transitions , on page 71](#).

Table 10: Failover States and Transitions

State	Server Action
STARTUP	Tries to contact its partner to learn its state, then transitions to another state after a short time, typically seconds.
NORMAL	Can communicate with its partner. The main and backup servers act differently in this state: <ul style="list-style-type: none"> • The main server responds to all client requests using its pool. If its partner requests a backup pool, the main server provides it. • The backup server only responds to renewal and rebinding requests. It requests a backup pool from the main server.

State	Server Action
COMMUNICATIONS-INTERRUPTED	<p>Cannot communicate with its partner, whether it or the communication with it is down. The servers cycle between this state and NORMAL state as the connection fails and recovers, or as they cycle between operational and nonoperational. During this time, the servers cannot give duplicate addresses.</p> <p>During this state, you usually do not need to intervene and move a server into the PARTNER-DOWN state. However, this is not practical in some cases. A server running in this state is not using the available pool efficiently. This can restrict the time a server can effectively service clients.</p> <p>A server is restricted in COMMUNICATIONS-INTERRUPTED state:</p> <ul style="list-style-type: none"> • It cannot reallocate an expired address to another client. • It cannot offer a lease or renewal beyond the maximum client lead time (MCLT) longer than the current lease time. The MCLT is a small additional time added that controls how much the client lease expiration is ahead of what the backup server thinks it is. • A backup server can run out of addresses to give new clients, because it normally has only a small pool, while the main server has most of them. <p>The server is limited only by the number of addresses allocated to it and the arrival rate of new clients. With a high new client arrival or turnover rate, you may need to move the server into PARTNER-DOWN state more quickly.</p>
PARTNER-DOWN	<p>Acts as if it were the only operating server, based on one of these facts:</p> <ul style="list-style-type: none"> • The partner notified it during its shutdown. • The administrator put the server into PARTNER-DOWN state. • The safe period expired and the partner automatically went into this state. <p>In this state, the server ignores that the other server might still operate and could service a different set of clients. It can control all its addresses, offer leases and extensions, and reallocate addresses. The same restrictions to servers in COMMUNICATIONS-INTERRUPTED state do not apply.</p> <p>Either server can be in this state, but only one should be in it at a time so that the servers do not issue duplicate addresses and can properly resynchronize later on. Until then, an address is in a pending-available state.</p>
POTENTIAL-CONFLICT	<p>Might be in a situation that does not guarantee automatic reintegration, and is trying to reintegrate with its partner. The server might determine that two clients (who might not be operating) were offered and accepted the same address, and tries to resolve this conflict.</p>
RECOVER	<p>Has no data in its stable storage, or is trying to reintegrate after recovering from PARTNER-DOWN state, from which it tries to refresh its stable storage. A main server in this state does not immediately start serving leases again. Because of this, do not reload a server in this state.</p>
RECOVER-DONE	<p>Can transition from RECOVER or PARTNER-DOWN state, or from COMMUNICATIONS-INTERRUPTED into NORMAL state.</p>

State	Server Action
PAUSED	Can inform its partner that it will be out of service for a short time. The partner then transitions to COMMUNICATIONS-INTERRUPTED state and begins servicing clients.
SHUTDOWN	Can inform its partner that it will be out of service for a long time. The partner then transitions to PARTNER-DOWN state to take over completely.

State Transitions During Integration

During normal operation, the failover partners transition between states. They stay in their current state until all the actions for the state transition are completed, and if communication fails, until the conditions for the next state are fulfilled. The table below describes what happens when servers enter various states and how they initially integrate and later reintegrate with each other under certain conditions.

Table 11: Failover State Transitions and Integration Processes

Integration	Results
Into NORMAL state, the first time the backup server contacts the main server	<ol style="list-style-type: none"> 1. The newly configured backup server contacts the main server, which starts in PARTNER-DOWN state. 2. Because the backup server is a new partner, it goes into RECOVER state and sends a Binding Request message to the main server. 3. The main server replies with Binding Update messages that include the leases in its lease state database. 4. After the backup server acknowledges these messages, the main server responds with a Binding Complete message. 5. The backup server goes into RECOVER-DONE state. 6. Both servers go into NORMAL state. 7. The backup server sends Pool Request messages. 8. The main server responds with the leases to allocate to the backup server based on the <i>backup-pct</i> configured.
After COMMUNICATIONS-INTERRUPTED state	<ol style="list-style-type: none"> 1. When a server comes back up and connects with a partner in this state, the returning server moves into the same state and then immediately into NORMAL state. 2. The partner also moves into NORMAL state.
After PARTNER-DOWN state	<p>When a server comes back up and connects with a partner in this state, the server compares the time it went down with the time the partner went into this state.</p> <ul style="list-style-type: none"> • If the server finds that it went down and the partner subsequently went into this state: <ol style="list-style-type: none"> 1. The returning server moves into RECOVER state and sends an Update Request message to the partner. 2. The partner returns all the binding data it was unable to send earlier and follows up with an Update Done message. 3. The returning server moves into RECOVER-DONE state. 4. Both servers move into NORMAL state.

Integration	Results
	<ul style="list-style-type: none"> • If the returning server finds that it was still operating when the partner went into PARTNER-DOWN state: <ol style="list-style-type: none"> 1. The server goes into POTENTIAL-CONFLICT state, which also causes the partner to go into this state. 2. The main server sends an update request to the backup server. 3. The backup server responds with all unacknowledged updates to the main server and finishes off with an Update Done message. 4. The main server moves into NORMAL state. 5. The backup server sends the main server an Update Request message requesting all unacknowledged updates. 6. The main server sends these updates and finishes off with an Update Done message. 7. The backup server goes into NORMAL state.
After the server loses its lease state database	<p>A returning server usually retains its lease state database. However, it can also lose it because of a catastrophic failure or intentional removal.</p> <ol style="list-style-type: none"> 1. When a server with a missing lease database returns with a partner that is in PARTNER-DOWN or COMMUNICATIONS- INTERRUPTED state, the server determines whether the partner ever communicated with it. If not, it assumes to have lost its database, moves into RECOVER state, and sends an Update Request All message to its partner. 2. The partner responds with binding data about every lease in its database and follows up with an Update Done message. 3. The returning server waits the maximum client lead time (MCLT) period, typically one hour, and moves into RECOVER-DONE state. For details on the MCLT, see the Setting the Maximum Client Lead Time, on page 65. 4. Both servers then move into NORMAL state.
After a lease state database backup restoration	<p>When a returning server has its lease state database restored from backup, and if it reconnects with its partner without additional data, it only requests lease binding data that it has not yet seen. This data may be different from what it expects.</p> <ol style="list-style-type: none"> 1. In this case, you must configure the returning server with the <i>failover-recover</i> attribute set to the time the backup occurred. 2. The server moves into RECOVER state and requests all its partner data. The server waits the MCLT period, typically one hour, from when the backup occurred and goes into RECOVER-DONE state. For details on the MCLT, see the Setting the Maximum Client Lead Time, on page 65. 3. Once the server returns to NORMAL state, you must unset its <i>failover-recover</i> attribute, or set it to zero. <pre>nrcmd> dhcp set failover-recover=0</pre>

Integration	Results
After the operational server had failover disabled	<p>If the operating server had failover enabled, disabled, and subsequently reenabled, you must use special considerations when bringing a newly configured backup server into play. The backup server must have no lease state data and must have the <i>failover-recover</i> attribute set to the current time minus the MCLT interval, typically one hour. For details on the MCLT, see the Setting the Maximum Client Lead Time, on page 65.</p> <ol style="list-style-type: none"> 1. The backup server then knows to request all the lease state data from the main server. Unlike what is described in “After the server loses its lease state database” section of this table, the backup server cannot request this data automatically because it has no record of having ever communicated with the main server. 2. After reconnecting, the backup server goes into RECOVER state, requests all the main server lease data, and goes into RECOVER-DONE state. 3. Both servers go into NORMAL state. At this point, you must unset the backup server <i>failover-recover</i> attribute, or set it to zero. <pre>nrcmd> dhcp set failover-recover=0</pre>

Setting Advanced Failover Attributes

The advanced failover properties that are important to set are the following:

- Setting backup allocation boundaries (see [Setting Backup Allocation Boundaries, on page 75](#))
- DHCPLEASEQUERY and failover (see [DHCPLEASEQUERY and Failover, on page 75](#))

Setting Backup Allocation Boundaries

You can be more specific as to which addresses to allocate to the backup server by using the *failover-backup-allocation-boundary* attribute on the scope. The IP address set as this value is the upper boundary of addresses from which to allocate addresses to a backup server. Only addressees below this boundary are allocated to the backup. If there are no addresses available below this boundary, then the addresses above it, if any, are allocated to the backup. The actual allocation works down from this address, while the normal allocation for DHCP clients works up from the lowest address in the scope.

If you set *failover-backup-allocation-boundary* for the scope, you must also enable the *allocate-first-available* attribute. If *failover-backup-allocation-boundary* is unset or set to zero, then the boundary used is halfway between the first and last addresses in the scope ranges. If there are no available addresses below this boundary, then the first available address is used.

DHCPLEASEQUERY and Failover

To accommodate DHCPLEASEQUERY messages sent to a DHCP failover backup server when the master server is down, the master server must communicate the *relay-agent-info* (82) option values to its partner server. To accomplish this, the master server uses DHCP failover update messages.

Maintaining Failover Server Pair

This section describes how to maintain failover server pair and perform the following administrative tasks:

- Changing failover pair names (see [Changing Failover Pair Names, on page 76](#))
- Restarting the failover servers (see [Restarting the Failover Servers, on page 76](#))

Changing Failover Pair Names

Use `failover-pair old-name set name=new-name` to change the name of the failover pair. In the web UI, you will have to remove and then create a new object (do this without reloading the DHCP server until the new object is ready).

**Note**

If a cluster role in a failover relationship is changed (main to backup or backup to main), any existing state information for that relationship is discarded.

Restarting the Failover Servers

For any failover synchronization to take effect, you must first connect to, and restart, both the main and backup failover servers.

-
- Step 1** On the List/Add DHCP Failover Pairs page, select the failover pair on the Failover pane.
 - Step 2** On the Manage Failover Servers tab for the main server, select the server you want to restart.
 - Step 3** Choose Restart Server from the Quick View menu.
-

Related Topics

[Confirming Failover, on page 70](#)

Recovering Failover Configuration

When you upgrade Cisco Prime Network Registrar to the latest version, you can revert to the earlier version, in case the upgrade fails. You can upgrade one partner and when it has recovered to normal state and is working well, then upgrade the other partner.

You may be able to recover from the archive created during the upgrade, but if the upgrade is scheduled during a maintenance window, then you need to:

- Stop Cisco Prime Network Registrar completely using `nwreglocal stop`.
- Tar up the Cisco Prime Network Registrar DATADIR (`/var/nwreg2/local/data`) and save it in a safe location.
- Upgrade the server.

If it fails, then you need to:

- Stop Cisco Prime Network Registrar completely using `nwreglocal stop`.
- Delete the corrupt version of Cisco Prime Network Registrar DATADIR (The location is: `/var/nwreg2/local/data`).
- Extract the saved Cisco Prime Network Registrar DATADIR tarfile in the path the tarfile came from.
- Install the original version of Cisco Prime Network Registrar, which finds the existing DATADIR and use it.

Using PARTNER-DOWN State to Allow a Failover Server to Operate for Extended Periods without Its Failover Partner

One or both failover partners could potentially move into COMMUNICATIONS-INTERRUPTED state. They cannot issue duplicate addresses while in this state. However, having a server in this state over longer periods is not a good idea, because there are restrictions on what a server can do. The main server cannot reallocate expired leases and the backup server can run out of addresses from its pool.

COMMUNICATIONS-INTERRUPTED state was designed for servers to easily survive transient communication failures of a few minutes to a few days. A server might function effectively in this state for only a short time, depending on the client arrival and departure rate. After that, it would be better to move a server into PARTNER-DOWN state so it can completely take over the lease functions until the servers resynchronize.

There are two ways a server can move into PARTNER-DOWN state:

- **User action**—An administrator sets a server into PARTNER-DOWN state based on an accurate assessment of reality. The failover protocol handles this correctly. Never set both partners to PARTNER-DOWN.
- **Failover safe period expires**—When the servers run unattended for longer periods, they need an automatic way to enter PARTNER-DOWN state.

For more information, see [Using the Failover Safe Period to Move Servers into PARTNER-DOWN State, on page 67](#).



Note

It is strongly recommended that when one server in a failover pair has been or will be out of service for any extended period that the other server be placed into PARTNER-DOWN state and that the failover relationship remain configured.

The alternative, unconfiguring the failover relationship, will have much the same effect on the server that remains operational, but reintegrating that server and the returning failover partner back into a working failover relationship with no impact on the lease state data is difficult and may be impossible.

When one server in a failover pair will be down for a while, you must place the remaining, operational server into PARTNER-DOWN state. DO NOT unconfigure the failover relationship on the operational server.

Reintegrating the Returning Failover Partner

If the returning server has retained an intact lease state database, it is brought back into service and should make contact with the operational server.

If the returning server has failed catastrophically and could not be returned to service with an intact lease state database, then the situation is a bit more complicated. In this case, a new installation of CPNR is usually

required on the returning server (which may not even be the same physical machine). The returning server should have the same IP address as the failed server and the new CPNR installation must be configured identically to the failed server. Which, typically, is the same as the operational server. Then the new server is brought into service and makes contact with the existing operational server.



Note In both cases, it is vital that the existing, operational server actually be operational at the time that the returning server is brought online, since if the returning server cannot contact the operational server it will think it is the only operational server and start handing out IP addresses without regard or knowledge of what the operational server has done.

When a returning server first comes up it will contact the operational server and they will exchange the times that they last communicated.

There are two possible situations that can arise:

- When a server with an intact lease state database (where CPNR was not re-installed) returns to service, it will determine after contacting its partner that it was out of service for a while, and move into RECOVER state and its partner will send it information about what has happened since it left service. When this update is complete, both servers will move into NORMAL state.
- When server that had CPNR re-installed on it completes this exchange, it will recognize that it has never communicated with the operational server, but the operational server has communicated with it (or with its predecessor), and the newly restored server will realize that it has lost its lease state database. It will move into RECOVER state and then request a complete download of all of the lease state information from the operational server. When this download is complete (which may take minutes or possibly longer, depending on the size of the lease state database and the load on the servers), both servers will move into NORMAL state.

Restoring a Standalone DHCP Failover Server - Tutorial

This section describes how to recreate a DHCP failover relationship between a main and backup server where a backup server was put in standalone mode. This situation does not come up very often.

The proper way to handle a situation where a main server is out of service for any period beyond a few minutes is to set the backup server into PARTNER-DOWN state. For more information, see [Using PARTNER-DOWN State to Allow a Failover Server to Operate for Extended Periods without Its Failover Partner, on page 77](#).

The following procedure is offered to recover from the situation where an administrator has mistakenly believed that the proper approach is to remove the backup server from the failover relationship if the main server is out of service. To reiterate, this is NOT the correct procedure. It is challenging to recover from this mistake, but the following procedure should help.

1. The standalone server assumes the role of the main server.
2. The original main server becomes the backup server.
3. The partners then synchronize.
4. Failover relationship to be intentionally broken to reverse the server roles.
5. Partners to resynchronize in their original failover roles.

Related Topics

- [Background](#) , on page 79
- [Repair Procedure](#) , on page 79
- [Reversing the Failover Role on Backup Server](#), on page 80
- [Starting with Server A Powered Off](#), on page 80
- [Starting with Server A Replaced](#), on page 82
- [Transferring Current Lease State to Server A](#), on page 82

Background

For the remainder of this section, the main DHCP failover server is identified as Server A (with a cluster object named cluster-A), and the backup server as Server B (with a cluster object named cluster-B). Server A is administratively or otherwise shut down or its Cisco Prime Network Registrar server agent gets stopped. At this point, Server B goes into the Communications-Interrupted mode.

The system administrator may then take one of the following approaches:

- **Continue running backup Server B in Communications-Interrupted mode**—The risk of running the backup server in this mode indefinitely is that it can exhaust the pool of typically 10% of the available addresses with which the backup server is allocated to service new clients.
- **Put Server B into Partner-Down mode without breaking the failover relationship**—One major caveat of giving the backup server full control of the address space, without suspending failover, is that the full transfer of the address space ownership does not occur until after the configured Maximum Client Lead Time (MCLT). The MCLT is an additional time period set on the main server, which controls the duration for which the client lease expiration is ahead of what the backup server detects it to be. The MCLT is typically 60 minutes. Until the MCLT expires, the available address pool of the backup server is limited to its allocated reserve.
- **Put Server B into Partner-Down mode and break the failover relationship**—This approach puts the backup server in standalone mode, and is the approach that the administrator chose in this scenario. The deciding factors were that the main server was expected to be offline for an extended period, and the number of new devices coming online was higher than anticipated. Because the low percentage of available addresses that the backup server could service would soon cause an outage for new devices, the administrator put Server B in standalone mode. The disadvantage of this approach is the care and effort required to preserve the original state of the network when restoring the partners to their original relationship.

The first two approaches have distinct advantages over the third. In most cases, the backup server is expected to have enough addresses to cover newly arrived clients until the MCLT expires. Pursuing the third approach can incur unnecessary administrative burden and risk.

Repair Procedure

The repair procedure is:

1. **Temporarily assign the backup Server B the role of the main failover server**—Reversing the failover partner roles effectively allows Server A to learn the current failover state from Server B.
2. **Migrate Server A and Server B back to their original failover roles**—The goal is for Server A to reacquire its original status as the main DHCP failover server.

The assumptions are:

- The Original main Server A is nonoperational and Cisco Prime Network Registrar is stopped.
- The Original backup Server B is operational.
- Failover between the partners is administratively disabled.
- Decision was made not to permanently reverse the failover roles of the two partners.
- Domain Name Services (DNS) is not running on either of the failover partners.



Note The IP addresses used as examples are for demonstration purposes only.

Reversing the Failover Role on Backup Server

The following steps restore failover by temporarily moving Server B into the main server mode.

On **Server B** (cluster-B):

Step 1 Ensure that failover is disabled. Modify the failover configuration, so that Server B becomes the main and Server A the backup:

```
nrcmd> failover-pair examplepair set failover=false
nrcmd> failover-pair examplepair set main=cluster-B backup=cluster-A
```

Step 2 Save the changes and reload the server:

```
nrcmd> save
nrcmd> dhcp reload
```

Step 3 Re-enable failover and reload the server again:

```
nrcmd> failover-pair examplepair set failover=true
nrcmd> dhcp reload
```

Server B is now the main failover server, ready for its partner to become operational again. Any further action that you take to prevent Server A from beginning to give out addresses in the meantime depends on its current state.

If Server A is:

- **Powered off**—See [Starting with Server A Powered Off, on page 80](#).
 - **Powered on with the Cisco Prime Network Registrar DHCP not configured to start**—See [Starting with Server A Powered On and DHCP Server Stopped, on page 81](#).
 - **Replaced by another machine**—See [Starting with Server A Replaced, on page 82](#).
-

Starting with Server A Powered Off

If Server A was powered off, you must power it on again to continue. The next steps ensure that Server A comes online while preventing IP address leakage.

On **Server A** (cluster-A):

-
- Step 1** Before turning on the server, you must take steps to prevent it from communicating with clients. The best way to do this is to manually disconnect the network cable, then boot up the machine. You will require a local console to perform the next step. Other alternatives include reconfiguring the relay agents not to forward packets to the server or otherwise preventing DHCP traffic to be received on the machine (such as by installing a temporary filter for DHCP packets on a firewall).
- Note** If it is not possible to prevent client traffic from reaching the server, it may provide erroneous information to clients that do attempt to communicate with it, until the DHCP server can be stopped. Therefore, you must be ready to stop the DHCP server as soon as possible after turning the server on, as described in the next steps, to reduce the number of clients that might be provided erroneous information, potentially resulting in duplicated leases.
- Step 2** Turn on the server.
- Step 3** Stop the DHCP server as quickly as possible:
`nrcmd> dhcp stop`
- Step 4** Go to the [Starting with Server A Powered On and DHCP Server Stopped](#), on page 81.
-

Starting with Server A Powered On and DHCP Server Stopped

Starting from a point where Server A is powered on, but the Cisco Prime Network Registrar DHCP server is stopped:

On **Server A** (cluster-A):

-
- Step 1** Modify the failover configuration so that Server A becomes the backup server:
`nrcmd> failover-pair examplepair set main=cluster-B backup=cluster-A`
- Step 2** Stop Cisco Prime Network Registrar:
- RHEL/CentOS 6.x— `/etc/init.d/nwreglocal stop`
 - RHEL/CentOS 7.x— `systemctl stop nwreglocal`
 - Windows— `net stop nwreglocal`
- Step 3** Examine the DHCP logs to confirm that the DHCP server is not running.
- Step 4** Bring Server A back on the network. Reconnect the network cable, reconfigure the relay agents, or remove any firewall filter added in the previous section.
- Step 5** Remove the lease state database and event store:
- Linux
`rm -rf /var/nwreg2/local/data/dhcpeventstore`
`rm -rf /var/nwreg2/local/data/dhcp/ndb`
 - Windows

```
cd install-path\local\data
rmdir /s dhcpeventstore
rmdir /s dhcp\ndb
```

- Step 6** Start Cisco Prime Network Registrar:
- RHEL/CentOS 6.x— **/etc/init.d/nwreglocal start**
 - RHEL/CentOS 7.x— **systemctl start nwreglocal**
 - Windows— **net start nwreglocal**
- Step 7** Set the DHCP service to be enabled on reboot and start the DHCP server:
- ```
nrcmd> dhcp enable start-on-reboot
nrcmd> dhcp start
```
- Step 8** Go to the [Transferring Current Lease State to Server A, on page 82](#).
- 

## Starting with Server A Replaced

If Server A was decommissioned and replaced, you must install Cisco Prime Network Registrar and push the failover configuration from Server B to the new machine. Also, you must restore any customer configuration specific to Server A. After these steps, Cisco Prime Network Registrar will start but not give out addresses:

---

- Step 1** On **Server A** (cluster-A), install Cisco Prime Network Registrar.
- Step 2** Reconstruct the Cisco Prime Network Registrar operating environment by restoring the accompanying software, such as Cisco Broadband Access Center and its required DHCP extensions. Do not make any administrative changes to the configuration until after pushing the configuration to Server B.
- Step 3** On **Server B** (cluster-B), by using the Cisco Prime Network Registrar web UI, push an exact failover configuration to Server A. This effectively makes Server A the backup partner.
- Step 4** On **Server A**:
- a) If necessary, customize the Cisco Prime Network Registrar configuration as required for the operating environment, which might include making administrative changes.
  - b) Reload the DHCP server:
- ```
nrcmd> dhcp reload
```
- Step 5** Go to [Transferring Current Lease State to Server A, on page 82](#).
-

Transferring Current Lease State to Server A

- At this point, the failover partnership reestablishes itself, both servers will resynchronize their states.
- Server A becomes operational as the backup server.
- The operation will pause for the MCLT period (of one hour) and both partners resume their failover operations in normal communication mode.



Note Do not proceed to the [Repairing Partners to Their Original Roles, on page 83](#) until both partners synchronize and report normal communication.

Repairing Partners to Their Original Roles

Assume that both partners are fully synchronized and report normal communication. To ensure that the failover partners can assume their original roles, you should:

Step 1 On **Server A** (cluster-A), stop the DHCP server:

```
nrcmd> dhcp stop
```

Step 2 On **Server B** (cluster-B), stop the DHCP server:

```
nrcmd> dhcp stop
```

Step 3 On **Server A**:

a) Disable failover, then make Server A the main server and Server B the backup:

```
nrcmd> failover-pair examplepair set failover=false
```

```
nrcmd> failover-pair examplepair set main=cluster-A  
backup=cluster-B
```

b) Save the changes and reload DHCP:

```
nrcmd> save
```

```
nrcmd> dhcp reload
```

c) Ensure that the configuration is in place and currently running. At this point, Server A is the sole operational DHCP server with 100% of the address pool.

d) Re-enable failover:

```
nrcmd> failover-pair examplepair set failover=true
```

e) Reload DHCP and double-check the configuration changes:

```
nrcmd> dhcp reload
```

Server A is now the failover main server awaiting Server B to become operational.

Step 4 On **Server B**:

a) Make Server A the main server and Server B the backup, then enable failover:

```
nrcmd> failover-pair examplepair set main=cluster-A  
backup=cluster-B
```

```
nrcmd> failover-pair examplepair set failover=true
```

- b) Save the new configuration, but do not reload the server:

```
nrcmd> save
```

- c) Restart the DHCP server on Server B:

```
nrcmd> dhcp reload
```

At this point, the failover partnership reestablishes itself in its original roles, both servers will resynchronize their states, and Server B becomes operational as the backup server. The operation will pause for the MCLT period (of one hour) and both partners resume their failover operations in normal communication mode.

Step 5 On Server A and Server B:

- a) Validate whether both partners are in normal failover state:

```
nrcmd> dhcp getRelatedservers
```

- b) Run a report and ensure that the results match on both partners, allowing a bit of skew for the difference in running times between the partners.

Changing Failover Server Roles



Caution

Be careful when you change the role of a failover server. Remember that all address states in a DHCPv4 scope or DHCPv6 prefix are lost from a server if it is ever reloaded without that scope or prefix in its configuration.

Related Topics

[Establishing Failover Using Standalone Server as Main, on page 84](#)

[Replacing Servers Having Defective Storage, on page 85](#)

[Removing Backup Servers and Halting Failover Operation, on page 86](#)

[Adding Main Servers to Existing Backup Servers, on page 86](#)

[Configuring Failover on Multiple Interface Hosts, on page 86](#)

Establishing Failover Using Standalone Server as Main

You can update an existing installation and increase the availability of the DHCP service it offers. You can use this procedure only if the standalone server never participated in failover.

Step 1 Install Cisco Prime Network Registrar on the machine that is to be the backup server. Note the IP address of the backup server.

Step 2 Configure the cluster. Enable failover on the standalone server, configure it to be the main server and recently installed as the backup.

To configure the cluster, use:

```
cluster name create address | ipv6-address scp-port=value admin=value password=value
```

For example:

```
cluster backup create 10.65.201.23 scp-port=1234 admin=admin password=changeme
```

- Step 3** Reload the main server. It should go into PARTNER-DOWN state. It cannot locate the backup server, because it is not yet configured. There should be no change in main server operation at this point.
- Step 4** To sync the configuration use failover synchronization and do a exact sync from Main to Backup.
- Step 5** Reconfigure all operational BOOTP relays to forward broadcast packets to the main server and backup server.
- Step 6** Reload the backup server.

What to do next

After you complete these steps:

1. The backup server detects the main server and moves into RECOVER state.
2. The backup server refreshes its stable storage with the main server lease data, and when complete, moves into RECOVER-DONE state.
3. The main server moves into NORMAL state.
4. The backup server moves into NORMAL state.
5. The backup server sends a pool request to get its pool of address.
6. After allocating these addresses, the main server allocates the IP address to backup based on backup percentage.

Replacing Servers Having Defective Storage

If a failover server loses its stable storage (hard disk), you can replace the server and have it recover its state information from its partner.

- Step 1** Determine which server lost its stable storage.
- Step 2** Use **failover-pair name setPartnerDown [date]** in the CLI to tell the other server that its partner is down. If you do not specify a time, the current time is used.
- Step 3** When the server is again operational, reinstall Cisco Prime Network Registrar.
- Step 4** Sync the server configuration from its partner configuration using failover synchronization. However, do not recover any lease databases from an earlier backup or the partner system.
- Step 5** Reload the replacement server.

What to do next

After you complete these steps:

1. The recovered server moves into RECOVER state.
2. Its partner sends it all its data.

3. The server moves into RECOVER-DONE state when it reaches its maximum client lead time (and any time set for *failover-recover*).
4. Its partner moves into NORMAL state.
5. The recovered server moves into NORMAL state. It can request addresses, but can allocate few new ones, because its partner already sent it all its previously allocated addresses.

Removing Backup Servers and Halting Failover Operation

Sometimes you might need to remove the backup server and halt all failover operations.

-
- Step 1** On the backup server, remove all the scopes or prefixes that were designated as a backup to the main server.
 - Step 2** On the main server, remove the failover capability from those scopes or prefixes that were main for the backup server, or disable failover server-wide if that is how it was configured.
 - Step 3** Reload both servers.
-

Adding Main Servers to Existing Backup Servers

You can use an existing backup server for a main server.

-
- Step 1** Sync the main server scopes, policies, and other configurations on the backup server using failover synchronization.
 - Step 2** Configure the main server to enable failover and point to the backup server.
 - Step 3** Configure the backup server to enable failover for the new scopes that point to the new main server.
 - Step 4** Reload both servers. Cisco Prime Network Registrar performs the same steps as those described in the [Establishing Failover Using Standalone Server as Main, on page 84](#).
-

Configuring Failover on Multiple Interface Hosts

If you plan to use failover on a server host with multiple interfaces, you must explicitly configure the local server name or address. This requires an additional command. For example, if you have a host with two interfaces, server A and server B, and you want to make server A the a main failover server, you must define server A as the failover-main-server before you set the backup server name (external server B). If you do not do this, failover might not initialize correctly and tries to use the wrong interface.

Set the DHCP server properties *failover-main-server* and *failover-backup-server*.

With multiple interfaces on one host, you must specify a hostname that points to only one address or a record. You cannot set up your servers for round-robin support.

Troubleshooting Failover

This section describes how to avoid failover configuration mistakes, monitor failover operations, and detect and handle network problems.

Related Topics

[Monitoring Failover Operations, on page 87](#)

[Detecting and Handling Network Failures, on page 87](#)

[Things to Avoid When Troubleshooting Issues Related to Failover, on page 88](#)

Monitoring Failover Operations

You can examine the DHCP server log files on both partner servers to verify your failover configuration.

You can make a few important log and debug settings to troubleshoot failover. Set the DHCP log settings to *failover-detail* to track the number and details of failover messages logged. To ensure that previous messages do not get overwritten, add the *failover-detail* attribute to the end of the list. Use the *no-failover-conflict* attribute to inhibit logging server failover conflicts, or the *no-failover-activity* attribute to inhibit logging normal server failover activity. Then, reload the server.

You can also isolate misconfigurations more easily by clicking the **Related Servers** icon on the Manage DHCP Server or List/Add DHCP Failover Pairs page, or by using `dhcp getRelatedServers` in the CLI.

Detecting and Handling Network Failures

The table below describes some symptoms, causes, and solutions for failover problems.

Table 12: Detecting and Handling Failures

Symptom	Cause	Solution
New clients cannot get addresses	A backup server is in COMMUNICATIONS-INTERRUPTED state with too few addresses.	Increase the backup percentage on the main server.
Error messages about mismatched scopes	There are mismatched scope configurations between partners.	Reconfigure your servers.
Log messages about failure to communicate with partner	Server cannot communicate with its partner.	Check the status of the server.
Main server fails. Some clients cannot renew or rebind leases. The leases expire even when the backup server is up and possibly processing some client requests.	Some BOOTP relay agent (ip-helper) was not configured to point at both servers; see the Configuring BOOTP Relays, on page 89 .	<ul style="list-style-type: none"> Reconfigure BOOTP relays to point at both main and backup server. Run a fire drill test—Take the main server down for a day or so and see if your user community can get and renew leases.
SNMP trap: other server not responding	Server cannot communicate with its partner.	Check the status of the server.

Symptom	Cause	Solution
SNMP trap: dhcp failover configuration mismatch	Mismatched scope configurations between partners	Reconfigure your servers.
Users complain that they cannot use services or system as expected	Mismatched policies and client-classes between partners	Reconfigure partners to have identical policies; possibly use LDAP for client registration if currently registering clients directly in partners.

Things to Avoid When Troubleshooting Issues Related to Failover

When using failover, here are some things NOT to do when troubleshooting issues:

- Removing the failover configuration. It is far better to set the remaining server into PARTNER-DOWN state. There are cases where this will require a longer wait to reuse the lease, but it is far safer to keep failover configured and operate in PARTNER-DOWN.
- Never copy the DHCP lease database (.../data/dhcp/ndb) from one failover partner to the other. See the *Restoring DHCP Data from a Failover Server* section in *Cisco Prime Network Registrar 9.0 Administration Guide* for how to recover the lease data from the failover partner. If this is done, you MUST use the leaseadmin tool to remove the server-duid after copying the database (see [Moving Leases Between Servers, on page 213](#) for more details on the leaseadmin tool). Any time the lease databases are copied, the server-duid must be removed from the copy.



Note If the server-duid is not deleted, you can end up with two servers having the same server-id and hence DHCPv6 will not work as intended; this can have serious consequences for regional lease history data.

Supporting BOOTP Clients in Failover

You can configure scopes to support two types of BOOTP clients—static and dynamic.

Related Topics

[Static BOOTP, on page 88](#)

[Dynamic BOOTP, on page 89](#)

[Configuring BOOTP Relays, on page 89](#)

Static BOOTP

You can support static BOOTP clients using DHCP reservations. When you enable failover, remember to configure both the main and backup servers with identical reservations.

Dynamic BOOTP

You can enable dynamic BOOTP clients by enabling the *dynamic-bootp* attribute on a scope. When using failover, however, there are additional restrictions on address usage in such scopes, because BOOTP clients get permanent addresses and leases that never expire.

When a server whose scope does not have the *dynamic-bootp* option enabled goes to PARTNER-DOWN state, it can allocate any available (unassigned) address from that scope, whether or not it was initially available to any partner. However, when the *dynamic-bootp* option is set, each partner can only allocate its own addresses. Consequently, scopes that enable the *dynamic-bootp* option require more addresses to support failover.

When using dynamic BOOTP:

- Segregate dynamic BOOTP clients to a single scope. Disable DHCP clients from using that scope by disabling the *dhcp* attribute on the scope.
- Set the *dynamic-bootp-backup-pct* failover pair attribute to allocate a greater percentage of addresses to the backup server for this scope, as much as 50 percent higher than a regular backup percentage.

Configuring BOOTP Relays

The Cisco Prime Network Registrar failover protocol works with BOOTP relay (also called IP helper), a router capability that supports DHCP clients that are not locally connected to a server.

If you use BOOTP relay, ensure that the implementations point to both the main and backup servers. If they do not and the main server fails, clients are not serviced, because the backup server cannot see the required packets. If you cannot configure BOOTP relay to forward broadcast packets to two different servers, configure the router to forward the packets to a subnet-local broadcast address for a LAN segment, which could contain both the main and backup servers. Then, ensure that both the main and backup servers are on the same LAN segment.

BOOTP Backup Percentage

For scopes for which you enable dynamic BOOTP, use the *dynamic-bootp-backup-pct* attribute rather than the *backup-pct* attribute for the failover pair. The *dynamic-bootp-backup-pct* is the percentage of available addresses that the main server should send to the backup server for use with BOOTP clients.

The *dynamic-bootp-backup-pct* is distinct from the *backup-pct* attribute, because if you enable BOOTP on a scope, a server, even in PARTNER-DOWN state, never grants leases on addresses that are available to the other server. Cisco Prime Network Registrar does not grant leases because the partner might give them out using dynamic BOOTP, and you can never safely assume that they are available again.



Note You must define the dynamic BOOTP backup percentage on the main server. If you define it on the backup server, Cisco Prime Network Registrar ignores it (to enable duplicating configuration through scripts). If you do not define it, Cisco Prime Network Registrar uses the default *backup-pct* for the failover pair or scope.

To properly support dynamic BOOTP while using the failover protocol, do this on every LAN segment in which you want BOOTP support:

- Create one scope for dynamic BOOTP
- Enable BOOTP and dynamic BOOTP
- Disable DHCP for that scope



CHAPTER 4

Managing Address Space

Address blocks provide an organizational structure for addresses used across the network. Address blocks can consist of static addresses or dynamic addresses allocated to DHCP servers for lease assignment. An address block can have any number of child address blocks and can culminate in one or more child subnets. The address block administrator is responsible for these objects. This administrator can create parent and child address blocks or subnets, which are always the leaf nodes of the address space. Static subnets can be further subdivided into one or more IP address ranges. However, dynamically added subnets create their own subnets that the administrator cannot modify or delete.



Note For IPv6 address management, see the [Viewing IPv6 Address Space, on page 103](#)

- [Address Block Administrator Role, on page 91](#)
- [Address Blocks and Subnets, on page 92](#)
- [Pulling and Pushing, on page 99](#)
- [Viewing Address Space, on page 100](#)
- [Generating Subnet Utilization History Reports, on page 105](#)

Address Block Administrator Role

The address block administrator role manages address space at a higher level than that of specific subnet or static address allocations. This is actually a middle manager role, because there is likely to be a higher authority handing out address blocks to the system.

Related Topics

[Required Permissions, on page 91](#)

[Role Functions, on page 92](#)

Required Permissions

To exercise the functions available to the address administrator, you must have at the:

- **Regional cluster**—The regional-addr-admin role assigned. This role should probably be unencumbered by further subnet-utilization, lease-history, and dhcp-management subrole restrictions.

- **Local cluster**—The `addrblock-admin` role assigned.

Role Functions

These functions are available to the address block administrator at the:

- **Regional cluster:**
 - Address aggregation. For example, if the 10.0.0.0/16 address block exists at the regional cluster and a local cluster administrator creates the 10.1.1.0/24 address block, the local address block (through replication) is rolled up under its parent at the regional cluster. This allows a unified view of the address space at the regional cluster without affecting the local cluster configuration.
 - Address delegation. Administrators can delegate address space to the local cluster, thereby giving up authority of the delegated object.
 - Subnet utilization reports. The regional cluster supports subnet utilization reporting across regions, protocol servers, and sets of network hardware. The central configuration administrator can poll the local clusters for subnet utilization by virtual private network (VPN), if defined, time range, and criteria that contain the following choices: owner, region, address type, address block, subnet, or all. For details on querying subnet utilization, see [Generating Subnet Utilization History Reports, on page 105](#).
 - Lease history reports. This provides a single vantage point on the lease history of multiple DHCP servers. The administrator can query the history data at the local cluster to constrain the scope of the history report. Lease histories can be queried by VPN (if defined), time range and criteria that contain the following choices: IP address, MAC address, IP address range, or all. This is an important feature to meet government and other agency mandates concerning address traceability. For details on querying lease history, see [Querying Leases, on page 216](#).
 - Polling configurations. The administrator can control the intervals and periods of local cluster polling for replication, IP histories, and subnet utilization. You can also set the lease history and subnet utilization trimming ages and compacting intervals at the CCM server level. (See the *"Managing Central Configuration" chapter in Cisco Prime Network Registrar 9.0 Administrator Guide*)
 - Check the DHCP and address data consistency.
- **Local cluster:**
 - Manage address blocks, subnets, and address types.
 - Check the DHCP and address data consistency.

Address Blocks and Subnets

An address block is an aggregate of IP addresses based on a power-of-two address space that can be delegated to an authority. For example, the 192.168.0.0/16 address block (part of the RFC 1918 private address space) includes 216 (or 65536) addresses. Address blocks can be further divided into child address blocks and subnets. For example, you might want to delegate the 192.168.0.0/16 address block further into four child address blocks—192.168.0.0/18, 192.168.64.0/18, 192.168.128.0/18, and 192.168.192.0/18.



Note The DHCP server also uses address blocks to manage subnet allocation for on-demand address pools (see [Configuring Subnet Allocation, on page 48](#)). Address blocks used for dynamic address pools must be created using the **dhcp-address-block** command in the CLI. The unified address view in the web UI also displays these dynamic address blocks, but does not provide an edit link to them, because they have been delegated in their entirety to the DHCP server. They should not be further subdivided for subnet allocation. The DHCP server automatically handles these address blocks as it receives subnet requests. These address pools are indicated by a **D** (for “Delegated”).

A subnet is the leaf node of the address space and cannot be further subdivided. If you create the 192.168.50.0/24 subnet, you can subsequently create an address block by that same name, and the subnet will become a child of the address block. However, you cannot further subdivide or delegate the 192.168.50.0/24 subnet.

Subnets can have one or more defined address ranges. Address blocks cannot have address ranges. When you create an address range for a subnet by using the web UI, it becomes a static range, meaning that it cannot be allocated dynamically using DHCP. However, the web UI shows any dynamic ranges defined by DHCP scopes for the subnet. Displaying the ranges as such indicates where overlaps may occur between assigning static addresses for the address space and dynamic addresses for scopes.

The address space view shows the hierarchy of address block and subnets and their parent-child relationships. The hierarchy does not go down to the level of address ranges for each subnet. These are displayed when you access the subnet.

Related Topics

[Viewing Address Blocks, Subnets, and Address Types, on page 102](#)

[Knowing When to Add Address Blocks, on page 94](#)

[Adding Address Blocks, on page 95](#)

[Delegating Address Blocks, on page 97](#)

[Pushing Subnets to Local DHCP Servers and Routers, on page 100](#)

[Creating Reverse Zones from Subnets, on page 97](#)

[Reclaiming Subnets, on page 97](#)

[Adding Children to Address Blocks, on page 98](#)

[Adding Address Ranges to Subnets, on page 99](#)

[Viewing Address Utilization for Address Blocks, Subnets, and Scopes, on page 101](#)

Subnet Allocation and DHCP Address Blocks

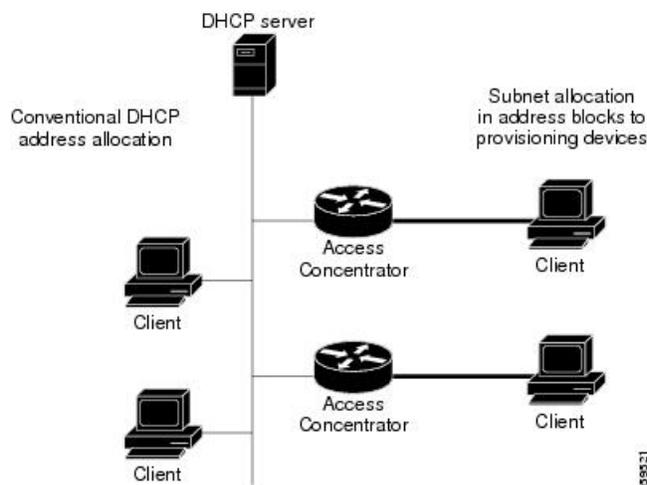
Cisco Prime Network Registrar supports creating on-demand address pools as a network infrastructure for address provisioning and VPNs. Traditionally, the DHCP server is limited to interact with individual host devices. Through subnet allocation, the server can interact with VPN routers and other provisioning devices to provision entire IP subnets. This Cisco Prime Network Registrar feature enhances the on-demand address pool capability currently supported by the Cisco IOS Relay Agent.

Cisco Prime Network Registrar supports explicitly provisioned subnets. You must explicitly configure the DHCP server address space and subnet allocation policies before the server can allocate pools or leases. You can thereby configure a server as a pool manager to manage subnets and delegate them to client devices.

You manage DHCP subnet allocation using DHCP server address block objects in Cisco Prime Network Registrar. A DHCP address block is a range of contiguous IP addresses delegated to the DHCP server for assignment. The server expects to subdivide these addresses into pools so that it or other servers or devices can allocate them. DHCP address blocks are parents to subnets. These DHCP address blocks are distinct from the address blocks you can create using the Cisco Prime Network Registrar web UI, which are static. DHCP address blocks cannot include static address ranges or lease reservations.

The image below shows a sample environment where a DHCP server allocates entire subnets to access concentrators or other provisioning devices, in addition to servicing individual clients. The traditional client/server relationship is shown on the left of the diagram, while the subnet allocation to access concentrators is shown on the right of the diagram. Dialup customers, for example, connect to the service provider network at two ISP gateways (routers), which connect to the management network segment where the DHCP server resides. The gateways provision addresses to their connected clients based on the subnet requested from the DHCP server.

Figure 10: Sample DHCP Subnet Allocation Configuration



Knowing When to Add Address Blocks

This use case describes the set of user actions associated with adding a new address block to the network in a shared management network. These preconditions are assumed:

1. From summary IP address utilization reports (see the *"Enabling Subnet Utilization Collection" section in Cisco Prime Network Registrar 9.0 Administrator Guide*), an address block administrator notes that the top level address block of the company is nearing the 90% utilization mark.
2. The address block administrator submits a request for more address space from ARIN (or some other numbering authority) and the request is granted.

Once the address space is made available, the regional address administrator:

1. Adds the new blocks to the central address block map, and based on a review of the utilization reports, creates and delegates address blocks to be used by the local clusters. The action of delegating the address blocks causes them to be pushed to the local clusters.

2. Allocates the new address space to network elements as needed using failover synchronization to simplify the configuration tasks:
 - Allocates subnets to a failover pair (gets a scope template for the subnet, either from the subnet or the failover pair).
 - Finds a free subnet (finds the address block of the right type).
 - Allocates the free subnet to an address destination (DHCP server or other destination).

Adding Address Blocks

Once you configure your network, you can add DHCPv4 address blocks.

Local Advanced and Regional Web UI

To view the CCM address blocks, from the **Design** menu, choose **Address Blocks** under the **DHCPv4** submenu to open the List/Add DHCP Address Blocks page.

To add an address block, click the **Add Address Block** icon in the Address Blocks pane on the left. Enter its network address in the Address field, then choose the address mask from the drop-down list. For example, enter 192.168.50.0 in the Address field, then choose 24 in the drop-down list to create the 192.168.50.0/24 address block, which is all the addresses in the range 192.168.50.0 through 192.168.50.255.

For a review of the number of available addresses for each subnet mask, see the table below. These available hosts exclude the two network and broadcast addresses in each range.

Table 13: Subnet Masking

Network Mask	Octet Designation	Available Hosts in Each Address Range
/8	255.0.0.0	16777214
/9	255.128.0.0	8338606
/10	255.192.0.0	4194302
/11	255.224.0.0	2097150
/12	255.240.0.0	1048574
/13	255.248.0.0	524286
/14	255.252.0.0	262142
/15	255.254.0.0	131070
/16	255.255.0.0	65534
/17	255.255.128.0	32766
/18	255.255.192.0	16382
/19	255.255.224.0	8190
/20	255.255.240.0	4084
/21	255.255.248.0	2046

Network Mask	Octet Designation	Available Hosts in Each Address Range
/22	255.255.252.0	1022
/23	255.255.254.0	510
/24	255.255.255.0	254
/25	255.255.255.128	126
/26	255.255.255.192	62
/27	255.255.255.224	30
/28	255.255.255.240	14
/29	255.255.255.248	6
/30	255.255.255.252	2

Configuring Private Networks in VPN for a Tenant

To configure the private networks in VPN for a tenant, do the following:

Regional Advanced Web UI

-
- Step 1** Select the required tenant from the **Tenant** sub menu under the **Settings** drop-down list at the top of the web UI.
- Step 2** Select the required VPN from the **VPN** sub menu under the **Settings** drop-down list at the top of the web UI.
- Step 3** Navigate to the **Design** menu and choose **VPNs** under the **DHCP Settings** submenu to open the List/Add VPNs page. Set the *Tenant-Private-Network* attribute to true. This attribute indicates that this VPN represents the tenant non-routable (RFC1918) addresses on a local cluster.
- Note** It applies only to regional CCM clusters and will be ignored if set on a local cluster.
- Step 4** Create the private address blocks (for example, 10.0.0.0/24).
-

CLI Commands

Use `session set attribute=value` to set tenants and VPNs. Use `vpn name set tenant-private-network=value` to indicate that this VPN represents the tenant non-routable addresses on a local cluster. Then, create the private address block using `address-block [vpn-name/]address/mask create`. For example:

```
nrcmd-R> session set tenant=t1
nrcmd-R [Tenant:t1]> session set vpn=vpn1
nrcmd-R [Tenant:t1 VPN:vpn1]> vpn vpn1 set tenant-private-network=true
nrcmd-R [Tenant:t1 VPN:vpn1]> address-block 10.0.0.0/24 create
```

Delegating Address Blocks

Address block delegation is the coordinated actions of marking the delegated address block at the regional cluster as being delegated to a local cluster and creating the delegated address block in the local cluster. To delegate an address block to a local cluster, the address block cannot have child address blocks or subnets. The delegated address block created at the local server must have the same address size as the one at the regional cluster.

You can delegate only one address block to one local cluster at a time; you cannot delegate it to multiple local clusters. You can also delegate an address block to an owner.

To delegate an address block, you must:

1. Have the central configuration administrator create a local cluster to which to delegate the address block (see the “*Configuring Server Clusters*” section in *Cisco Prime Network Registrar 9.0 Administrator Guide*).
2. Have the central configuration administrator synchronize the regional cluster with the local cluster (see the “*Synchronizing with Local Clusters*” section in *Cisco Prime Network Registrar 9.0 Administrator Guide*). The local cluster will have address source references to the regional cluster through the synchronization process.
3. Delegate the address block to the cluster or an owner.

Creating Reverse Zones from Subnets

You can create reverse zones from subnets directly on the List/Add Subnets page instead of having to do so manually (see the “*Adding Reverse Zones from Subnets*” section in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide*). Click the **Create** icon in the Reverse Zone column of the List/Add Subnets page to open the Create Reverse Zone(s) for Subnet page. On that page, choose a configured zone template from the drop-down list, then click **Report** to return to the List/Add Subnets page.

Related Topics

[Reclaiming Subnets, on page 97](#)

[Adding Address Ranges to Subnets, on page 99](#)

[Viewing Address Utilization for Address Blocks, Subnets, and Scopes, on page 101](#)

[Pushing Subnets to Local DHCP Servers and Routers, on page 100](#)

Reclaiming Subnets

Once you delegate a subnet to the DHCP server or router, you can reclaim it if necessary.

Local Advanced and Regional Web UI

-
- Step 1** From the **Design** menu, choose **Subnets** under the **DHCPv4** submenu to open the List/Add Subnets page.
 - Step 2** Select the subnet from the Subnets pane on the left, to open the corresponding Edit Subnet page.
 - Step 3** Click **Reclaim** at the top of the page. This opens the Reclaim Subnet page.
 - Step 4** If you want to force deleting the subnet, check the Force Delete check box.
 - Step 5** Click **Reclaim Subnet**.

Note When you push or reclaim subnets for a managed or virtual router, this sets the primary and secondary relationships that are set for the router for all the related subnets and scopes as well. For details on routers, see the *"Pushing and Reclaiming Subnets for Routers"* section in *Cisco Prime Network Registrar 9.0 Administrator Guide*.

Adding Children to Address Blocks

You might want to subdivide undelegated address blocks into child address blocks or subnets.

Local Advanced and Regional Web UI

Step 1 From the **Design** menu, choose **Address Blocks** under the **DHCPv4** submenu to open the List/Add Address Blocks page.

Step 2 Click the name of an address block that is not marked as delegated (**D**). This opens the Edit Address Block page.

Step 3 To add a child address block, add an address that is part of the address block network address in the Address/Mask field of the Child Address Blocks section of the Edit Address Block page. Choose a higher mask value than the parent address block, then click **Add**.

An error message appears if you try to set the same network address for a child address block as for a child subnet.

Omitting a value when you click **Add** automatically adds the subdivisions of the parent address space with the appropriate mask value. For example, if the parent space is 192.168.50.0/24, you omit any child subnet value, and click **Add**, the web UI adds the children in this order:

192.168.50.0/26

192.168.50.64/26

192.168.50.128/26

192.168.50.192/26

Step 4 To add a child subnet, add an address in the Address/Mask field of the Child Subnets section of the page that is part of the address block network address, but choose a higher mask value than the parent address block. Then click **Add**.

An error message appears if you try to set the same network address for a child address block as for a child subnet.

If you omit a value when you click **Add**, this automatically adds the subdivisions of the parent address space with the appropriate mask value. For example, if the parent space is 192.168.50.0/24, you omit any child subnet value, and click **Add**, the web UI adds the children in this order:

192.168.50.0/26

192.168.50.64/26

192.168.50.128/26

192.168.50.192/26

Adding Address Ranges to Subnets

You can edit the subnet data and add any number of address ranges to a subnet. These ranges must be in the designated network of the subnet.

Local Advanced and Regional Web UI

- Step 1** From the **Design** menu, choose **Subnets** under the **DHCPv4** submenu to open the List/Add Subnets page.
 - Step 2** Click the name of the subnet to which you want to add address ranges, in the Subnets pane on the left. This opens the Edit Subnet page.
 - Step 3** Enter the starting address of the range in the Start field in the IP Ranges area of the page, then add the ending address in the End field. If you add just the host numbers in these fields, the relative address in the range determined by the address mask is used.
 - Step 4** Click **Add IP Range**.
-

Pulling and Pushing

Pulling Replica Address Space from Local Clusters

You may choose to pull address space from the replica data of the local clusters instead of explicitly creating it.



- Note** Pulling replica address space from a local cluster where IPv4 subnets were removed does not clear the server name on the subnet. Although the subnet is no longer used, it is still considered allocated to the server. Hence, the delete operation does not appear for the subnet, so that you cannot delete the subnet from the regional cluster. To push or reallocate the subnet to a different cluster, or remove it from the regional cluster, you must first reclaim the subnet (see [Reclaiming Subnets, on page 97](#)). This clears the reference to the local server.
-

Regional Advanced Web UI

- Step 1** In the DHCP Address Tree (or DHCPv6 Address Tree) page, click the **Pull Data** icon in the Address Tree pane.
 - Step 2** In the Select Pull Replica Address Space (or Select Pull Replica IPv6 Address Space) page:
 - To omit reservations while pulling replica, check the **Omit Reservations** check box.
 - Select the Data Synchronization Mode (**Update**, **Complete** or **Exact**)
 - Step 3** Click **Report** at the bottom of the page.
 - Step 4** Confirm the summary and click **Run**.
 - Step 5** Click **OK**.
-

Pushing Subnets to Local DHCP Servers and Routers

You can push subnets to local DHCP servers and routers.

Local Advanced and Regional Web UI

-
- Step 1** Have the central configuration administrator create a local cluster and resynchronize it with the local cluster.
- Step 2** Create a subnet at the regional cluster:
- From the **Design** menu, choose **Subnets** under the **DHCPv4** submenu. This opens the List/Add Subnets page.
 - Click the **Add Subnet** icon in the Subnets pane on the left.
 - Enter at least the network address and choose the mask of the subnet, then click **Add Subnet**.
- Step 3** Have the central configuration administrator create a scope template so that it can create a scope to contain a subnet:
- Log into the regional cluster as the central configuration administrator.
 - From the **Design** menu, choose **Scope Templates** under the **DHCPv4** submenu to open the List/Add DHCP Scope Templates page.
 - Click the **Add Scope Templates** icon in the left pane, to open the Add DHCP Scope Template page.
 - Enter the name for the scope template and click **Add Scope Template**.
 - In the Edit DHCP Scope Template *scopename* page, among other entries on this page, enter the **create-range** expression in the Range Expression field to create a scope with that subnet. (If you choose a policy for the scope template, be sure that the policy exists at the local cluster, or you must push the policy to the local cluster. See the *"Pushing Policies to Local Clusters"* section in *Cisco Prime Network Registrar 9.0 Administrator Guide*.)
- Step 4** As regional address administrator, add the subnet to the local cluster DHCP server:
- Log into the regional cluster as the regional address administrator.
 - From the **Design** menu, choose **Subnets** under the **DHCPv4** submenu to open the List/Add Subnets page.
 - Select the subnet from the Subnets pane on the left, to open the corresponding Edit Subnet page.
 - Click **Push** at the top of the page. This opens the Push Subnet page.
 - Choose the scope template from the drop-down list.
 - Choose the router and the router interface from the drop-down lists.
 - Choose the cluster from the drop-down list.
 - Click **Push Subnet**.
-

Viewing Address Space

The address space is a hierarchical tree of address blocks and subnets in IPv4 and prefixes in IPv6, sorted in IP address order. You can choose the level of depth at which to display the tree. You can also expand and contract nodes, which recursively expands or contracts all child nodes. If you pick a new level, this overrides the previous expansion or contraction.

Local Advanced and Regional Web UI

To view the address space as a hierarchical tree:

- From the **Design** menu, choose **Address Tree** under the **DHCPv4** submenu to open the DHCP Address Tree page. Note that you can choose a VPN (if configured).

- From the **Design** menu, choose **Address Tree** under the **DHCPv6** submenu to open the DHCP v6 Address Tree page. Note that you can choose a VPN (if configured).

Viewing Address Utilization for Address Blocks, Subnets, and Scopes

You can view the current address utilization for address blocks, subnets, and scopes.



Tip For address utilization for IPv6 prefixes, see [Viewing Address Utilization for Prefixes, on page 103](#).

Local Advanced and Regional Web UI

The function is available on the DHCP Address Tree page, List/Add DHCP Address Blocks page, and List/Add Subnets page. When you click the **Current Usage** tab, the utilization detail appears.



Note To ensure the proper subnet-to-server mapping on this page, you must update the regional address space view so that it is consistent with the relevant local cluster. Do this by pulling the replica address space, or reclaiming the subnet to push to the DHCP server (see [Reclaiming Subnets, on page 97](#)). Also ensure that the particular DHCP server is running.

The other columns in the Current Usage tab identify:

- **Type**—Whether the address space is an address block, subnet, or scope.
- **Active Dynamic**—Addresses that are part of a dynamic range managed by DHCP and that are currently leased, but not reserved.
- **Free Dynamic**—Addresses that are not currently leased.
- **Active Reserved**—Addresses that are part of a dynamic range and are reserved.
- **View Utilization History**—Appears at the regional cluster only. Clicking the **Report** icon opens the Query Subnet Utilization History page, where you can refine the subnet utilization history query.

In the Current Usage tab, the Utilization Detail column items are expandable so that you can view the scope data for an address block or subnet. If you click the address block, subnet, or scope name in this column, this opens the Utilization Detail pane for selected item.

The Utilization Detail pane is read-only, and shows detailed address utilization attributes for the address block, subnet, or scope. The address utilization attributes are described in the table below.

Table 14: Address Utilization Attributes

Utilization Attribute	Description
Tenant	The tenant organization or group that is associated with the administrator
Total Addresses	
<i>total-dynamic</i>	Total number of leases, excluding reserved ones.
<i>total-reserved</i>	Total number of reserved leases.
Free Dynamic	

Utilization Attribute	Description
<i>avail</i>	Number of dynamic leases that are currently available for issue to clients.
<i>other-avail</i>	Number of dynamic leases that the DHCP failover partner currently has available for issue to clients.
Active Dynamic	
<i>offered</i>	Number of dynamic leases that are currently offered to clients, but not yet acknowledged as being leased.
<i>leased</i>	Number of dynamic leases that are currently acknowledged as leased to clients.
<i>expired</i>	Number of dynamic leases that are past the lease expiration period, but will not be available for other clients (except after the policy grace-period expires).
<i>pend-avail</i>	Number of dynamic leases that are waiting acknowledgement from the failover partner that it did not reissue the lease.
Reserved	
<i>reserved-active</i>	Number of reserved leases that clients are actively using.
<i>reserved-inactive</i>	Number of reserved leases that clients are not actively using.
Unavailable	
<i>unavail</i>	Number of unreserved dynamic leases that a client declines or the server marks with an address conflict (usually indicating configurations that need correcting).
<i>reserved-unavail</i>	Number of reserved leases that a client declines or the server marks with an address conflict (usually indicating configurations that need correcting).
Deactivated	
<i>deactivated</i>	Number of dynamic and reserved leases that clients are actively leasing (that are not offered, expired, or released), but that an administrator deactivated.
<i>leased-deactivated</i>	Number of dynamic leases that clients are actively leasing (that are not offered, expired, or released), but that an administrator deactivated.
<i>reserved-leased- deactivated</i>	Number of reserved leases that clients are actively leasing (that are not offered, expired, or released), but that an administrator deactivated.

Viewing Address Blocks, Subnets, and Address Types

You can view the address blocks and subnets created for a network.

Local Advanced and Regional Web UI

From the **Design** menu, choose **Address Tree** under the **DHCPv4** submenu to open the DHCP Address Tree page.

To choose a level of depth for the address space, click one of the addresses in the Address Tree pane on the left. The details about the address appear in the page. The Address Type column identifies the type of object displayed, an address block or a subnet. The Owner column identifies the owner of the address space, and the Region column identifies the assigned region for the address space.

Address spaces that were assigned dynamically are indicated by a **D** (for “Delegated”) in the Address Type column. You cannot delete this delegated address space.

To refresh the view, click the **Refresh** icon.

You can add, modify, and delete address types. From the **Design** menu, choose **Address Types** under the **DHCP Settings** submenu to open the List/Add Address Types page. Click the **Add Address Type** icon in the Address Types pane on the left, to open the Add Address Type page, and modify settings on the Edit Address Type page. You can also pull replica address types and push address types to the local clusters on the List/Add Address Types page.

Viewing IPv6 Address Space

From the **Design** menu, choose **Address Tree** under the **DHCPv6** submenu, in the local advanced or regional web UI, to view the DHCP v6 Address Tree page. This page is like the DHCP Address Tree page for IPv4 (see [Viewing Address Space, on page 100](#)). On the View Unified v6 Address Space page you can:

- Set a VPN for the address space from the username drop-down list on the top right of the window.
- Add a prefix by clicking the **Add Address Tree** icon in the Address Tree pane, enter the prefix name, address and choosing a DHCP type and possible template. Click **Add IPv6 Prefix** (see [Creating and Editing Prefixes, on page 131](#)).
- Edit a prefix by selecting its name in the Address Tree pane. This opens the Edit IPv6 Prefix page (see [Creating and Editing Prefixes, on page 131](#)).
- View the current usage of the prefix space (see [Viewing Address Utilization for Prefixes, on page 103](#)).

Viewing Address Utilization for Prefixes

You can view the current address utilization for prefixes.

Local Advanced and Regional Web UI

The function is available on the DHCP v6 Address Tree page [Viewing Address Space, on page 100](#).



Tip You can use the DHCP v6 Address Tree page to push and reclaim prefixes. Click the **Push** or **Reclaim** icon for the desired prefix. (See [Creating and Editing Prefixes, on page 131](#) for details.)

When you click the Current Usage tab, the Utilization Detail appears.



Note To ensure the proper subnet-to-server mapping on this page, you must update the regional address space view so that it is consistent with the relevant local cluster. Do this by pulling the replica address space, or reclaiming the subnet to push to the DHCP server. Also ensure that the particular DHCP server is running.

The other columns under the Current Usage tab identify:

- **Range**—Address range of the prefix.

- **Type**—Whether the address space is a prefix or link.
- **Active Dynamic**—Addresses that are part of a dynamic range managed by DHCP and that are currently leased, but not reserved.
- **Allocation Group**—Allocation group to which the prefix belongs.

The items under Utilization Detail are expandable so that you can view the prefix or parent prefix data.

The Utilization Detail page is a read-only page that shows detailed address utilization attributes for the prefix or the parent prefix (identified as Totals). The address utilization attributes are described in the table below.

Table 15: Address Utilization Attributes

Utilization Attribute	Description
Tenant	
<i>aggregation-level</i>	Granularity of the utilization data. Prefix-level indicates the data is for the specific prefix; totals indicates the data is for the parent prefix, which is the sum of its prefix-level counters.
<i>dhcp-type</i>	DHCP address assignment type, which can be dhcp (stateful), stateless (option configuration), prefix-delegation, or infrastructure (maps a client address to a link without an address pool).
Total Addresses	
<i>active-dynamic</i>	Total number of dynamic leases in active use (leased, offered, released, expired, or revoked). The Active Dynamic category shows the states of these leases.
<i>total-reserved</i>	Total number of reserved leases.
Active Dynamic	
<i>offered</i>	Number of dynamic (unreserved) leases that are currently offered to clients, but not yet acknowledged as being leased.
<i>leased</i>	Number of dynamic leases that are currently acknowledged as leased to clients.
<i>expired</i>	Number of dynamic leases that are past the lease expiration period, but will not be available for other clients (except after the policy grace-period expires).
<i>revoked</i>	Number of dynamic leases that the client can no longer use, but that some other client could be using.
Reserved	
<i>reserved-active</i>	Number of reserved leases that clients are actively using.
<i>reserved-inactive</i>	Number of reserved leases that clients are not actively using.
Unavailable	

Utilization Attribute	Description
<i>unavail</i>	Number of unreserved dynamic leases that a client declines or the server marks with an address conflict (usually indicating configurations that need correcting).
<i>reserved-unavail</i>	Number of reserved leases that a client declines or the server marks with an address conflict (usually indicating configurations that need correcting).
Deactivated	
<i>deactivated</i>	Number of dynamic and reserved leases that clients are actively leasing (that are not offered, expired, or released), but that an administrator deactivated.
<i>leased-deactivated</i>	Number of dynamic leases that an administrator deactivated.
<i>reserved-leased- deactivated</i>	Number of reserved leases that an administrator deactivated.
Prefix Delegation Lease Counts	
<i>max-pd-balancing-length</i>	Prefix length used for counting the prefix delegation leases.
<i>prefixes-in-use</i>	Number of prefixes, of the max-pd-balancing-length prefix length, that are in use.
<i>prefixes-available</i>	Number of prefixes, of the max-pd-balancing-length prefix length, available to any client on this server.
<i>prefixes-other-available</i>	Number of prefixes, of the max-pd-balancing-length prefix length, available to any client on the failover partner.
<i>prefixes-in-transition</i>	Number of prefixes, of the max-pd-balancing-length prefix length, in transition between the failover partners.
Failover Related	
<i>available</i>	Number of prefix delegation leases available to any client on this server. This is the count of the number of lease objects and not the number of prefixes of a specific prefix-length.
<i>other-available</i>	Number of prefix delegation leases that this server believes the partner has available to any client.
<i>pending-available</i>	Number of leases that are in the pending-available state.
<i>pending-delete</i>	Number of leases that are in the pending-delete state.

Generating Subnet Utilization History Reports

You can extract subnet utilization history data so that you can determine how many addresses in the subnet were allocated and what the free address space is. You can use additional administrative functions to trim and compact the subnet utilization database of records, to manage the size of the database.

Related Topics

[Enabling Subnet Utilization History Collection at the Local Cluster, on page 106](#)

[Querying Subnet Utilization History Data, on page 106](#)

[Trimming and Compacting Subnet Utilization History Data, on page 107](#)

[Viewing Subnet Utilization History Data, on page 108](#)

Enabling Subnet Utilization History Collection at the Local Cluster

You must explicitly enable subnet utilization collection for the local cluster DHCP server.

Local Basic or Advanced Web UI

-
- Step 1** From the **Deploy** menu, choose **DHCP Server** under the **DHCP** submenu.
- Step 2** On the Manage DHCP Server page, click the **Local DHCP Server** link.
- Step 3** On the Edit DHCP Server page, look for the Subnet Utilization Settings attributes, which determine how frequently snapshots of the data occur and over which period of time the data should be maintained:
- **collect-addr-util-duration**—Maximum period, in hours, the DHCP server maintains address utilization data. The preset value is 0. To disable DHCP server from collecting any address utilization data, unset this parameter or set it to 0.
 - **collect-addr-util-interval**—Frequency, in minutes or hours, that the DHCP server should maintain address utilization data snapshots, assuming that the *collect-addr-util-duration* attribute is not unset or set to 0. The preset value is 15 minutes.

Note that both of these parameters can impact DHCP server memory. Each snapshot of data collected for every interval is 68 bytes. For example, if there are 10 scopes, the collection duration is set to 24 hours, and the collection interval is set to one hour, memory used by the DHCP server to maintain address utilization data is 24 times 68 bytes for each scope, or 16 K.

- Step 4** Click **Save**.
- Step 5** Reload the DHCP server.
-

Querying Subnet Utilization History Data

You collect subnet utilization by first having subnets and setting up the scopes, address ranges, and collection criteria at the local cluster. You then set up the local cluster containing the DHCP server as part of the regional cluster, and enable polling the subnet utilization data from the regional cluster.

Regional Web UI

-
- Step 1** From the **Operate** menu, choose **Manage Clusters** under the **Servers** submenu to open the List/Add Remote Clusters page.
- Step 2** Click the name of the local cluster in the Manage Clusters pane on the left, to open the Edit Remote Cluster page.

- Step 3** Look for the Subnet Utilization Settings attributes:
- **poll-subnet-util-interval**—Polling interval; be sure that this is set to a reasonable time interval greater than 0.
 - **poll-subnet-util-retry**—Retry count in case of a polling failure; preset to one retry.
 - **poll-subnet-util-offset**—Fixed time when polling occurs. For example, setting the offset to 13h (1 P.M.) with the polling interval set to 2h means that polling occurs every two hours, but it must occur at 1 P.M. each day.
- Step 4** You must also set the selection criteria for querying the subnet utilization data—In the Advanced mode, from the **Operate** menu, choose **DHCPv4 Subnet Utilization History** under the **Reports** submenu. This opens the Query Subnet Utilization History page.
- Step 5** You can query subnet utilization history based on the following criteria:
1. **Time range**—Choose from one of the following time ranges for the lease history data:
 - Most Recent
 - last 10 days
 - last 30 days
 - last 60 days
 - last 90 days
 - from/to (limited to 90 days)

If you choose this value, also choose the Start Date and End Date month, day, and year from the drop-down lists. The result depends on the value of the *poll-subnet-util-interval* attribute.
 2. **Criteria**—Choose the criteria on which you want to base the query:
 - **By Owner**—Choose the owner from the adjacent drop-down list.
 - **By Region**—Choose the region from the adjacent drop-down list.
 - **By Address Type**—Choose the address type from the adjacent drop-down list.
 - **By Address Block**—Choose the address block from the adjacent drop-down list.
 - **By Subnet**—Choose the subnet from the adjacent drop-down list.
 - **All**—Choose by all owners, regions, address types, address blocks, and subnets.
- Step 6** Click **Query Subnet Utilization** to open the List Subnet Utilization Records page (see [Viewing Subnet Utilization History Data, on page 108](#)).

Trimming and Compacting Subnet Utilization History Data

If you enable subnet utilization, its database is trimmed automatically based on the expiration time of each record. You can also compact the data so that you can view subsets of the records older than a certain age. The CCM server performs background trimming at the regional cluster, which trims off the subnet utilization data older than a certain age at regular intervals. The trimming interval is preset to 24 hours, and the age (how far back to go in time before trimming) to 24 weeks.

You must be a central configuration administrator assigned the database subrole to adjust the values of and perform subnet utilization database trimming and compacting.

Regional Web UI

- Step 1** From the **Operate** menu, choose **Manage Servers** under the **Servers** submenu to open the Manage Servers page.
- Step 2** Click the **Local CCM Server** link in the Manage Servers pane on the left, to open the Edit Local CCM Server page.
- Step 3** Under the Subnet Utilization Settings, set the following attributes:
- poll-subnet-util-interval***—How often to collect subnet utilization from all the DHCP servers. If it is set to 0, the polling is disabled.
 - poll-subnet-util-retry***—The number of retries for a given polling interval, if polling fails.
 - poll-subnet-util-offset***—Provides a fixed time of day for subnet utilization polling. This time is interpreted as a time of day offset, with 0 being 12 midnight, provided the polling interval is less than 24 hours, and the offset value is less than the polling interval. If the offset value is greater than the polling interval, or the interval is greater than 24 hours, the offset will be ignored.

The scheduler for polling will ensure that the first polling event occurs at the offset time. For example, if you set the interval to 4 hours and the offset to 2 A.M., the polling would occur at 2 A.M., 6 A.M., 10 A.M., 2 P.M., 6 A.M., and 10 P.M.
 - trim-subnet-util-interval***—How often to trim the old subnet utilization data automatically, the default being not to trim the data. You must set this to a value to trigger any background trimming. The bounded values are 0 to one year, and you can use units in seconds (s), minutes (m), hours (h), days (d), weeks (w), months (m), and years (y).
 - trim-subnet-util-age***—How far back in time to trim the old subnet utilization data automatically, the preset value being 24 weeks. (However, the *trim-subnet-util-interval* value must be set to other than 0 for trimming to be in effect at all.) The bounded values are 24 hours to one year, and you can use units in seconds (s), minutes (m), hours (h), days (d), weeks (w), months (m), and years (y).
- Step 4** You can also force immediate trimming and compacting. Find the Trimming/Compacting section:
- Trim/Compact age**—How far in time to go back to trim the data. There are no bounds to this value. However, if you set a very small value (such as 1m), it trims or compacts very recent data, which can be undesirable. In fact, if you set it to zero, you lose all of the collected data. Setting the value too high (such as 10y) may end up not trimming or compacting any data.
 - Compact interval**—Time interval at which to compact the subnet utilization records older than the Trim/Compact age. This interval can be some multiple of the polling interval. For example, if the compact interval is set to twice the polling interval, it eliminates every other record.
- Step 5** If you are trimming immediately, click **Trim All Subnet Utilization** among the controls at the bottom of the page. If you are compacting the data, click **Compact All Subnet Utilization**.
-

Viewing Subnet Utilization History Data

The DHCP server gathers subnet utilization data into three broad categories:

- Active Reserved
- Active Unreserved
- Free Reserved

Each of these categories has a current value for a given collection interval, and low and high values over the life of the DHCP server.

To illustrate the three subnet utilization categories, consider this DHCP scope configuration:

```
Scope 10.10.10.0/24
Range 10.10.10.1 10.10.10.10
Range 10.10.10.20 10.10.10.30
Reservation 10.10.10.1 MAC-1
Reservation 10.10.10.2 MAC-2
Reservation 10.10.10.41 MAC-3
Reservation 10.10.10.42 MAC-4
```

Of the 254 potential leases, only 31 are configured, and two reservations are outside the address range.

Immediately after configuring the scope, adding the ranges and reservations, and reloading the DHCP server, these counters appear for subnet utilization:

```
Active Reserved 0
Active Unreserved 0
Free Unreserved 20
```

As soon as clients MAC-1 and MAC-2 get their reserved leases, subnet utilization then shows as:

```
Active Reserved 2
Active Unreserved 0
Free Unreserved 20
```

When the client MAC-5 gets lease 10.10.10.3, subnet utilization then shows as:

```
Active Reserved 2
Active Unreserved 1
Free Unreserved 19
```




CHAPTER 5

Managing Scopes, Prefixes, Links, and Networks

The Dynamic Host Configuration Protocol (DHCP) is an industry-standard protocol for automatically assigning IP configuration to devices. DHCP uses a client/server model for address allocation. As administrator, you can configure one or more DHCP servers to provide IP address assignment and other TCP/IP-oriented configuration information to your devices. DHCP frees you from having to manually assign an IP address to each client. The DHCP protocol is described in RFC 2131. For an introduction to the protocol, see [Introduction to Dynamic Host Configuration, on page 1](#).

This chapter describes how to set up scopes, prefixes, and links. Before clients can use DHCP for address assignment, you must add at least one scope (dynamic address pool) or prefix to the server.

- [Managing Scopes, on page 111](#)
- [DHCPv6 Addresses, on page 126](#)
- [Configuring Prefixes and Links, on page 130](#)
- [Managing DHCP Networks, on page 136](#)

Managing Scopes

This section describes how to define and configure scopes for the DHCP server. A scope consists of one or more ranges of dynamic addresses in a subnet that a DHCP server manages. You must define one or more scopes before the DHCP server can provide leases to clients. (For more on listing leases and defining lease reservations for a scope, see [Managing Leases, on page 185](#))

Related Topics

- [Creating and Applying Scope Templates, on page 139](#)
- [Creating Scopes, on page 112](#)
- [Getting Scope Counts on the Server, on page 120](#)
- [Configuring Multiple Scopes, on page 113](#)
- [Editing Scopes, on page 118](#)
- [Staged and Synchronous Mode, on page 119](#)
- [Configuring Embedded Policies for Scopes, on page 121](#)
- [Configuring Multiple Subnets on a Network, on page 121](#)

[Enabling and Disabling BOOTP for Scopes, on page 122](#)

[Disabling DHCP for Scopes, on page 124](#)

[Deactivating Scopes, on page 124](#)

[Setting Scopes to Renew-Only, on page 123](#)

[Setting Free Address SNMP Traps on Scopes, on page 123](#)

[Removing Scopes if Reusing Addresses, on page 125](#)

[Removing Scopes if Not Reusing Addresses, on page 125](#)

Creating Scopes

Creating scopes is a local cluster function. Each scope needs to have the following:

- Name
- Policy that defines the lease times, grace period, and options
- Network address and subnet mask
- Range or ranges of addresses

You can configure scopes at the local cluster only. The web UI pages are different for local basic and advanced modes.

Local Basic Web UI

-
- Step 1** From the **Design** menu, choose **Scopes** from the **DHCPv4** submenu to open the Manage Scopes page.
 - Step 2** Choose a VPN for the scope from the username drop-down list on the top right of the window, if necessary.
 - Step 3** Click the **Add Scopes** icon in the Scopes pane, enter a scope name, enter the subnet IP address and choose a mask value from the drop-down list.
 - Step 4** If desired, choose a preconfigured class of service (client-class) for the scope from the drop-down list.
 - Step 5** Click **Add DHCP Scope**.
 - Step 6** Reload the DHCP server.

Note When a scope is created in Basic mode, the range and the router address will be added automatically. If you want to change them, you have to change the mode to Advanced since it cannot be configured on the Basic mode.

Local Advanced Web UI

-
- Step 1** From the **Design** menu, choose **Scopes** from the **DHCPv4** submenu to open the List/Add DHCP Scopes page.
 - Step 2** Choose a VPN for the scope from the username drop-down list on the top right of the window, if necessary.
 - Step 3** Click the **Add Scopes** icon in the Scopes pane, enter a scope name, or leave it blank to use the one defined in the scope name expression of a scope template, if any (see [Using Expressions in Scope Templates, on page 146](#)). In the latter case, choose the scope template. You must always enter a subnet and mask for the scope.
 - Step 4** Choose a policy for the scope from the drop-down list. The policy defaults to the *default* policy.

Step 5 Click **Add DHCP Scope**.

Step 6 Add ranges for addresses in the scope. The ranges can be any subset of the defined scope, but cannot overlap. If you enter just the host number, the range is relative to the netmask. Do not enter ranges that include the local host or broadcast addresses (usually 0 and 255). Add the range and then click **Add Range**.

Step 7 Reload the DHCP server.

Tip To view any leases and reservations associated with the scope, see [Managing Leases, on page 185](#). To search for leases, see [Searching Server-Wide for Leases , on page 197](#).

Related Topics

[Getting Scope Counts on the Server, on page 120](#)

[Configuring Multiple Scopes, on page 113](#)

[Editing Scopes, on page 118](#)

[Staged and Synchronous Mode, on page 119](#)

Configuring Multiple Scopes

You can configure multiple scopes (with disjointed address ranges) with the same network number and subnet mask. By default, the DHCP server pools the available leases from all scopes on the same subnet and offers them, in a round-robin fashion, to any client that requests a lease. However, you can also bypass this round-robin allocation by setting an allocation priority for each scope (see [Configuring Multiple Scopes Using Allocation Priority, on page 114](#)).

Configuring the addresses of a single subnet into multiple scopes helps to organize the addresses in a more natural way for administration. Even though you can configure a virtually unlimited number of leases per scope, if you have a scope with several thousand leases, it can take a while to sort them. This can be a motivation to divide the leases among multiple scopes.

You can divide the leases among the scopes according to the types of leases. Because each scope can have a separate reservations list, you can put the dynamic leases in one scope that has a policy with one set of options and lease times, and all the reservations in another scope with different options and times. Note that in cases where some of the multiple scopes are not connected locally, you should configure the router (having BOOTP relay support) with the appropriate helper address.

Related Topics

[Configuring Multiple Scopes for Round-Robin Address Allocation, on page 113](#)

[Configuring Multiple Scopes Using Allocation Priority, on page 114](#)

Configuring Multiple Scopes for Round-Robin Address Allocation

By default, the DHCP server searches through the multiple scopes in a round-robin fashion. Because of this, you would want to segment the scopes by the kind of DHCP client requests made. When multiple scopes are available on a subnet through the use of secondary scopes, the DHCP server searches through all of them for one that satisfies an incoming DHCP client request. For example, if a subnet has three scopes, only one of which supports dynamic BOOTP, a BOOTP request for which there is no reservation is automatically served by the one supporting dynamic BOOTP.

You can also configure a scope to disallow DHCP requests (the default is to allow them). By using these capabilities together, you can easily configure the addresses on a subnet so that all the DHCP requests are satisfied from one scope (and address range), all reserved BOOTP requests come from a second one, and all dynamic BOOTP requests come from a third. In this way, you can support dynamic BOOTP while minimizing the impact on the address pools that support DHCP clients.

Configuring Multiple Scopes Using Allocation Priority

As of Cisco Prime Network Registrar Release 6.1, you can set an allocation priority among scopes instead of the default round-robin behavior described in the previous section. In this way, you can have more control over the allocation process. You can also configure the DHCP server to allocate addresses contiguously from within a subnet and control the blocks of addresses allocated to the backup server when using DHCP server failover (see [Managing DHCP Failover, on page 55](#)).

A typical installation would set the allocation priority of every scope by using the *allocation-priority* attribute on the scope. Some installations might also want to enable the *allocate-first-available* attribute on their scopes, although many would not. There is a small performance loss when using *allocate-first-available*, so you should only use it when absolutely required.

You can control:

- A hierarchy among scopes of which should allocate addresses first.
- Whether to have a scope allocate the first available address rather than the default behavior of the least recently accessed one.
- Allocating contiguous and targeted addresses in a failover configuration for a scope.
- Priority address allocation server-wide.
- In cases where the scopes have equal allocation priorities set, whether the server should allocate addresses from those with the most or the least number of available addresses.

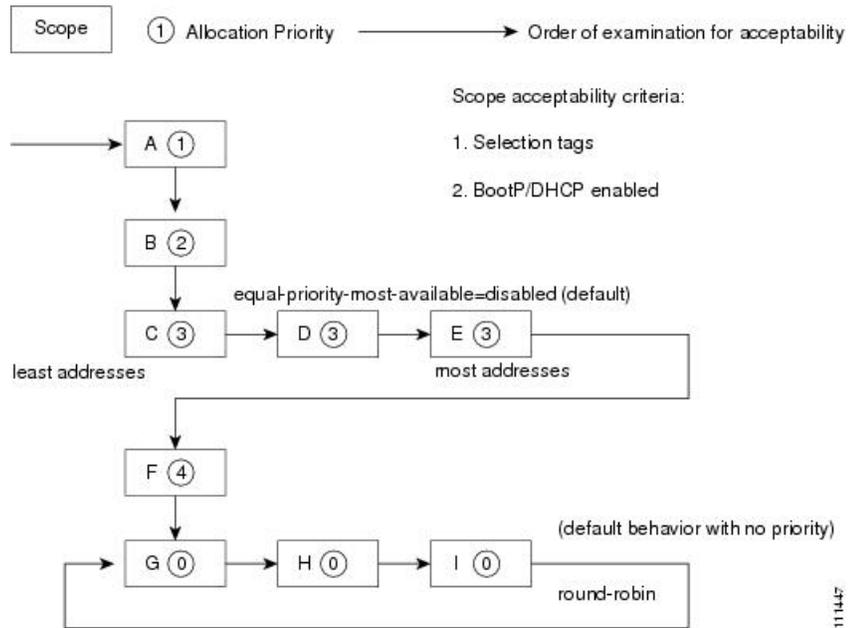
When there is more than one scope in a network, then the DHCP must decide which scope to allocate an IP address from when it processes a DHCPDISCOVER request from a DHCP client that is not already associated with an existing address. The algorithm that the DHCP server uses to perform this allocation is described in the following section.

Allocation Priority Algorithm

The DHCP server examines the scopes in a network one at a time to determine if they are acceptable. When it finds an acceptable scope, it tries to allocate an IP address from it to fulfill the DHCPDISCOVER request. The *allocation-priority* scope attribute is used to direct the DHCP server to examine the scopes in a network in a particular order, because in the absence of any allocation priority, the DHCP server examines the scopes in a round-robin order.

The image below shows an example of a network with nine scopes (which is unusual, but serves to illustrate several possibilities of using allocation priority).

Figure 11: Scope Allocation Priority



Six of these scopes were configured with an allocation priority, and three of them were not. The server examines the six that were configured with an allocation priority first, in lowest to highest priority order. As the server finds an acceptable scope, it tries to allocate an IP address from it. If the server succeeds, it then finishes processing the DHCPDISCOVER request using this address. If it cannot allocate an address from that scope, it continues examining scopes looking for another acceptable one, and tries to allocate an address from it.

This process is straightforward if no scopes have the same allocation priority configured, but in the case where (as in the example in) more than one scope has the same nonzero allocation priority, then the server has to have a way to choose between the scopes of equal priority. The default behavior is to examine the scopes with equal priority starting with the one with the fewest available addresses. This uses up all of the addresses in one scope before using any others from another scope. This is the situation shown in the image above. If you enable the *equal-priority-most-available* DHCP server attribute, then the situation is reversed and the scope with the most available addresses is examined first when two scopes have equal priority. This spreads out the utilization of the scopes, and more or less evenly distributes the use of addresses across all of the scopes with equal allocation priority set.

You can use this *equal-priority-most-available* approach because of another feature in the processing of equal priority scopes. In the situation where there are two scopes of equal priority, if the DHCPDISCOVER request, for which the server is trying to allocate an address, also has a *limitation-id* (that is, it is using the option 82 limitation capability; see [Subscriber Limitation Using Option 82, on page 297](#)), then the DHCP server tries to allocate its IP address from the same scope as that used by some existing client with the same *limitation-id* (if any). Thus, all clients with the same *limitation-id* tend to get their addresses allocated from the same scope, regardless of the number of available addresses in the scopes of equal priority or the setting of the *equal-priority-most-available* server attribute.

To bring this back to the *equal-priority-most-available* situation, you might configure *equal-priority-most-available* (and have several equal priority scopes), and then the first DHCP client with a particular *limitation-id* would get an address from the scope with the most available addresses (since there are no other clients with that same *limitation-id*). Then all of the subsequent clients with the same *limitation-id* would go into that same scope. The result of this configuration is that the first clients are spread out evenly among the

acceptable, equal priority scopes, and the subsequent clients would cluster with the existing ones with the same *limitation-id*.

If there are scopes with and without allocation priority configured in the same network, all of the scopes with a nonzero allocation priority are examined for acceptability first. Then, if none of the scopes were found to be acceptable and also had an available IP address, the remaining scopes without any allocation priority are processed in a round-robin manner. This round-robin examination is started at the next scope beyond the one last examined in this network, except when there is an existing DHCP client with the same *limitation-id* as the current one sending the DHCPDISCOVER. In this case, the round-robin scan starts with the scope from which the existing client IP address was drawn. This causes subsequent clients with the same *limitation-id* to draw their addresses from the same scope as the first client with that *limitation-id*, if that scope is acceptable and has available IP addresses to allocate.

Address Allocation Attributes

The attributes that correspond to address allocation are described in the table below.

Table 16: Address Allocation Priority Settings

Attribute	Type	Description
<i>allocation-priority</i>	Scope (set or unset)	<p>If defined, assigns an ordering to scopes such that address allocation takes place from acceptable scopes with a higher priority until the addresses in all those scopes are exhausted. An allocation priority of 0 (the preset value) means that the scope has no allocation priority. A priority of 1 is the highest priority, with each increasing number having a lower priority. You can mix scopes with an allocation priority along with those without one. In this case, the scopes with a priority are examined for acceptability before those without a priority.</p> <p>If set, this attribute overrides the DHCP server <i>priority-address-allocation</i> attribute setting. However, if <i>allocation-priority</i> is unset and <i>priority-address-allocation</i> is enabled, then the allocation priority for the scope is its subnet address. With <i>allocation-priority</i> unset and <i>priority-address-allocation</i> disabled, the scope is examined in the default round-robin fashion.</p>
<i>allocate-first-available</i>	Scope (enable or disable)	<p>If enabled, forces all allocations for new addresses from this scope to be from the first available address. If disabled (the preset value), uses the least recently accessed address. If set, this attribute overrides the DHCP server <i>priority-address-allocation</i> attribute setting. However, if unset and <i>priority-address-allocation</i> is enabled, then the server still allocates the first available address. With <i>allocate-first-available</i> unset and <i>priority-address-allocation</i> disabled, the scope is examined in the default round-robin fashion.</p>

Attribute	Type	Description
<i>failover-backup-allocation-boundary</i>	Scope (set or unset)	<p>If <i>allocate-first-available</i> is enabled and the scope is in a failover configuration, this value is the IP address to use as the point from which to allocate addresses to a backup server. Only addresses below this boundary are allocated to the backup server. If there are no available addresses below this boundary, then the addresses above it are allocated to the backup server. The actual allocation works down from this address, while the normal allocation for DHCP clients works up from the lowest address in the scope.</p> <p>If this attribute is unset or set to zero, then the boundary used is halfway between the first and last addresses in the scope ranges. If there are no available addresses below this boundary, then the first available address is used.</p> <p>See Figure 12: Address Allocation with allocate-first-available Set for an illustration of how addresses are allocated in a scope using this setting.</p>
<i>priority-address-allocation</i>	DHCP (enable or disable)	<p>Provides a way to enable priority address allocation for the entire DHCP server without having to configure it on every scope. (However, the scope <i>allocation-priority</i> setting overrides this one.) If <i>priority-address-allocation</i> is enabled and the scope <i>allocation-priority</i> attribute is unset, then the scope subnet address is used for the allocation priority. If the scope <i>allocate-first-available</i> is unset, then priority address allocation is considered enabled. Of course, when exercising this overall control of the address allocation, the actual priority of each scope depends only on its subnet address, which may or may not be desired.</p>
<i>equal-priority-most-available</i>	DHCP (enable or disable)	<p>By default, when two or more scopes with the same nonzero <i>allocation-priority</i> are encountered, the scope with the least available IP addresses is used to allocate an address for a new client (if that client is not in a limitation list). If <i>equal-priority-most-available</i> is enabled and two or more scopes have the same nonzero allocation priority, then the scope with the most available addresses is used to allocate an address for a new client (if that client is not in a limitation list). In either case, if a client is in a limitation-list, then among those scopes of the same priority, the one that contains other clients in the same list is always used.</p>

Allocating Addresses In Scopes

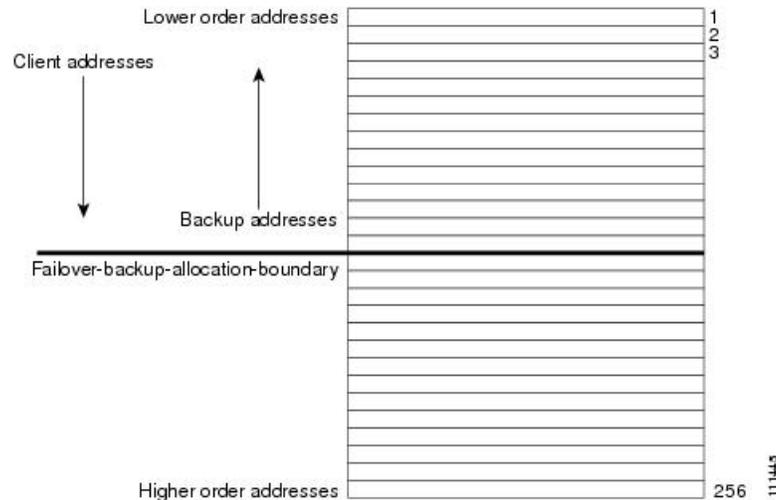
When trying to allocate an IP address from within a scope, the default action of the DHCP server is to try to allocate the least recently accessed address first, from the list of available leases. But all the operations that require accessing the lease like listing all the leases or all leases in a scope, asking for a specific lease (`nr cmd> lease addr`), searching leases, or modifying leases (activate, deactivate, or force available) affect the ordering of the leases in the list of available leases with the server.

Operating on a single lease places that lease at the end of the list. Listing leases causes the leases to be arranged in numerical order, making the lowest numbered lease to end up first on the available list. Other operations that require the server to access the lease, like leasequery requests also impacts the order of leases.

Thus, in general there is no way to predict which IP address within a scope is allocated at a given time. Usually this poses no difficulty, but there are times when a more deterministic allocation strategy is desired. To configure a completely deterministic address allocation strategy, you can enable the *allocate-first-available* attribute on a scope. This causes the available address with the lowest numeric value to be allocated for a

DHCP client. Thus, the first client gets the first address in the lowest range, and the second client the second one in that range, and so on. This is shown in the image below.

Figure 12: Address Allocation with *allocate-first-available* Set



Note that there is some minor performance cost to this deterministic allocation strategy, not so much that you should not use it, but possibly enough so that you should not use it if you do not need it. When using this deterministic allocation strategy approach in a situation where the scope is in a failover relationship, the question of how to allocate the available IP addresses for the backup server comes up on the main server. By default, the address halfway between the lowest and highest ones in the scope becomes the *failover-backup-allocation-boundary*. The available addresses for the backup server are allocated working down from this boundary (if any addresses are available in that direction). If no address is available below this boundary, then the first available one above the boundary is used for the backup server. You can configure the *failover-backup-allocation-boundary* for the scope if you want to have a different address boundary than the halfway point.

You would use a deterministic allocation strategy and configure *allocate-first-available* in situations where you might allocate a scope with a larger number of IP addresses than you were sure you needed. You can later shrink back the ranges in the scope so as to allow moving address space to another network or server. In the nondeterministic approach, the allocated addresses are scattered all over the ranges, and it can be very hard to reconfigure the DHCP clients to free up, say, half of the scope addresses. However, if you configure *allocate-first-available*, then the allocated addresses tend to cluster low in the scope ranges. It is then probably simpler to remove ranges from a scope that does not need them, so that those addresses can be used elsewhere.

Editing Scopes



Note You can only make changes to a scope's subnet, if there are no reservations or ranges that conflicts with the change, either in the current scope or any other scope with the same old subnet as those scopes' subnet will also be changed.

Local Advanced Web UI

- Step 1** Create a scope, as described in [Creating Scopes, on page 112](#).
- Step 2** Reload the DHCP server.
- Step 3** Click the name of the scope on the Scopes pane on the List/Add DHCP Scopes page to open the Edit DHCP Scope page. (If a server reload is required, a status message indicates it and you must reload first before proceeding.)
- Step 4** Modify the fields or attributes as necessary. You can also modify the name of the scope.
- Step 5** To edit the scope embedded policy, see [Configuring Embedded Policies for Scopes, on page 121](#). To list leases for the scope, see [Viewing Leases, on page 191](#).
- Step 6** Click **Save**.
- Step 7** Reload the DHCP server.
-

CLI Commands

After you create a scope, look at the properties for all the scopes on the server, use **scope list** (or **scope listnames**, **scope listbrief**, **scope name show**, or **scope name get attribute**). Then:

- To reset an attribute, use **scope name set attribute=value [attribute=value ...]**. For example, you can reset the name of the scope by using **scope name set name =new name**
- To enable or disable an attribute, use **scope name enable attribute** or **scope name disable attribute**.

See the **scope** command in the CLIGuide.html file in the /docs directory for syntax and attribute descriptions.

Related Topics

- [Staged and Synchronous Mode, on page 119](#)
- [Configuring Embedded Policies for Scopes, on page 121](#)
- [Configuring Multiple Subnets on a Network, on page 121](#)
- [Enabling and Disabling BOOTP for Scopes, on page 122](#)
- [Disabling DHCP for Scopes, on page 124](#)
- [Deactivating Scopes, on page 124](#)
- [Setting Scopes to Renew-Only, on page 123](#)
- [Setting Free Address SNMP Traps on Scopes, on page 123](#)
- [Removing Scopes if Reusing Addresses, on page 125](#)
- [Removing Scopes if Not Reusing Addresses, on page 125](#)

Staged and Synchronous Mode

New scopes or modifications to scopes can be in one of two modes—staged or synchronous:

- **Staged**—New scopes or modifications to existing scopes are written to the database, but not propagated to the DHCP server until the DHCP server is reloaded.
- **Synchronous**—Most new scopes and scope modifications (including deletions) are immediately propagated to the DHCP server (without the need for a reload). Not all scope changes are possible. For

example, changing the primary subnet on a scope is not allowed (a reload is required to effect the change). Furthermore, only scope attribute changes can be propagated without a reload. For example, changes to named policies require a DHCP server reload.

If you add or modify a scope while in staged mode and then change the dhcp edit mode to synchronous, the first change in synchronous mode applies all pending changes for that scope (not just the ones made while in synchronous mode).



Note In Cisco Prime Network Registrar versions earlier than Release 7.1, the dhcp edit mode was called scope edit mode.

Local Basic or Advanced Web UI

To view the current dhcp edit mode or change the dhcp edit mode, click the *username* drop-down list on the top right of the window and choose **Session Settings**. If the scope is up to date in the DHCP server, the **Total synchronized scopes** message appears on the List/Add DHCP Scopes page (in Advanced mode) and the **Scope status: synchronized** message appears on the Edit DHCP Scope page (in both modes). If the scope is not up to date, the **Scope name status: reload required** message is displayed.

CLI Commands

View the dhcp edit mode by using **session get dhcp-edit-mode**, or set the dhcp edit mode using **session set dhcp-edit-mode={sync | staged}**. To view the scopes that are not synchronized with the DHCP server, use **scope report-staged-edits**. For example:

```
nrcmd> scope report-staged-edits

100 Ok
example-scope: [reload-required]
```

Getting Scope Counts on the Server

You can view the created scopes associated with the DHCP server, hence obtain a count, in the web UI.

CLI Commands

Using the CLI, you can get an exact count of the total scopes for the DHCP server by using **dhcp getScopeCount [vpn name | all]**. You can specify a VPN or all VPNs. Omitting the **vpn name** returns a count for the current VPN. Specifying a failover pair name returns the total scopes and networks for the failover pair. Because a failover pair definition includes explicit VPN settings in its matchlist, these counts are not limited to the current VPN only.

To create a scope, use **scope name create address mask [template=template-name] [attribute=value ...]**. Each scope must identify its network address and mask. When you create the scope, Cisco Prime Network Registrar places it in its current virtual private network (VPN), as defined by **session set current-vpn**. You cannot change the VPN once you set it at the time of creation of the scope.

To set a policy for the scope, use **scope name set policy**.

To add a range of IP addresses to the scope, use **scope name addRange start end**.

Configuring Embedded Policies for Scopes

When you create a scope, Cisco Prime Network Registrar automatically creates an embedded policy for it. However, the embedded policy has no associated properties or DHCP options until you enable or add them. An embedded policy can be useful, for example, in defining the router for the scope. As [Types of Policies, on page 163](#) describes, the DHCP server looks at the embedded policy of a scope before it looks at its assigned, named policy.



Note If you delete a scope policy, you remove all of its properties and attributes.

Local Advanced Web UI

- Step 1** Create a scope, as described in [Creating Scopes, on page 112](#).
 - Step 2** Click the name of the scope on the Scopes pane on the List/Add DHCP Scopes page to open the Edit DHCP Scope page.
 - Step 3** Click **Create New Embedded Policy** to create a new embedded policy, or **Edit Existing Embedded Policy** if there is already an existing one, to open the Edit DHCP Embedded Policy for Scope page.
 - Step 4** Modify the fields, options, and attributes on this page. If necessary, unset attributes.
 - Step 5** Click **Save**.
-

CLI Commands

Create a scope first. In the CLI, **scope-policy** uses the same syntax as **policy**, except that it takes the scope name as an argument. Then, to:

- Determine if there are any embedded property values already set for a scope, use **scope-policy scope-name show**.
- Enable or disable an attribute, use **scope-policy name enable attribute** or **scope-policy name disable attribute**.
- Set and unset attributes, use **scope-policy name set attribute=value [attribute=value...]** and **scope-policy name unset attribute**.
- List, set, and unset vendor options (see [Using Standard Option Definition Sets, on page 172](#)).

Configuring Multiple Subnets on a Network

Cisco Prime Network Registrar supports multiple logical subnets on the same network segment, which are also called secondary subnets. With several logical subnets on the same physical network, for example, 192.168.1.0/24 and 192.168.2.0/24, you might want to configure DHCP so that it offers addresses from both pools. By pooling addresses this way, you can increase the available number of leases.

To join two logical subnets, create two scopes, and elect one to be primary and the other to be a secondary. After you configure the secondary subnet, a new client on this physical network gets a lease from one or the other scope on a round-robin basis.

Local Advanced Web UI

- Step 1** Create a scope (see [Creating Scopes, on page 112](#)) that you will make a secondary scope.
- Step 2** Click the name of the scope on the Scopes pane on the List/Add DHCP Scopes page to open the Edit DHCP Scope page.
- Step 3** To make this a secondary scope, enter the network address of the subnet of the primary scope in the *Primary Subnet* attribute field in the Edit DHCP Scope page.

It is common practice for the *primary-subnet* to correspond directly to the network address of the primary scope or scopes. For example, with *examplescope1* created in the 192.168.1.0/24 network, associate *examplescope2* with it using *primary-subnet=192.168.1.0/24*. (Note that if Cisco Prime Network Registrar finds that the defined subnet has an associated scope, it ignores the mask bit definition and uses the one from the primary scope, just in case they do not match.) However, the *primary-subnet* can be a subnet address that does not have a scope associated with it.

- Step 4** Click **Save**.
- Step 5** Restart or reload the server.
-

CLI Commands

To assign the secondary scope to a primary one, use **scope name set primary-subnet=value**, then reload the server.

To remove the secondary scope, use **scope name unset primary-subnet**. When setting the *primary-subnet* attribute, include the number bits for the network mask, using slash notation. For example, represent the network 192.168.1.0 with mask 255.255.255.0 as 192.168.1.0/24. The mask bits are important. If you omit them, a /32 mask (single IP address) is assumed.

Enabling and Disabling BOOTP for Scopes

The BOOTstrap Protocol (BOOTP) was originally created for loading diskless computers. It was later used to allow a host to obtain all the required TCP/IP information so that it could use the Internet. Using BOOTP, a host can broadcast a request on the network and get the data required from a BOOTP server. The BOOTP server listens for incoming requests and generates responses from a configuration database for the BOOTP clients on that network. BOOTP differs from DHCP in that it has no concept of lease or lease expiration. All addresses that a BOOTP server allocates are permanent.

You can configure the Cisco Prime Network Registrar DHCP server to act like a BOOTP server. In addition, although BOOTP normally requires static address assignments, you can choose either to reserve addresses (and use static assignments) or have addresses dynamically allocated (known as *dynamic BOOTP*).

When you need to move or decommission a BOOTP client, you can reuse its lease simply by forcing lease availability. See [Forcing Lease Availability, on page 211](#).

Local Advanced Web UI

On the Edit DHCP Scope page, under BootP Settings, enable the *bootp* attribute for BOOTP, or the *dynamic-bootp* attribute for dynamic BOOTP. They are disabled by default. Then click **Save**.

CLI Commands

Use `scope name enable bootp` to enable BOOTP, and `scope name enable dynamic-bootp` to enable dynamic BOOTP. Reload the DHCP server (if in staged dhcp edit mode).

Setting Scopes to Renew-Only

You can control whether to allow existing clients to re-acquire their leases, but not to offer any leases to new clients. A renew-only scope does not change the client associated with any of its leases, other than to allow a client currently using an available IP address to continue to use it.

Local Advanced Web UI

On the Edit DHCP Scope page, under Miscellaneous Settings, explicitly enable the *renew-only* attribute. Then click **Save**.

CLI Commands

Use `scope name enable renew-only` to set a scope to renew-only.

Setting Free Address SNMP Traps on Scopes

You can set SNMP traps to capture unexpected free address events by enabling the traps and setting the low and high thresholds for a scope. You can also set traps based on networks and selection tags instead of scopes.

When setting the threshold values, it is advisable to maintain a small offset between the low and high values, as described in the *"Simple Network Management" section in Cisco Prime Network Registrar 9.0 Administrator Guide*). The offset can be as little as 5%, for example, a low value of 20% and a high value of 25%, which are the preset values.

Here are some variations on how you can set the server and scope values for these attributes:

- Get each scope to trap and reset the free address values based on the server settings, as long as at least one recipient is configured.
- Disable the traps at the scope level or specify different percentages for each scope.
- Disable the traps globally on the server, but turn them on for different scopes.
- Set the traps at the network level or selection tags level.

Local Advanced Web UI

-
- Step 1** To create a trap configuration, from the **Deploy** menu, choose **Traps** under the **DHCP** submenu to open the List/Add Trap Configurations page.
- Step 2** Click the **Add Traps** icon, enter a name for the trap configuration, choose **scope** from the mode drop-down list, and enter the low and high threshold values (they are 20% and 25%, respectively, by default). Click **Add AddrTrapConfig**. (You can go back to edit these values if you need to.)
- Step 3** Edit the created scope to which you want to apply the threshold settings. Under SNMP Trap Settings, enter the name of the trap in the *free-address-config* attribute field. Click **Save**.
-

CLI Commands

Use **addr-trap name create** to add a trap configuration. To set the thresholds, use the **addr-trap name set** method (or include the threshold settings while creating the trap). For example:

```
nrcmd> addr-trap trap-1 create
```

```
nrcmd> addr-trap trap-1 set low-threshold
```

```
nrcmd> addr-trap trap-1 set high-threshold
```

To set the free-address trap, use **scope name set free-address-config=trap-name**. For example:

```
nrcmd> scope scope-1 set free-address-config=trap-1
```

Disabling DHCP for Scopes

You can disable DHCP for a scope if you want to use it solely for BOOTP. See [Enabling and Disabling BOOTP for Scopes, on page 122](#). You can also temporarily deactivate a scope by disabling DHCP, but deactivation is more often used if you are enabling BOOTP. See [Deactivating Scopes, on page 124](#).

Local Advanced Web UI

On the Edit DHCP Scope page, under BootP Settings, disable the *dhcp* attribute and enable the *bootp* attribute. Then click **Save**.

CLI Commands

Use **scope name disable dhcp** to disable DHCP. You should also enable BOOTP and reload the server (if in staged dhcp edit mode).

Deactivating Scopes

You might want to temporarily deactivate all the leases in a scope. To do this, you must disable both BOOTP and DHCP for the scope.

Local Advanced Web UI

On the Edit DHCP Scope page, under Miscellaneous Settings, explicitly enable the *deactivated* attribute. Then click **Save**.

CLI Commands

Use **scope name enable deactivated** to disable BOOTP and DHCP for the scope. Reload the DHCP server (if in staged dhcp edit mode).

Removing Scopes



Caution Although removing a scope from a DHCP server is easy to do, be careful. Doing so compromises the integrity of your network. There are several ways to remove a scope from a server, either by re-using or not re-using addresses, as described in the following sections.

DHCP, as defined by the IETF, provides an address lease to a client for a specific time (defined by the server administrator). Until that time elapses, the client is free to use its leased address. A server cannot revoke a lease and stop a client from using an address. Thus, while you can easily remove a scope from a DHCP server, the clients that obtained leases from it can continue to do so until it expires. This is true even if the server does not respond to their renewal attempts, which happens if the scope was removed.

This does not present a problem if the addresses you remove are not reused in some way. However, if the addresses are configured for another server before the last lease expires, the same address might be used by two clients, which can destabilize the network.

Cisco Prime Network Registrar moves the leases on the removed scope to an orphaned leases pool. When creating a scope, orphaned leases are associated with appropriate scopes.

Removing Scopes if Not Reusing Addresses

You can remove scopes if you are not reusing addresses.

Local Basic or Advanced Web UI

If you are sure you do not plan to reuse the scope, on the Manage Scopes or List/Add DHCP Scopes page, click the **Delete Scopes** icon in the Scopes pane after selecting the name, and confirm or cancel the deletion.

CLI Commands

Be sure that you are not immediately planning to reuse the addresses in the scope, then use **scope name delete** to delete it.

Removing Scopes if Reusing Addresses

If you want to reuse the addresses for a scope you want to remove, you have two alternatives:

- **If you can afford to wait until all the leases in the scope expire**—Remove the scope from the server, then wait for the longest lease time set in the policy for the scope to expire. This ensures that no clients are using any addresses from that scope. You can then safely reuse the addresses.
- **If you cannot afford to wait until all the leases in the scope expire**—Do not remove the scope. Instead, deactivate it. See [Deactivating Scopes, on page 124](#). Unlike a removed scope, the server refuses all clients' renewal requests, which forces many of them to request a new lease. This moves these clients more quickly off the deactivated lease than for a removed scope.

You can use the **ipconfig** utility in Windows to cause a client to release (**/release**) and re-acquire (**/renew**) its leases, thereby moving it off a deactivated lease immediately. You can only issue this utility from the client machine, which makes it impractical for a scope with thousands of leases in use. However, it can be useful in moving the last few clients in a Windows environment off deactivated leases in a scope.

DHCPv6 Addresses

Cisco Prime Network Registrar supports the following IPv6 addressing for DHCP (DHCPv6):

- **Stateless autoconfiguration (RFC 3736)** -The DHCPv6 server does not assign addresses, but instead provides configuration parameters, such as DNS server data, to clients.
- **Stateful autoconfiguration (RFC 3315)** -The DHCPv6 server assigns nontemporary or temporary addresses and provides configuration parameters to clients.
- **Prefix Delegation (RFC 3633)** -The DHCPv6 server delegates prefixes to clients (routers).

The DHCPv6 service provides these capabilities:

- **Allocation groups** -Allows multiple prefixes to be treated as one from an allocation standpoint, and provides control over the order in which the prefixes are used.
- **Client-classing** -You can classify clients and select prefixes based on known clients or packet-based expressions.
- **DNS Updates** -DNS server updates of DHCP activity (over IPv4).
- **Extensions** -Extend the DHCP server processing by using C/C++ and Tcl extensions.
- **Failover** -You can configure a DHCP failover pair so that if one cannot provide leases to requesting clients, another one can take over.
- **LDAP** -Allows client entry lookups in an LDAP repository (external to Cisco Prime Network Registrar) and these clients may specify client reservations.
- **Leasequery** -Offers leasequery support.
- **Links and prefixes** -Similar to DHCPv4 networks and scopes that define the network topology. Each link can have one or more prefixes.
- **Policies and options** -You can assign attributes and options to links, prefixes, and clients.
- **Prefix Stability** -Clients can retain the delegated prefix when they change their location, that is even when they move from one CMTS to another or move within an address space. Prefix Stability, with appropriate infrastructure support (CMTS, routers), allows the subscriber to be moved or move without requiring a different delegated prefix.
- **SNMP traps** -Generate traps for events, such as if the number of leases in a prefix exceeds a certain limit (or drops below a certain limit) or if the server detects duplicate addresses.
- **Reservations** -Clients can receive predetermined addresses.
- **Statistics collection and logging** -Provides server activity monitoring.
- **VPN support** -Provides multiple address spaces (virtual private networks).

The DHCPv6 service requires that the server operating system support IPv6 and that you configure at least one interface on the system for IPv6.

IPv6 Addressing

IPv6 addresses are 128 bits long and are represented as a series of 16-bit hexadecimal fields separated by colons (:). The A, B, C, D, E, and F in hexadecimal are case insensitive. For example:

```
2001:db8:0000:0000:0000:0000:0000:0000
```

A few shortcuts to this addressing are:

- Leading zeros in a field are optional, so that you can write 09c0 as 9c0, and 0000 as 0.

- You can represent successive fields of zeros (any number of them) by a double colon (::), but only once in an address (because, if used more than once, the address parser has no way of identifying the size of each block of zeros). This reduces the length of addresses; for example, 2001:db8:0000:0000:0000:0000:0000:0000 can be written:

2001:db8::

Link-local addresses have a scope limited to the link, and use the prefix fe80::/10. Loopback addresses have the address ::1. Multicast addresses have the prefix ff00::/8 (there are no broadcast addresses in IPv6).

The IPv4-compatible addresses in IPv6 are the IPv4 decimal quad addresses prefixed by ::. For example, you can write the IPv4 address interpreted as ::c0a8:1e01 in the form ::192.168.30.1.

Determining Links and Prefixes

When the DHCPv6 server receives a DHCPv6 message, it determines the links and prefixes it uses to service the request. The server:

1. Finds the source address:
 1. If the client message was relayed, the server sets the source address to the first nonzero link-address field starting with the Relay-Forward message closest to the client (working outwards). If the server finds a source address, it proceeds to step **Step 2**.
 2. Otherwise, if the message source address is a link-local address, the server sets the source address to the first address for the interface on which it received the message for which a prefix exists (or 0 if it finds no prefix for any address). It then proceeds to step **Step 2**.
 3. Otherwise, the server sets the source address to the message source address.
2. Locates the prefix for the source address. If the server cannot find a prefix for the source address, it cannot service the client and drops the request.
3. Locates the link for the prefix. This always exists and is either an explicitly configured link or the implicitly created link based on the prefix address. The link must be a topological link (see [Prefix Stability, on page 128](#)).

Now that the server can determine the client link, it can process the client request. Depending on whether the client request is stateful or prefix-delegated, and on the selection criteria and other factors, the server might use one or more prefixes for the link to service the client request.

This is one area of difference between DHCPv4 and DHCPv6. In DHCPv4, the server selects only one of the scopes from the network to service the client request. In DHCPv6, the server can use all the prefixes for the link. Thus, the server might assign a client an address, or delegate a prefix, from multiple prefixes for the link (subject to selection criteria and other conditions). (See [Creating and Editing Links, on page 134](#))

Generating Addresses

IPv6 addresses are 128-bit addresses (as compared to 32-bit addresses for IPv4). In most cases, DHCPv6 servers assign 64 of those bits, the interface-identifier (EUI-64) portion (see RFC 4291). You can generate addresses by using the client 64-bit interface-identifier or a random number generator. The interface-identifier emulates how stateless autoconfiguration assigns addresses to clients. Unfortunately, there are privacy concerns regarding its use, and it is limited to one address per prefix for the client.

By default, Cisco Prime Network Registrar generates an address using an algorithm similar to that described in RFC 4941 to generate a random interface identifier. These random interface identifiers have a zero value

for the universal/local bit to distinguish them from EUI-64-based identifiers. The server also skips randomly generated interface identifiers from `::0` to `::ff` so that you can use identifiers for infrastructure devices (such as routers). You can configure whether to assign the interface-identifier (if available) first for each prefix (through the interface-identifier flag of the prefix *allocation-algorithm* s attribute). (See [Creating and Editing Prefixes, on page 131](#).) If you specify use of the interface-identifier, the server might still use randomly generated addresses if the address is not available to the client, or the client requests multiple addresses on a prefix.

The server generates addresses based on the prefix-configured range (or the prefix address if there is no range). If the range prefix length is shorter than 64, the server supplies only 64 bits and places them in the address interface-identifier field. If the prefix length is longer than 64, the server supplies only the remaining bits of the address. Thus, a /96 range uses 96 bits from the specified range followed by 32 bits of either the client interface-identifier or a randomly generated value. If the resulting address is not available (such as if it is already leased to another client, or to the same client, but on a different binding), the server tries to generate another address. It repeats this process up to at most 500 times.

When DHCP failover is configured, the server generated addresses are always odd addresses on the main and even addresses on the backup.



Note The DHCP server tests only the randomly generated interface identifier for values from `::0` to `::ff`, not the resulting address. Thus, a randomly generated address may end up using an `xxxx :xxxx :xxxx :xxxx ::0` through `xxxx :xxxx :xxxx :xxxx ::ff` address if the length of the prefix is longer than /64 and the prefix bits that extend beyond the /64 boundary are all zero.



Tip You can also choose from additional address generation algorithms for a prefix and prefix template; see [Creating and Editing Prefix Templates, on page 141](#).

Generating Delegated Prefixes

The DHCPv6 server uses the best first-fit algorithm when generating delegated prefixes. The server uses the first longest available prefix of the length configured or requested.

For DHCP failover, each server only considers the delegated prefix leases in the available state. When the server is in the PARTNER-DOWN state, the server can also use leases in the other-available or pending-available states after certain time restrictions have passed.

Prefix Stability

Prefix Stability is introduced in Cisco Prime Network Registrar 8.1 to let you control prefix delegation independent of the network topology. A new link attribute *type* specifies the type of link.

There are three different link types:

- **Topological**—This is the link type supported in 8.0 and earlier. A client on a topological link is allocated leases based on the network segment it is connected to.
- **Location independent**—This link type is introduced to support the CableLabs DOCSIS 3.0 concept of CMTS prefix stability. It supports service provider load balancing and reconfiguration events within a group of CMTS (such as in a central office). A subscriber that is moved from one CMTS to another on

a location-independent link can retain a delegated prefix. This link type allows movement within a single DHCP server.

- **Universal**—This link type is introduced to let subscribers retain a delegated prefix anywhere in the network. Use of this link type requires administrative assignment of the delegated prefixes and use of client or lease reservations. It can be deployed across multiple DHCP servers.



Note Use of prefix stability has routing implications and requires appropriate support from relay agents (that is, CMTS) in order to advertise the routes. For CMTS prefix stability, these are localized to the CMTS group. The implications are greater for universal prefix stability as routes need to be advertised throughout the service providers network.

CMTS Prefix Stability

Location independent links implement the CableLabs DOCSIS 3.0 requirements for CMTS prefix stability. CMTS prefix stability is possible as long as all prefixes are serviced by a single DHCP server.

If you want to introduce CMTS prefix stability in a particular area, you need to:

- Modify existing links to specify the same link group name across all of the links within the group. Each CMTS (or CMTS bundle) will have a separate link, but all of these links within the area for which CMTS Prefix Stability is desired need to be made part of the same link group.
- Create a new link, flagged as location-independent and made part of this link group. Create or move one or more prefix delegation prefixes under this location-independent link - these are the prefixes from which the stable prefixes will be allocated.
- Remove any prefix delegation prefixes from the existing links that are no longer needed. Note that stateful prefixes (dhcp-type of dhcp) should not be removed.



Note You can have only one location independent link in a group.

When a client request is received, the server locates the link by checking for the longest matching prefix and using the link of the prefix. However, if this topological link is part of a link group and that group has a location-independent link, the prefixes under the location-independent link will be checked first for possible leases requested by the client. Only if no leases are available from this location-independent link will the topological link be used. This is used for each binding requested by the client.

Any leasing mechanism (lease or client reservations, first best-fit, or extension generated/supplied) may be used with CMTS Prefix Stability as the leases are only known within the single server that services the CMTS group.

Universal Prefix Stability

Universal Prefix Stability lets you retain a delegated prefix regardless of where you connect. To use this feature, you must configure reservations for the delegated prefixes. Either client and lease reservations can be used.

Client reservations let you specify the delegated prefixes in a central LDAP repository that the DHCP servers access dynamically (see [Using Client Reservations, on page 199](#)). Lease reservations are managed centrally on the CCM regional server, and are pushed to each local DHCP with the universal link. Because the complete

list of reservations is replicated on each server when using lease reservations, you should consider client reservations for larger deployments.



Note You can have only one universal link in a particular VPN address space.

If a link is configured with the universal link type, the prefixes in that link are considered first when attempting to allocate a lease for a client. If no lease is available, the prefixes in the location-independent link type from the link group (if any) is used. Finally, the prefixes in the topological link are used.



Note You can enable both CMTS Prefix Stability and Universal Prefix Stability at the same time, though only one will apply to a subscriber.

Prefix Allocation Groups

Cisco Prime Network Registrar 8.1 introduces prefix allocation groups to let you define multiple prefixes that do not result in multiple lease assignments to clients, and control the order in which the prefixes are used. The `allocation-group` and `allocation-group-priority` attributes are introduced to specify this behavior.

All prefixes on a link with the same allocation group name belong to that allocation group. A prefix with no allocation group name is in its own allocation group. At most one lease per binding is allocated across all the prefixes in the same allocation group.

The `allocation-group-priority` setting controls which prefixes are used. Lower numeric values have higher priority, except for 0 (the default), which has the lowest possible priority. Prefixes with the same priority are ordered by the active lease count, where the prefix with the lowest count will have the highest priority.



Note The `allocation-group` name is only specific to the link. Different links can reuse the same allocation group names.

Starting from Cisco Prime Network Registrar 8.2, to control the number of leases a client can obtain from an allocation group prefix you can set the `max-leases-per-binding` attribute for the DHCP policy. For example, if you set `max-leases-per-binding` as 1, the client can obtain only one lease from an allocation group prefix. In addition, if more than one lease is already allocated from the same allocation group prefix then the additional leases are revoked (usually the oldest lease is revoked).

Configuring Prefixes and Links

You can configure DHCPv6 prefixes and links directly, or you can create prefix or link templates for them first. See the following subsections:

- [Creating and Editing Prefix Templates, on page 141](#)
- [Creating and Editing Prefixes, on page 131](#)
- [Viewing Address Utilization for Prefixes, on page 103](#)

Creating and Editing Prefixes

You can create prefixes directly (and optionally apply an existing template to it; see [Creating and Editing Prefix Templates, on page 141](#)). These are the prefix attributes that you can set:

- **name**—Assigns a name to this prefix.
- **vpn-id**—VPN that contains the prefix.
- **description**—Describes the prefix.
- **dhcp-type**—Defines how DHCP manages address assignment for a prefix:
 - **dhcp (preset value)**—Uses the prefix for stateful address assignment.
 - **stateless**—Uses the prefix for stateless option configuration.
 - **prefix-delegation**—Uses the prefix for prefix delegation.
 - **infrastructure**—Uses the prefix to map a client address to a link, when the prefix does not have an address pool.
 - **parent**—Do not have DHCP use the prefix. But, use it as a container object to group child prefixes. Parent prefixes appear only in the IPv6 address space listing in the web UI, not in the prefixes listing.
- **address**—Prefix (subnet) to which an interface belongs to, using the high-order bits of an IPv6 address.
- **owner**—Owner of the prefix.
- **region**—Region for the prefix.
- **reverse-zone-prefix-length**—Prefix length of the reverse zone for ip6.arpa updates. (See [Determining Reverse Zones for DNS Updates, on page 248](#) for details.)
- **range**—Subrange the server can use to configure prefixes for address assignment. The prefix used depends on the value set for the *dhcp-type* attribute. If unset, the prefix address applies. This value can specify a longer prefix than the prefix address to limit the range of addresses or prefixes available for assignment. (See [Links and Prefixes, on page 4](#) for details.)
- **link**—Link associated with the prefix (subnet), used to group prefixes that are on a single link.
- **policy**—Shared policy to use when replying to clients.
- **selection-tags**—List of selection tags associated with the prefix.
- **allocation-algorithms**—One or more algorithms the server uses to select a new address or prefix to lease to a client. The available algorithms are:
 - **client-request (preset to off)**—Controls whether the server uses a client requested lease.
 - **reservation (preset to on)**—Controls whether the server uses an available reservation for the client.
 - **extension (preset to on)**—Controls whether the server calls extensions attached at the **generate-lease** extension point to generate an address or prefix for the client. When you use generate-lease extension point with DHCPv6 failover, the server uses the address or delegated prefix that the extension returns and does not perform a hash on this address or prefix as it does with the randomly generated addresses. If the extension is using some algorithmic method to generate the address or delegated prefix then the extension must be failover aware (extension will be able to determine if failover configuration is enabled and the role of the failover server). For details on extensions, see [Using Extension Points, on page 351](#).
 - **interface-identifier (preset to off)**—Controls whether the server uses the interface-identifier from the client (link-local) address to generate an address; ignored for temporary addresses and prefix delegation.
 - **random (preset to on)**—Controls whether the server generates an address using an RFC 3041 algorithm; ignored for prefix delegation.
 - **best-fit (preset to on)**—Controls whether the server delegates the first, best-fit available prefix; ignored for addresses.

When the server needs an address to assign to a client, it processes the flags in the following order until it finds a usable address: client-request, reservation, extension, interface-identifier, and random. When the server needs to delegate a prefix to a client, it processes the flags in the following order until it finds a usable prefix: client-request, reservation, extension, and best-fit.

- ***restrict-to-reservations***—Controls whether the prefix is restricted to client (or lease) reservations.
- ***max-leases***—Maximum number of nonreserved leases allowed on the prefix. When a new lease needs to be created, the server does so only if the limit is not exceeded. When the limit is exceeded, the server cannot create or offer new leases to clients. If you also enable SNMP traps, the *max-leases* value also calculates the percentage of used and available addresses.



Tip Set the *max-leases* value to the expected maximum so that the SNMP address traps can return meaningful results.

- ***ignore-declines***—Controls whether the server responds to a DHCPv6 DECLINE message that refers to an IPv6 address or a delegated prefix from this prefix. If enabled, the server ignores all declines for leases in this prefix. If disabled (the preset value) or unset, the server sets to UNAVAILABLE every address or delegated prefix requested in a DECLINE message if it is leased to the client.
- ***expiration-time***—Time and date at which a prefix expires. After this date and time, the server neither grants new leases nor renews existing leases from this prefix. Enter a value in the format "[*weekday*] *month day hh:mm[:ss] year*"; for example, "**Dec 31 23:59 2006**". See the explanation for *expiration-time* attribute under [Creating and Editing Prefix Templates, on page 141](#).
- ***free-address-config***—Identifies which trap captures unexpected free address events on this prefix. If not configured, the server looks for the *free-address-config* attribute value for the parent link. If that attribute is not configured, the server looks at its *v6-default-free-address-config* attribute.
- ***deactivated***—Controls whether a prefix extends leases to clients. A deactivated prefix does not extend leases to any clients and treats all addresses in its ranges as if they were individually deactivated. The preset value is false (activated).
- ***max-pd-balancing-length***—Controls the maximum prefix-delegation prefix length that the failover pool balancing will consider in balancing a prefix-delegation prefix. The default value is 64 and it should never be longer than the longest prefix length allowed for the prefix delegation.
- ***allocation-group***—Allocation group to which this prefix belongs.
- ***allocation-group-priority***—Priority of this prefix over other prefixes in the same allocation group. The default value is zero.
- ***embedded-policy***—Policy embedded in the prefix.

Local Advanced and Regional Web UI

Step 1 From the **Design** menu, choose **Prefixes** under the **DHCPv6** submenu. The List/Add DHCP v6 Prefixes page shows the existing prefixes.

To create the prefix:

1. If creating it in other than the current VPN, choose a VPN from the username drop-down list on the top right of the window at the top right of the window.
2. Click the **Add Prefixes** icon in the Prefixes pane, enter a prefix name and address, and choose a prefix length from the drop-down list.

3. If you want a range of addresses for the prefix, enter the subnet address and choose a prefix length.
4. Choose a DHCP type (see the attribute descriptions at the top of this section). The default is DHCP.
5. If you want to apply a preconfigured prefix template, choose it from the drop-down list. (Note that the attribute values of an applied template overwrite the ones set for the prefix.)
6. Click **Add IPv6 Prefix**, which should add the prefix to the list.
7. Reload the DHCP server. When you return to the List/Add DHCPv6 Prefixes page, a message indicates how many prefixes are synchronized.

Step 2 To create a reverse zone from the prefix, click the Reverse Zone tab. On this tab, you can select a zone template, and click **Report**, then **Run**.

Step 3 Once you create a prefix, you can view and manage the leases for the prefix by clicking the Leases tab. On the Leases tab, you can view the leases for the client lookup key and manage each lease separately by clicking its name.

Step 4 You can view and manage the reservations for the prefix by clicking the Reservations tab. Add each reservation IP address and lookup key and whether the lookup key is a string or binary, then click **Add Reservation**.

Step 5 To edit a prefix, click its name on the Prefixes pane. On the Edit Prefix page, edit the prefix attributes, assign prefix to a group and set priorities, or create a new or edit an existing embedded policy.

To assign the prefix to a group and set priorities:

1. Enter the name of the group in the allocation-group attribute field.
2. Enter the priority value in the allocation-group-priority attribute field. If you do not enter any value here, it will be allotted the default value (0) and this prefix will have the lowest priority in the group.

You can find these attributes under Allocation Group in Advanced mode (see [Prefix Allocation Groups, on page 130](#)).

To manage an embedded policy:

1. Click **Create New Embedded Policy** or **Edit Existing Embedded Policy** to open the Edit DHCP Embedded Policy for Prefix page.
2. Modify the embedded policy properties (see [DHCPv6 Policy Hierarchy, on page 165](#)).
3. Click **Modify Embedded Policy**. The next time the Edit DHCPv6 Prefix page appears, you can edit the embedded policy for the prefix.
4. Click **Save**.

Step 6 In the regional web UI, you can push prefixes to local clusters and reclaim prefixes on the List/Add DHCPv6 Prefixes page:

- To push the prefix, click **Push** to open the DHCPv6 Push Prefix page. Choose the cluster or prefix template to which you want to push the prefix, then click **Push Prefix**. When the prefix is pushed, the reservations on the prefix is pushed with the prefix. Also, if the prefix is on a link, the parent prefix is pushed if it is not already present on the local cluster.
- To reclaim the prefix, click **Reclaim** to open the DHCPv6 Reclaim Prefix page. Choose the cluster or prefix template to which you want to reclaim the prefix, then click **Reclaim Prefix**. When the prefix is reclaimed, the reservations are deleted with the prefix, if there are no active leases, or if the force option is specified. Otherwise the prefix is deactivated.

Note If the prefix is on a universal link, it can be pushed to more than one cluster and that local changes will not take effect until the next server reload.

CLI Commands

Use **prefix name create ipv6address/length**. (The **prefix** command is a synonym for the **dhcp-prefix** command from previous releases.) Reload the DHCP server. For example:

```
nrcmd> prefix example-prefix create 2001:0db8::/32 [attribute=value]
nrcmd> dhcp reload
```

To apply a prefix template during prefix creation, use **prefix name create ipv6address/length template=name**. To apply a template to an existing prefix definition, use **prefix name applyTemplate template-name**. For example:

```
nrcmd> prefix example-prefix create 2001:0db8::/64 template=preftemp-1
nrcmd> prefix example-prefix applyTemplate template=preftemp-1
nrcmd> dhcp reload
```

You can set and enable the aforementioned attributes in the usual way. Add reservations by using **prefix name addReservation ipv6address/length lookup-key [-blob | -string]**. List leases by using **prefix name listLeases**.



Tip See the [Reconfiguring IPv6 Leases, on page 211](#) for additional syntax.

You can get an exact count of the total prefixes and links for the DHCP server by using **dhcp getPrefixCount [vpn name | all]**. You can specify a VPN or all VPNs. Omitting the **vpn name** returns a count for the current VPN.

Creating and Editing Links

You can create links directly. The attributes you can set for the link are:

- **name**—User-assigned name for the link.
- **vpn-id**—VPN that contains the link.
- **description**—Descriptive text for the link.
- **policy**—Shared policy used when replying to clients.
- **owner**—Owner of the link.
- **region**—Region for this link.
- **free-address-config**—Identifies which trap captures unexpected free address events on this prefix. If not configured, the server looks at its *v6-default-free-address-config* attribute.
- **interface**—Router interface associated with this link.
- **type**—Type of link (topological, location-independent, universal).
- **group-name**—Link group to which the link belongs.

Local Advanced and Regional Web UI

-
- Step 1** From the **Design** menu, choose **Links** under the **DHCPv6** submenu. The List/Add DHCP v6 Links page displays the existing links.
- Step 2** To add a link, click the **Add Link** icon in the Links pane.
- Step 3** Enter the desired name for the link.
- Step 4** If the link is for Prefix Stability, select the link type (*type*) and specify a link group name (*group-name*). The Link Type is *topological*, by default. You can also find these attributes in the Prefix Stability area in the Edit DHCP v6 Link Template page (see [Prefix Stability, on page 128](#) for details on link types and link groups).
- Note** You can have only one location independent link in a link group and one universal link in a VPN address space. Also, you cannot assign a link of type universal to a link group.
- Step 5** Click **Add Link**.
- Step 6** In the Edit Link page of the new link, choose the predefined prefixes for the link by moving them from the Available field to the Selected field.
- Step 7** To add new prefixes for the link, enter each prefix name and its address at the bottom of the page, indicate a range, choose the DHCP type and template (if needed), then click **Apply Prefix** for each one.
- Step 8** Click **Save**.
- Step 9** In the regional web UI, you can push links to local clusters and reclaim links on the Edit DHCP v6 Link page and pull replica IPv6 address space on the List/Add DHCP v6 Links page:
- To push the link, click **Push** (at the top of the page) to open the Push DHCP v6 Link page. Choose the cluster or link template to which you want to push the link, then click **Push Link**. When the link is pushed, all prefixes on the link, and all reservations on the prefixes are also be pushed.
 - To reclaim the prefix, click **Reclaim** (at the top of the page) to open the Reclaim DHCP v6 Link page. Choose the cluster or link template to which you want to reclaim the link, then click **Reclaim Link**. When the link is reclaimed, the reservations, prefixes, and link is deleted from the local cluster, provided there are no active leases. If active leases are found, prefixes are deactivated instead. The force option lets you remove the link and its prefixes when there are active leases.
- Note** Only universal links can be pushed to more than one cluster.
- To pull replica IPv6 address space, click the **Pull Data** icon (at the top of the links pane on the left) to open Select Pull Replica IPv6 Address Space. Choose the data synchronization mode (update, complete, or exact) and click **Report**.

The local changes will not take effect until the next server reload.

CLI Commands

Use **link name create**. (The **link** command is a synonym for the **dhcp-link** command from previous releases.) For example:

```
nrcmd> link example-link create [attribute=value]
```

To apply a link template during link creation, use **link name create template=name [template-root-prefix=address]**, with the *template-root-prefix* specified if the template could create more

than one prefix. To apply a template to an existing link definition, use **link name applyTemplate** *template-name* [*template-root-prefix*].

You can set and enable the aforementioned attributes in the usual way, and you can show and list links. To list prefixes or prefix names associated with a link, use **link name listPrefixes** or **link name listPrefixNames**.

Managing DHCP Networks

When you create a scope, you also create a network based on its subnet and mask. Scopes can share the same subnet, so that it is often convenient to show their associated networks and the scopes. Managing these networks is a local cluster function only. You can also edit the name of any created network.

Related Topics

[Listing Networks, on page 136](#)

[Editing Networks, on page 136](#)

Listing Networks

The List Networks page lets you list the networks created by scopes and determine to which scopes the networks relate. The networks are listed by name, which the web UI creates from the subnet and mask. On this page, you can expand and collapse the networks to show or hide their associated scopes.

In Basic mode, from the **Design** menu, choose **Networks** from the **DHCPv4** to open the DHCP Network Tree page. On this page, you can:

- **List the networks**—The networks appear alphabetically by name. You can identify their subnet and any assigned selection tags. Click the plus (+) sign next to a network to view the associated scopes.

To expand all network views, click **Expand All**. To collapse all network views to show just the network names, click **Collapse All**.

- **Edit a Network Name**—Click the network name. See [Editing Networks, on page 136](#).

To view the networks in the DHCPv6 address space, choose **Networks** from the **Design > DHCPv6** menu, to open the DHCPv6 Network Tree page. On this page you can add DHCPv6 links using a template and a template root prefix, as you would on the List/Add DHCPv6 Links page. Adding a link opens the Add DHCPv6 Link page. After creating the link, you can select it on the View DHCPv6 Networks page for editing.



Tip

You can use the DHCP v6 Network Tree page to push and reclaim links. Click the **Push** or **Reclaim** icon for the desired link. (See in the [Creating and Editing Links, on page 134](#) section for details)

Editing Networks

You can edit a network name. The original name is based on the subnet and mask as specified in the scope. You can change this name to an arbitrary but descriptive string.

Local Basic or Advanced Web UI

- Step 1** From the **Design** menu, choose **Networks** from the **DHCPv4** submenu or **Networks** from the **DHCPv6** submenu to open the DHCP Network Tree page (DHCP v4) or the DHCP v6 Network Tree page (DHCP v6).
For DHCPv6, the DHCP v6 Networks page is for creating networks. Enter a name for the network, choose a template, if desired, and enter the template root prefix name and click **Add Link** (see [Listing Networks, on page 136](#)).
If you want to edit a network, click the name of the network you want to edit. This opens the Edit DHCP v6 Link page.
- Step 2** Click **Save**.
-



CHAPTER 6

Managing Scopes, Prefixes, and Link Templates

This chapter describes how to set up templates for scopes, prefixes, and links.

- [Creating and Applying Scope Templates, on page 139](#)
- [Creating and Editing Prefix Templates, on page 141](#)
- [Creating and Editing Link Templates, on page 144](#)
- [Using Expressions in Templates , on page 146](#)

Creating and Applying Scope Templates

Scope templates apply certain common attributes to multiple scopes. These common attributes include a scope name based on an expression, policies, address ranges, and an embedded policy option based on an expression (see [Using Expressions in Scope Templates, on page 146](#)).

Local Advanced and Regional Web UI

Scope templates you add or pull from the local clusters are visible on the List DHCP Scope Templates page. To get there, from the **Design** menu, choose **Scope Templates** from **DHCPv4** submenu. This functionality is available only to administrators assigned the dhcp-management subrole of the regional central-cfg-admin or local ccm-admin role.

To explicitly create a scope template, click **Add Scope Templates** on the Scope Templates pane. This opens the Add DHCP Scope Template dialog box, which includes the template name. You can also choose an existing policy for the scope template. The other fields require expression values (see *Create a Scope Template* section in the *Cisco Prime Network Registrar 9.0 Administrator Guide* that describes these fields).

Related Topics

- [Using Expressions in Scope Templates, on page 146](#)
- [Additional Scope Template Attributes, on page 140](#)
- [Editing Scope Templates, on page 140](#)
- [Applying Scope Templates to Scopes, on page 140](#)
- [Cloning a Scope Template, on page 141](#)

CLI Commands

Create a scope template using **scope-template name create** [*attribute=value ...*]. For example:

```
nrcmd> scope-template example-scope-template create
```

You can also associate a policy with the scope template:

```
nrcmd> scope-template example-scope-template set policy=examplepolicy
```

Additional Scope Template Attributes

The optional additional attributes appear in functional categories. For a description of each attribute, click the attribute name to open a help window. For example, you might want to enable dynamic DNS updates for the scope, or set the main and backup DHCP failover servers.

After you complete these fields, click **Add Scope Template**.

Editing Scope Templates

To edit a scope template, select its name from the Scope Templates pane. The Edit DHCP Scope Template page is essentially the same as the Add DHCP Scope Template page (see [Creating and Applying Scope Templates, on page 139](#)) except for an additional attribute unset function. Make your changes, then click **Save**.

In the CLI, edit a scope template attribute by using **scope-template name set attribute**. For example:

```
nrcmd> scope-template example-scope-template set policy=default
```

Applying Scope Templates to Scopes

You can apply a scope template to a scope in a few ways.



Caution

Be careful applying a scope template to an existing scope. The template overwrites all the scope attributes with its own, which can have a detrimental effect if the scope is active.

Local Advanced Web UI

- **When a template is applied to a target**—If the scope-template has an embedded policy, it is copied to the scope. This embedded policy may or may not have options. As the entire scope-template's embedded policy is used (if it exists), it will wipe out any existing options in the scope. If the scope-template has no embedded policy, the scope's embedded policy is retained. Next the scope-template's option expression, if any, is evaluated and the options are added to the embedded policy options in the scope (if no embedded policy exists, one is created).
- **While creating a scope, derive its name from the template**—If you set a Scope Name Expression for the scope template (see [Using Expressions in Scope Templates, on page 146](#)) on the List/Add DHCP Scope Template page, when you add a scope on the List/Add DHCP Scopes page, omit the name of the scope, but add its subnet and mask, then choose the scope template from the Template drop-down list. Clicking **Add DHCP Scope** creates a scope with a name synthesized from the scope name expression.

If you do not set a scope name expression in the template and apply it to the scope without specifying a name for the scope, you get an error. (Note that Basic mode does not provide this functionality.)

- **After creating a named scope**—On the Edit DHCP Scopes page, scroll to the bottom to find the **Apply Template** button. Choose a preconfigured template from the drop-down list, then click the button. Then click **Save**. (Be aware of the previous warning that the template attributes overwrite the existing ones of the scope.)

CLI Commands

To apply a template to the scope while creating the scope, use **scope name create address mask [template=template-name] [attribute=value ...]**. For example:

```
nrcmd> scope example-scope create 192.168.50.0 24 template=example-scope-template
```

To derive the scope name from the template during scope creation, use **scope-template name apply-to { all | scope1 , scope2 ,...}**. For example:

```
nrcmd> scope-template example-scope-template apply-to examplescope-1,examplescope-2
```

Cloning a Scope Template

In the CLI, you can also clone a scope template from an existing one by using **scope-template clone-name create clone=template**, and then make adjustments to the clone. For example:

```
nrcmd> scope-template cloned-template create clone=example-scope-template-1 ping-timeout=200
```

Creating and Editing Prefix Templates

You can create prefixes from predefined templates. The attributes you can set for a prefix template are the following (for the expression syntax, see [Using Expressions in Prefix Templates, on page 151](#)):

- **name**—User-assigned name for the prefix template.
- **description**—Descriptive text for the prefix template.
- **dhcp-type**—Defines how DHCP manages address assignment for a prefix:
 - **dhcp** (preset value)—Uses the prefix for stateful address assignment.
 - **stateless**—Uses the prefix for stateless option configuration.
 - **prefix-delegation**—Uses the prefix for prefix delegation.
 - **infrastructure**—Uses the prefix to map a client address to a link, when the prefix does not have an address pool.
- **policy**—Shared policy to use when replying to clients.
- **owner** —Owner of this prefix, referenced by name.
- **region** —Region for this prefix, referenced by name.
- **prefix-name-expr**—Expression that evaluates to a string value to use for the name of the prefix created. For example, you can have the prefix name prepended by **CM-** if you define *prefix-name-expr* as (**concat "CM-" prefix**). In the CLI, you would include the expression in a file and point to that file:

```
> type prefix-name.txt
```

```
(concat "CM-" prefix)
```

```
nrcmd> prefix-template ex-template create prefix-name-expr=@prefix-name.txt
```

- **prefix-description-expr**—Expression that evaluates to a string value to apply to the description on the prefix created when using the template.
- **range-expr**—Expression that evaluates to an IPv6 prefix value to create an address range. In the CLI, you must use a file reference. For example:

```
> type subprefix-expr.txt
(create-prefix-range 1 0x1)
```

```
nrcmd> prefix-template ex-template set range-expr=@subprefix-expr.txt
```

- **options-expr**—Expression that evaluates to embedded policy options to create. (Use the **list** function to create multiple options.)
- **allocation-algorithms**—One or more algorithms the server uses to select a new address or prefix to lease to a client. The available algorithms are:
 - **client-request** (preset to off)—Controls whether the server uses a client-requested lease.
 - **reservation** (preset to on)—Controls whether the server uses an available reservation for the client.
 - **extension** (preset to on)—Controls whether the server calls extensions attached at the **generate-lease** extension point to generate an address or prefix for the client. When you use generate-lease extension point with DHCPv6 failover, the server uses the address or delegated prefix that the extension returns and does not perform a hash on this address or prefix as it does with the randomly generated addresses. If the extension is using some algorithmic method to generate the address or delegated prefix then the extension must be failover aware (extension will be able to determine if failover configuration is enabled and the role of the failover server). For details on extensions, see [Using Extensions, on page 351](#).
 - **interface-identifier** (preset to off)—Controls whether the server uses the interface-identifier from the client (link-local) address to generate an address; ignored for temporary addresses and prefix delegation.
 - **random** (preset to on)—Controls whether the server generates an address using an RFC 3041 algorithm; ignored for prefix delegation.
 - **best-fit** (preset to on)—Controls whether the server delegates the first, best-fit available prefix; ignored for addresses.

When the server needs an address to assign to a client, it processes the flags in the following order until it finds a usable address: client-request, reservation, extension, interface-identifier, and random. When the server needs to delegate a prefix to a client, it processes the flags in the following order until it finds a usable prefix: client-request, reservation, extension, and best-fit.

- **restrict-to-reservations** —Controls whether the prefix is restricted to client (or lease) reservations.
- **max-leases**—Maximum number of nonreserved leases allowed on the prefix. When a new lease needs to be created, the server does so only if the limit is not exceeded. When the limit is exceeded, the server cannot create or offer new leases to clients. If you also enable SNMP traps, the *max-leases* value also calculates the percentage of used and available addresses.



Note Be sure to set the *max-leases* value to the expected maximum so that the SNMP address traps can return meaningful results.

- **ignore-declines**—Controls whether the server responds to a DHCPv6 DECLINE message that refers to an IPv6 address or a delegated prefix from this prefix. If enabled, the server ignores all declines for leases in this prefix. If disabled (the preset value) or unset, the server sets to UNAVAILABLE every address or delegated prefix requested in a DECLINE message if it is leased to the client.
- **deactivated**—Controls whether a prefix extends leases to clients. A deactivated prefix does not extend leases to any clients and treats all addresses in its ranges as if they were individually deactivated. The preset value is false (activated).
- **expiration-time**—Time and date at which a prefix expires. After this date and time, the server neither grants new leases nor renews existing leases from this prefix. Enter a value in the format "[*weekday*] *month day hh :mm [:ss] year*"; for example, "**Dec 31 23:59 2006**". The reason for an expiration time is to support network renumbering events. The general idea is a new prefix is added and the old is taken away sometime at or after the *expiration-time*. Clients will be given leases on both prefixes. The server will automatically stop giving new clients leases once the configured valid lifetime before the *expiration-time* is reached. At this time, new clients will not get a lease on the prefix. Existing clients will continue to be able to use an existing lease, but will get shorter and shorter lifetimes (preferred and valid). The delta between the preferred and valid is always maintained. Thus if the preferred is 1 day and the valid 2 days, new clients will stop getting leases 2 days before the *expiration-time*, existing clients will continue to be able to renew leases with preferred lifetimes lesser than 1 day and valid lifetimes greater than 2 days. 1 day before the *expiration-time*, clients will get a 0 preferred lifetime.
- **free-address-config**—Trap that captures unexpected free address events on the prefix.
- **reverse-zone-prefix-length**—Prefix length of the reverse zone for ip6.arpa updates. (See [Determining Reverse Zones for DNS Updates, on page 248](#) for details.)
- **max-pd-balancing-length**—Controls the maximum prefix-delegation prefix length that the failover pool balancing will consider in balancing a prefix-delegation prefix. The default value is 64 and it should never be longer than the longest prefix length allowed for the prefix delegation.
- **selection-tags**—List of selection tags associated with the prefix.
- **allocation-group**—Allocation group to which the prefix belongs.
- **allocation-group-priority**—Priority of the prefix over other prefixes in the same allocation group. The default value is zero.

Local Advanced and Regional Web UI

-
- Step 1** From the **Design** menu, choose **Prefix Templates** under the **DHCPv6** submenu. The List/Add DHCP v6 Prefix Templates page shows the existing templates.
- Step 2** Click the **Add Prefix Templates** icon in the **Prefix Templates** pane to open the Add Prefix Template dialog box.
- Step 3** Enter the prefix template name and click **Add Prefix Template**.
- Step 4** To edit a prefix template, select its name on the Prefix Templates pane. Set the attributes and add expressions for the templates that require expressions (see [Using Expressions in Prefix Templates, on page 151](#)).
- Step 5** On the Edit DHCP v6 Prefix Template page, edit the template attributes, such as adding a selection tag, assigning a group and setting priorities, then click **Save**.
- Step 6** In the regional web UI, you can pull replica prefix templates or push templates to local clusters:
- Click **Pull Data** to open the Select Replica Prefix Template Data to Pull page. Choose a pull mode for the cluster (ensure, replace, or exact), then click **Pull All Prefix Templates**. On the Report Pull DHCPv6 Prefix Template page, click **OK**.

- Click **Push** for a specific template (or **Push All**) to open the Push Data to Local Clusters page. Choose a data synchronization mode (ensure, replace, or exact), move the desired cluster or clusters to the Selected table, then click **Push Data to Clusters**.

CLI Commands

To create the prefix template, use **prefix-template name create** [*attribute=value ...*]. For example:

```
nrcmd> prefix-template example-prefix-template create [attribute=value]
```

You can set and enable the aforementioned attributes in the usual way, and you can show and list prefix templates. In addition:

- To clone a prefix template, use **prefix-template name create clone=name**.
- To apply a template to one or more prefixes, use **prefix-template name apply-to** {**all** | *prefix* [*,prefix*,...]}.
- The prefix-template includes an embedded-policy object. The prefix-template-policy CLI command and the Web UI supports the embedded policy on the prefix-template page.

Creating and Editing Link Templates

You can create links from predefined templates. The attributes you can set for a link template are as follows (for the expression syntax, see [Using Expressions in Link Templates, on page 155](#)):

- **name** —User-assigned name for the link template.
- **description** —Description of the link template itself.
- **policy** —Shared policy used when replying to clients, as applied to the link.
- **owner** —Owner of the link.
- **region** —Region for this link.
- **link-name-expr** —Expression to define the name of the link once the template is applied.
- **link-description-expr** —Expression to define the description on the link once applied.
- **prefix-expr** —Expression to create the list of associated prefixes once the template is applied. For example, you can specify creating prefixes based on defining *prefix-expr* as **@link-prefix-expr.txt** to point to the file that contains this expression (and assuming that the cm-prefix, cpe-address-prefix, and cpe-pd-prefix templates exist):

```
(list
 (create-prefix "cm-prefix" (create-prefix-range 32 0x1))
 (create-prefix "cpe-address-prefix" (create-prefix-range 32 0x2))
 (create-prefix "cpe-pd-prefix" (create-prefix-range 16 0x1))
 )
```

- **options-expr** —Expression to define the list of embedded policy options to create with the link.
- **free-address-config** —Trap that captures unexpected free address events on this link
- **type** —Type of the link (topological, location-independent, universal).
- **group-name** —Link group to which the link belongs.

Local Advanced and Regional Web UI

-
- Step 1** From the **Design** menu, choose **Link Templates** under the DHCPv6 submenu. The List/Add DHCP v6 Link Templates page appears. The page displays the existing templates.
- Step 2** Click the **Add Link Templates** icon in the **Link Templates** pane to open the Add Link Template dialog box.
- Step 3** Enter a link template name and click **Add Link Template**.
- Step 4** Enter an optional description, and optionally choose a preconfigured policy from the drop-down list.
- Step 5** Add expressions for the *link-name-expr*, *link-description-expr*, *prefix-expr*, or *options-expr* field attributes (see [Using Expressions in Link Templates, on page 155](#)).
- Step 6** If the link template is for Prefix Stability, select the link type (type) and specify a link group name (group-name). You can find these attributes in the Prefix Stability block in the Add DHCP v6 Link Template page (see [Prefix Stability, on page 128](#) for details on link types and link groups).
- Step 7** Click **Save**.
- Step 8** In the regional web UI, you can pull replica link templates or push templates to local clusters:
- Click **Pull Data** to open the Select Replica Link Template Data to Pull page. Choose a pull mode for the cluster (ensure, replace, or exact), then click **Pull All Link Templates**. On the Report Pull DHCPv6 Link Template page, click **OK**.
 - Click **Push** for a specific template (or **Push All**) to open the Push Data to Local Clusters page. Choose a data synchronization mode (ensure, replace, or exact), move the desired cluster or clusters to the Selected table, then click **Push Data to Clusters**.
-

CLI Commands

To create the link template, use **link-template name create** [*attribute=value ...*]. For example:

```
nrcmd> link-template example-link-template create [attribute=value]
```

You can set and enable the aforementioned expression setting attributes in the usual way, and you can show and list link templates. For example, to set a prefix expression for the link template, use the following file definition and pointer to the file (and assuming that the cm-prefix, cpe-address-prefix, and cpe-pd-prefix templates exist):

```
> type link-prefix-expr.txt
(list (create-prefix "cm-prefix" (create-prefix-range 32 0x1))
 (create-prefix "cpe-address-prefix" (create-prefix-range 32 0x2))
 (create-prefix "cpe-pd-prefix" (create-prefix-range 16 0x1) )
nrcmd> link-template example-link-template set prefix-expr=@link-prefix-expr.txt
```

In addition:

- To clone a link template, use **link-template name create clone=name**.
- To apply a template to one or more links, use **link-template name apply-to** {**all** | *link* [*,link,...*]}. You can create prefixes by using **link-template name apply-to link** [*prefix*], but only with one link specified.

- The link-template includes an embedded-policy object. The link-template-policy CLI command and the Web UI supports the embedded policy on the link-template page.

Using Expressions in Templates

Using Expressions in Scope Templates

You can specify expressions in a scope template to dynamically create scope names, IP address ranges, and embedded options when creating a scope. Expressions can include context variables and operations.



Note

Expressions are not the same as DHCP extensions. Expressions are commonly used to create client identities or look up clients. Extensions (see [Using Extension Points, on page 351](#)) are used to modify request or response packets. If you apply the template to a scope that already has ranges defined, the address range expression of the scope template is not evaluated for that scope.

The following table lists the scope expression functions. Note that these functions are not case-sensitive.

Table 17: Scope Template Expression Functions

Expression Function	Description
Context Variables	
bcast-addr	Derived from the broadcast address in the subnet, such as 192.168.50.255. Use in any expression field.
first-addr	Derived from the first address in the subnet, such as the first address in 192.168.50.64/26 is 192.168.50.65. Use in any expression field.
last-addr	Derived from the last address in the subnet, such as the last address in 192.168.50.64/26 is 192.168.50.127. Use in any expression field.
mask-addr	Derived from the network mask address in the subnet, such as 255.255.255.0. Use in any expression field.
mask-count	Derived from the number of bits in the network address of the subnet, such as 24. Use in the Scope Name Expression or Embedded Policy Option Expression field.
naddrs	Derived from the number of IP addresses in the subnet, such as 255. Use in the Scope Name Expression field.
nhosts	Derived number of usable hosts in the subnet, such as 254. Use in any expression field.

Expression Function	Description
subnet	Derived from the IP address and mask of the subnet, such as 192.168.50.0/24. Use in the Scope Name Expression or Embedded Policy Option Expression field.
subnet-addr	Derived from the subnet address, such as 192.168.50.0. Use in any expression field.
template.attribute	Attribute of the scope template, such as <code>template.ping-timeout</code> . Use in the Embedded Policy Option Expression field. Note The attribute must be explicitly set. Otherwise, the expression will fail to evaluate.
this.attribute	Attribute of the scope. Note The attribute must be explicitly set. Otherwise, the expression will fail to evaluate.
Arithmetic Operations (unsigned integer arguments only)	
(+ arg1 arg2)	Adds the two argument values, such as (+ 2 3).
(- arg1 arg2)	Subtracts the second argument value from the first one, such as with <code>ping-timeout</code> defined as 100, (<code>- template.ping-timeout 10</code>) yields 90.
(* arg1 arg2)	Multiplies the values of two arguments.
(/ arg1 arg2)	Divides the value of the first argument by that of the second one (which cannot be zero).
Concatenation Operation	
(concat arg1 ... argn)	Concatenates the arguments into a string, to be used in the Scope Name Expression field. Examples: With <code>subnet=192.168.50.0/24</code> and <code>template.ping-timeout=100</code> : <pre>(concat "ISP-" subnet) --> ISP-192.168.50.0/24 (concat subnet "-" (+ template.ping-timeout 10)) --> 192.168.50.0/24-110 (concat "ISP-" subnet "-" (+ template.ping-timeout 10)) --> ISP-192.168.50.0/24-110</pre> <p>See also Scope Name Expression Example, on page 150.</p>

Expression Function	Description
Create Option Operation	
(create-option <i>opt val</i>)	<p>Use create-option in the Embedded Policy Option Expression field to create new DHCP options for the scope. The first argument can be an integer or string to represent the option number or name. The second argument can be a string or blob to give the option a value.</p> <p>You can also specify custom defined and unknown options. For undefined options, the option number must be specified and the data is used as is (as blob data). If the data is a string, the string is used as is and if the data is a number or address, it is used as is.</p> <p>Examples:</p> <pre>(list (create-option "domain-name" "example.com") (create-option 3 "10.10.10.1")) (create-option "routers" "10.10.10.1,10.10.10.2,10.10.10.3") (create-option "routers" (create-ipaddr subnet 10))</pre> <p>See also Embedded Policy Option Expression Example, on page 151.</p>
Create Vendor Option Operation	
(create-vendor-option <i>set-name opt val</i>)	<p>Use the create-vendor-option in the Embedded Policy Option Expression field to creates a DHCP vendor option. The set-name specifies the option definition set for the vendor option. The opt can be the literal string or integer identifying the vendor option in the set. The val is representation of the option value.</p> <p>For example:</p> <pre>(list (create-option "routers" (create-ipaddr subnet 1)) (create-vendor-option "dhcp-cablelabs-config" 125 (concat "(tftp-servers 2 " (create-ipaddr subnet 2) ")"))</pre>
Create Range Operation	

Expression Function	Description
<code>(create-range start end)</code>	<p>Use this operation in the Range Expression field. It creates an IP address range for the scope. The first argument is the start of the address range and can be an integer or IP address string. The second argument is the end of the range and can be an integer or IP address string. Do not include the local host or broadcast address determined by the mask (such as 0 and 255 for /24 subnets) in the range. Validation ensures that the range must be in the subnet defined by the template and that the first argument value must be lower than the second. An integer value determines the position of the address in the given subnet. Examples (with subnet=192.168.50.0/26):</p> <pre>(create-range "192.168.50.65" "192.168.50.74") --> 192.168.50.65 - 192.168.50.74 (create-range 1 10) --> 192.168.50.65 - 192.168.50.74</pre> <p>See also Range Expression Example, on page 150.</p>
Create IP Operation	
<code>(create-ipaddr net host)</code>	<p>Use this operation in the Embedded Policy Option Expression or Range Expression fields. It creates an IP address string. The net argument is a string or variable. The host argument is an integer. Example:</p> <pre>(create-ipaddr subnet 4)</pre>
List Operation	
<code>(list oper1 ... opern)</code>	<p>Arguments must all be create-option or create-range operations. Nesting is not supported. Examples:</p> <pre>(list (create-option "routers" "10.10.10.1") (create-option "domain-name" "example.com")) (list (create-range 1 5) (create-range 10 20))</pre>

Local Advanced and Regional Web UI

There are three fields on the Add DHCP Scope Template page for which you must specify an expression:

- **Scope Name Expression**—Must return a string
- **Range Expression**—Must return IP addresses
- **Embedded Policy Option Expression**—No requirements

CLI Commands

Use the following **scope-template** command attributes:

- **scope-name**
- **ranges-exp**
- **options-exp**

Scope Name Expression Example

You might want to set an expression so that the template constructs scope names starting with “ISP-” and followed by the subnet of the scope and a derivative of its ping timeout value. You would use the following expression in the Scope Name Expression field:

```
(concat "ISP-" subnet "-" (+ template.ping-timeout 10))
```

The elements of the example expression are:

- **(concat ...)**—Concatenation operation, which concatenates all the following values into one value.
- **“ISP-”**—String with which to start the scope name.
- **subnet**—Keyword variable that indicates to use the existing subnet defined for the scope.
- **“-”**—Indicates to include this hyphen to construct the value.
- **(+ template.ping-timeout 10)**—Indicates to add the *ping-timeout* property value for the scope to the number 10.

If the scope subnet happens to be 192.168.50.0/24 and its *ping-timeout* value 100, the resulting constructed scope name would be:

```
ISP-192.168.50.0/24-110
```

Range Expression Example

You might want to set an expression so that the template constructs only certain address ranges for scopes. You can either be explicit about the actual starting and ending addresses, or you can make them relative to the subnet. Here are two ways of requesting relative ranges in the Range Expression field:

```
(create-range first-addr last-addr)
(create-range 1 10)
```

The first **create-range** operation creates the address range based on the first through last usable address in the subnet. For the 192.168.50.0/24 subnet, for example, the address range would be 192.168.50.1 through 192.168.50.254. Because the second operation specifies integers instead of full IP addresses, it makes the range relative to the subnet based on its mask. If the template discovers the subnet to be 192.168.50.0/26, it takes the first through tenth address in this subnet, which would be 192.168.50.65 through 192.168.50.74.

To set the range expressions in the CLI, you should place the expression into a file and use a command such as:

```
nrcmd> scope-template example-template set ranges-expr=@ file where file
is the name of the file that you created with the expressions.
```

Embedded Policy Option Expression Example

An embedded policy is important because the DHCP server looks at it before it looks at the assigned, named policy of the scope. This is usually where you would set the DHCP options on a scope. You might want to set an expression so that the template constructs DHCP options for the scope embedded policy. Here are some examples:

```
(create-option "domain-name" "example.com")
(create-option 3 "10.10.10.1")
(create-option "routers" (create-ipaddr subnet 10))
```

The first **create-option** operation associates the value `example.com` with the *domain-name* option for the scope. The second operation associates the address `10.10.10.1` with the *routers* option (number 3). The third operation creates an IP address for the *routers* option based on the tenth address in a subnet.

To set the policy options expressions in the CLI, you should place the expression into a file and use a command such as:

```
nrcmd> scope-template example-template set options-expr=@ file
```

where *file* is the name of the file that you created with the expressions.



Note Trying to specify the expression directly on the CLI command line will likely fail because of embedded spaces and special characters such as the quotes. Use the `@file` syntax as it avoids any potential issues with the CLI command parser. But the WebUI does not support the `@file` syntax. You can enter complex expressions directly in the Web UI.

Using Expressions in Prefix Templates

You can specify expressions in a prefix template to dynamically create prefix names, IP address ranges, and embedded options when creating a prefix. Expressions can include context variables and operations.



Note Expressions are not the same as DHCP extensions. Expressions are commonly used to create client identities or look up clients. Extensions (see [Using Extension Points, on page 351](#)) are used to modify request or response packets.

When a template is applied to a prefix, if the prefix-template has an embedded policy, it is copied to the prefix. This embedded policy may or may not have options. As the entire prefix-template's embedded policy is used (if it exists), it will wipe out any existing options in the prefix. If the prefix-template has no embedded policy, the prefix's embedded policy is retained. Next the prefix-template's option expression, if any, is evaluated and the options are added to the embedded policy options in the prefix (if no embedded policy exists, one is created).

The tables below lists the prefix template predefined variables and lists the operator. Note that these variables and operators are not case-sensitive.

Table 18: Prefix Template Expression Predefined Variables

Predefined Variable	Description
prefix	Network number and length, based on the template root prefix if applying a link template to a link, or the prefix address if applying a prefix template to a prefix.
vpn	VPN of the prefix.
prefix-addr	Address portion of the prefix.
prefix-length	Number of prefix address bits.
mask-length	Number of prefix mask bits.
template.attribute	Attribute of the prefix template. Note The attribute must be explicitly set. Otherwise, the expression will fail to evaluate.
this.attribute	Attribute of the prefix, such as this.link for the prefix's link name. Note The attribute must be explicitly set. Otherwise, the expression will fail to evaluate.

Table 19: Prefix Template Expression Operators

Expression Operator	Description
Arithmetic Operators (unsigned integer arguments only)	
(+ <i>arg1 arg2</i>)	Adds the two argument values, such as (+ 2 3).
(- <i>arg1 arg2</i>)	Subtracts the second argument value from the first one, such as with <i>ping-timeout</i> defined as 100, (- template.ping-timeout 10) yields 90.
(* <i>arg1 arg2</i>)	Multiplies the values of two arguments.
(/ <i>arg1 arg2</i>)	Divides the value of the first argument by that of the second one (which cannot be zero).
(% <i>arg1 arg2</i>)	Modulo arithmetic operator to determine the remainder of the result of the first argument divided by the second one.
Concatenation Operator	
(concat <i>arg1 ... argn</i>)	Concatenates the arguments into a string.
List Operator	

Expression Operator	Description
(list <i>oper1</i> ... <i>opern</i>)	Creates an options list or list of prefixes. Required if needing more than one option for a prefix. All arguments must be create-v6-option or create-prefix-range operations. Nesting is not supported.
Create IP Operator	
(create-prefix-addr <i>prefix-name interface-id</i>)	Creates an IPv6 address string based on the prefix name and interface ID (an IPv6 address that you can specify as a string), which is the lower 64-bit address in the prefix (which need not be contained in the parent prefix). Used in the <i>range-expr</i> and <i>options-expr</i> attributes.
Create Range Operator	
(create-prefix-range <i>size n</i>)	<p>Creates an address range (child) for the prefix, used in the <i>range-expr</i> attribute. The prefix value that the function is based on is either the <i>template-root-prefix</i> if applying a link template to a link, or the prefix address if applying a prefix template to a prefix.</p> <p>Range value—An increase in the prefix length.</p> <p>Size—The number of bits by which you can increase the prefix length. Must be a value from 1 through 32. Must be less than the parent prefix length.</p> <p>n—The nth occurrence of the child prefix. Value can be 0, but is limited to less than two to the power of the size. Must be less than or equal to the size.</p> <p>The size and n must be greater than zero. The <i>n</i> must be less than or equal to the <i>size</i>, and the <i>size</i> must be less than the parent prefix length. For example:</p> <p>(create-prefix-range 32 0x1)</p>
Create Option Operation	

Expression Operator	Description
<p>(create-option <i>opt val</i>)</p>	<p>Creates a DHCPv6 option, used in the options-expr attribute. The <i>opt</i> can be the literal string or integer identifying the option. The <i>val</i> is the string representation of the option value, as defined by the option TLV value.</p> <p>You can use custom defined and unknown options. For undefined options, the option number must be specified and the data is used as is (as blob data). If the data is a string, the string is used as is and if the data is a number or address, it is used as is.</p> <p>For example:</p> <pre>(list (create-option "dns-servers" (create-prefix-addr prefix "::2")) (create-option "domain-list" "sales.example.com,example.com"))</pre> <p>Note (create-v6-option <i>opt val</i>) is a synonym for (create-option) and can be used instead.</p>
<p>Create Vendor Option Operator</p>	

Expression Operator	Description
(create-vendor-option <i>set-name opt val</i>)	<p>Creates a DHCPv6 vendor option, used in the options-expr attribute. The set-name specifies the option definition set for the vendor option. The opt can be the literal string or integer identifying the vendor option in the set. The val is representation of the option value.</p> <p>For example:</p> <pre>(list (create-option "dns-servers" (create-prefix-addr prefix "::2")) set-name opt val (create-vendor-option "dhcp6-cablelabs-config" 17 "(enterprise-id 4491((tftp-servers 32 3800:0:0:180::6) (config-file-name 33 modem_ipv6.bin) (syslog-servers 34 3800:0:0:180::8) (rfc868-servers 37 3800:0:0:180::6) (time-offset 38 -5h) (cablelabs-client-configuration 2170 (primary-dhcp-server 1 10.38.1.5) (secondary-dhcp-server 2 10.38.1.6))))"))</pre> <p>Note (create-v6-vendor-option opt val) is a synonym for (create-vendor-option) and can be used instead.</p>



Note We recommend that you use create-option and create-vendor-option for v4 and v6.

Using Expressions in Link Templates

You can specify expressions in a link template to dynamically create prefix names, IP address ranges, and embedded options when creating a link. Expressions can include context variables and operations.



Note Expressions are not the same as DHCP extensions. Expressions are commonly used to create client identities or look up clients. Extensions (see [Using Extension Points, on page 351](#)) are used to modify request or response packets.

When a template is applied to a link, if the link-template has an embedded policy, it is copied to the link. This embedded policy may or may not have options. As the entire link-template's embedded policy is used (if it exists), it will wipe out any existing options in the link. If the link-template has no embedded policy, the link's embedded policy is retained. Next the link-template's option expression, if any, is evaluated and the options are added to the embedded policy options in the link (if no embedded policy exists, one is created).

The table below lists the link template predefined variables and [Table 21: Link Template Expression Operators](#) lists the link template operators. Note that these variables and operators are not case-sensitive. [Table 19: Prefix Template Expression Operators](#) lists the prefix template operators. The link template operators table and prefix template operations table both have same operators, except that only a link template can use **Create Prefix Operator** and prefix template can not use the operator.

Table 20: Link Template Expression Predefined Variables

Predefined Variable	Description
mask-length	Number of prefix mask bits (with a <i>template-root-prefix</i> defined).
prefix	Network number and length (with a <i>template-root-prefix</i> defined).
prefix-addr	Address portion of the prefix (with a <i>template-root-prefix</i> defined).
prefix-length	Number of prefix address bits (with a <i>template-root-prefix</i> defined).
template.attribute	Attribute of the link template. Note The attribute must be explicitly set. Otherwise, the expression will fail to evaluate.
this.attribute	Attribute of the link. Note The attribute must be explicitly set. Otherwise, the expression will fail to evaluate.
vpn	VPN of the link.

Table 21: Link Template Expression Operators

Expression Operator	Description
Arithmetic Operators (unsigned integer arguments only)	
(+ <i>arg1 arg2</i>)	Adds the two argument values, such as (+ 2 3).
(- <i>arg1 arg2</i>)	Subtracts the second argument value from the first one.
(* <i>arg1 arg2</i>)	Multiplies the values of two arguments.
(/ <i>arg1 arg2</i>)	Divides the value of the first argument by that of the second one (which cannot be zero).

Expression Operator	Description
(% <i>arg1 arg2</i>)	Modulo arithmetic operator to determine the remainder of the result of the first argument divided by the second one.
Concatenation Operator	
(concat <i>arg1 ... argn</i>)	Concatenates the arguments into a string.
List Operator	
(list <i>oper1 ... opern</i>)	Creates an options list or list of prefixes. Required if you need more than one option for a link or prefix, or more than one prefix for a link. All arguments must be create-v6-option operation. Nesting is not supported. For example: <pre>(list (create-prefix " cm-prefix" (create-prefix-range 32 0x1)) (create-prefix "cpe-address-prefix" (create-prefix-range 32 0x2)) (create-prefix "cpe-pd-prefix" (create-prefix-range 16 0x1)))</pre>
Create Prefix Operator	
(create-prefix <i>template prefix</i>)	Creates a prefix based on a predefined prefix template name and the prefix, including the link VPN (assuming that a <i>template-root-prefix</i> is defined). The <i>prefix</i> argument can be the prefix name, but also the create-prefix-addr or create-prefix-range operator value. You can use the list function to combine multiple operations. For example: <pre>(create-prefix "cm-prefix" (create-prefix-range 32 0x1))</pre>
Create IP Operator	
(create-prefix-addr <i>prefix interface-id</i>)	Creates an IPv6 address string (assuming that a <i>template-root-prefix</i> is defined) based on the prefix name and interface ID (an IPv6 address that you can specify as a string), which is the lower 64-bit address in the prefix (which need not be contained in the parent prefix). Used in the <i>prefix-expr</i> and <i>options-expr</i> attributes.
Create Range Operator	

Expression Operator	Description
(create-prefix-range <i>size n</i>)	<p>Creates an address range (child) for the prefix, used in the <i>prefix-expr</i> attribute. The prefix value that the function is based on is either the <i>template-root-prefix</i> if applying a link template to a link, or the prefix address, if applying a prefix template to a prefix.</p> <p>Range value—An increase in the prefix length.</p> <p>Size—The number of bits by which you can increase the prefix length. Must be a value from 1 through 32. Must be less than the parent prefix length.</p> <p><i>n</i>—The <i>n</i>th occurrence of the child prefix. Value can be 0, but is limited to less than two to the power of the size. Must be less than or equal to the size.</p> <p>The size and <i>n</i> must be greater than zero.</p> <p>The <i>n</i> must be less than or equal to the <i>size</i>, and the <i>size</i> must be less than the parent prefix length. For example:</p> <pre>(create-prefix-range 32 0x1)</pre>
Create Option Operator (create-option <i>opt val</i>)	<p>Creates a DHCPv6 option, used in the <i>options-expr</i> attribute. The <i>opt</i> can be the literal string or integer identifying the option. The <i>val</i> is the string representation of the option value, as defined by the option TLV value.</p> <p>You can use custom defined and unknown options. For undefined options, the option number must be specified and the data is used as is (as blob data). If the data is a string, the string is used as is and if the data is a number or address, it is used as is.</p> <p>For example:</p> <pre>(list (create-option "dns-servers" (create-prefix-addr prefix "::2")) (create-option "domain-list" "sales.example.com,example.com"))</pre> <p>Note (create-v6-option <i>opt val</i>) is a synonym for (create-option) and can be used instead; but we recommend that you use (create-option).</p>
Create Vendor Option Operation	

Expression Operator	Description
<p>(create-vendor-option <i>set-name opt val</i>)</p>	<p>Creates a DHCPv6 vendor option, used in the options-expr attribute. The set-name specifies the option definition set for the vendor option. The opt can be the literal string or integer identifying the vendor option in the set. The val is representation of the option value.</p> <p>For example:</p> <pre>(list (create-option "dns-servers" (create-prefix-addr prefix "::2")) (create-vendor-option "dhcp6-cablelabs-config" 17 "(enterprise-id 4491((tftp-servers 32 3800:0:0:180::6) (config-file-name 33 modem_ipv6.bin)(syslog-servers 34 3800:0:0:180::8) (rfc868-servers 37 3800:0:0:180::6)(time-offset 38 -5h) (cablelabs-client-configuration 2170 (primary-dhcp-server 1 10.38.1.5) (secondary-dhcp-server 2 10.38.1.6))))"))</pre> <p>Note (create-v6-vendor-option opt val) is a synonym for (create-vendor-option) and can be used instead; but we recommend that you use (create-vendor-option).</p>



CHAPTER 7

Managing Policies and Options

This chapter describes how to set up DHCP policies and options. Before clients can use DHCP for address assignment, you must add at least one DHCPv4 scope (dynamic address pool) or DHCPv6 prefix to the server. The policy attributes and options are assigned to the scope or prefix.

- [Configuring DHCP Policies, on page 161](#)
- [Configuring DHCPv6 Policies, on page 162](#)
- [Types of Policies, on page 163](#)
- [Policy Hierarchy, on page 164](#)
- [Creating and Applying DHCP Policies, on page 166](#)
- [Cloning a Policy, on page 168](#)
- [Setting DHCP Options and Attributes for Policies, on page 168](#)
- [Creating and Editing Embedded Policies, on page 170](#)
- [Creating DHCP Option Definition Sets and Option Definitions, on page 171](#)
- [Option Definition Set , on page 181](#)

Configuring DHCP Policies

Every DHCPv4 scope or DHCPv6 prefix must have one or more policies defined for it. Policies define lease duration, gateway routers, and other configuration parameters, in what are called DHCP options. Policies are especially useful if you have multiple scopes or prefixes, because you need only define a policy once.

This section describes how you can define named policies with specific attributes and option definitions, or use system default or embedded policies.

Related Topics

- [Types of Policies, on page 163](#)
- [DHCPv4 Policy Hierarchy, on page 164](#)
- [Creating and Applying DHCP Policies, on page 166](#)
- [Cloning a Policy, on page 168](#)
- [Setting DHCP Options and Attributes for Policies, on page 168](#)
- [Creating and Editing Embedded Policies, on page 170](#)

Configuring DHCPv6 Policies

You can edit DHCPv6 policy attributes, which are:

- **affinity-period**—See [Lease Affinity, on page 190](#) (no preset value).
- **allow-non-temporary-addresses**—Enable or disable DHCPv6 clients requesting nontemporary (IA_NA) addresses (preset value enable).
- **allow-rapid-commit**—With Rapid Commit enabled, clients receive information (when solicited) on committed addresses, which are then more quickly committed with a client request (preset value disable). Use Rapid Commit only if one DHCP server is servicing clients, otherwise it might seem like the client is receiving multiple addresses. (See [DHCPv6 Policy Hierarchy, on page 165](#) for special handling of this attribute, and Reconfigure support, when used in an embedded or named policy for a prefix.)
- **allow-temporary-addresses**—Enable or disable DHCPv6 clients requesting temporary (IA_IA) addresses (preset value enable).
- **default-prefix-length**—For prefix delegation, default prefix length of the delegated prefix if the client or router does not explicitly request it (or *allow-client-hints* is disabled); must always be less than or equal to the prefix range prefix length (preset value 64 bytes).
- **reconfigure**—Enables special handling during the policy hierarchy processing when checking the Prefix policies (embedded or named) for the Prefixes on a Link (see [Reconfiguring IPv6 Leases, on page 211](#)).
- **preferred-lifetime**—Default and maximum preferred lifetime for leases (preset value 1 week).
- **v6-reply-options**—DHCPv6 options returned in replies to clients (no preset value). (See [DHCPv6 Policy Hierarchy, on page 165](#) for special handling of this attribute when used in an embedded or named policy for a prefix.)
- **valid-lifetime**—Default and maximum valid lifetime for leases (preset value 2 weeks).



Tip For details on the Reconfigure attributes, see [Reconfiguring IPv6 Leases, on page 211](#).

Reconfigure Support (DHCPv6)

For DHCPv6, a server can send a RECONFIGURE message to a client to inform the client that the server has new or updated configuration parameters. If so authorized and through proper authentication, the client then immediately initiates a Renew, Rebind, or Information-request reply transaction with the server so that the client can retrieve the new data. Without this support, a client must wait until it renews its lease to get configuration updates.

You can have the server unicast the Reconfigure packet or deliver it through a relay agent. If you do not specify either way, the client's client-class policy, requested lease's prefix or link policies, or `system_default_policy` (but not the client policy) determines the preferred method. If the unicast method is not available (the client has no valid address lease), the server uses the relay agent; with no relay agent, the server tries to unicast; failing both results in an error. With the unicast method, if the specified lease is not usable, the server selects the lease with the longest valid lifetime.

The server and client negotiate Reconfigure support through the added security of a reconfigure key. The internal process is basically:

1. The client sends the server a REQUEST, SOLICIT, or ADVERTISE packet that includes the *reconfigure-accept* option (20) to indicate that the client wants to accept Reconfigure messages.

(Conversely, the DHCP server can send a *reconfigure-accept* option to the client about whether the client should accept Reconfigure messages.) This option is required for Reconfigure support.

2. If the Cisco Prime Network Registrar policy for the client has the *reconfigure* attribute set to **allow** or **require** (rather than **disallow**), the DHCP server accepts the packet and generates a reconfigure key for the client. (The server records the key value and its generation time in the *client-reconfigure-key* and *client-reconfigure-key-generation-time* attributes for the DHCPv6 lease.)
3. The server sends a Reply packet to the client with the reconfigure key in the *auth* option (11) along with the *reconfigure-accept* option.
4. The client records the reconfigure key to authenticate Reconfigure messages from the server.
5. When the server wants to reconfigure the client, it sends a Reconfigure packet with the *reconfigure-message* option (19) and an *auth* option containing a hash generated from the packet and the reconfigure key. The *reconfigure-message* option indicates in the *msg-type* field whether the client should respond with a Renew or an Information-request packet.
6. Upon receiving the packet, the client validates that the *auth* option contains the valid hash, then returns a Renew, Rebind, or Information-request packet. This packet includes an Option Request (*oro*) option (6) to indicate specific option updates. (If the server does not receive a reply from the client in a preconfigured timeout value of 2 seconds, the server retransmits the Reconfigure message at most 8 times, then aborts the reconfigure process for the client.)
7. The server sends the client a Reply packet that includes options for configuration parameters. The packet might also include options containing addresses and new values for other configuration parameters, even if the client did not request them. The client records these changes.

Types of Policies

There are three types of policies—system default, named, and embedded:

- **System default (*system_default_policy*)**—Provides a single location for setting default values on certain options for all scopes or prefixes. Use the system default policy to define attributes and standard DHCP options that have common values for all clients on all the networks that the DHCP server supports. You can modify the system default options and their values. If you delete a system default policy, it reappears using its original list of DHCP options and their system-defined values (see the table below)

Table 22: System Default Policy Option Values

System Default Option	Predefined Value
all-subnets-local	False
arp-cache-timeout	60 seconds
broadcast-address	255.255.255.255
default-ip-ttl	64
default-tcp-ttl	64
dhcp-lease-time	604800 seconds (7d)
ieee802.3-encapsulation	False
interface-mtu	576 bytes

System Default Option	Predefined Value
mask-supplier	False
max-dgram-reassembly	576 bytes
non-local-source-routing	False
path-mtu-aging-timeout	6000 seconds
path-mtu-plateau-tables	68, 296, 508, 1006, 1492, 2002, 4352, 8166, 17914, 32000
perform-mask-discovery	False
router-discovery	True
router-solicitation-address	224.0.0.2
tcp-keepalive-garbage	False
tcp-keepalive-interval	0 seconds
trailer-encapsulation	False

- **Named**—Policies you explicitly define by name. Named policies are usually named after their associated scope, prefix, or client grouping. For example, the policy might be assigned attributes and options that are unique to a subnet, such as for its routers, and then be assigned to the appropriate scope or prefix.

Cisco Prime Network Registrar includes a policy named **default** when you install the DHCP server. The server assigns this policy to newly created scopes and prefixes. You cannot delete this default policy.

- **Embedded**—A policy embedded in (and limited to) a named scope, scope template, prefix, prefix template, client, or client-class. An embedded policy is implicitly created (or removed) when you add (or remove) the corresponding object. Embedded policy options have no default values and are initially undefined.



Tip Be sure to save the object (scope, prefix, client, or client-class) for which you are creating or modifying an embedded policy. Not doing so is a common error when using the web UI. Click **Modify** for both the embedded policy and the parent object.

Policy Hierarchy

DHCPv4 Policy Hierarchy

To eliminate any conflicting attribute and option values that are set at various levels, the Cisco Prime Network Registrar DHCP server uses a local priority method. It adopts the more locally defined attribute and option values first while ignoring the ones defined on a more global level, and includes any default ones not otherwise

defined. When the DHCP server makes processing decisions for a DHCPv4 client, it prioritizes the attributes and options in this order:

1. Client embedded policy.
2. Client named policy.
3. Client-class embedded policy.
4. Client-class named policy.
5. Scope embedded policy for clients, or address block embedded policy for subnets.
6. Scope named policy for clients (or default policy if a named policy is not applied to the scope), or address block named policy for subnets.
7. Any remaining unfulfilled attributes and options in the `system_default_policy`. For attributes, the default value for the most local policy applies.



Note For DHCPv6 policy prioritization, see [DHCPv6 Policy Hierarchy, on page 165](#).

DHCPv6 Policy Hierarchy

DHCPv6 uses the existing policy objects, with additional DHCPv6 specific attributes (that are mostly analogous to those in DHCPv4). For DHCPv6, the hierarchy is:

1. Client embedded policy
2. Client named policy
3. Client-class embedded policy
4. Client-class named policy
5. Prefix embedded policy
6. Prefix named policy
7. Link embedded policy
8. Link named policy
9. `system_default_policy`

For attributes, the default value for the most local policy applies. This hierarchy is the same as for DHCPv4, except for the additional link policies and the fact that the prefix policies replace the scope policies. (For a comparison with the DHCPv4 policy hierarchy, see [DHCPv4 Policy Hierarchy, on page 164](#).)

The hierarchy applies to most policy attributes, which the server processes in the context of a single prefix. However, the server processes a few attributes (specifically *allow-rapid-commit*, *reconfigure*, *v6-reply-option*, *v6-options*, and *v6-vendor-options*) in the context of multiple prefixes. In these cases, the processing at the prefix levels (steps 5 and 6) is a bit different:

- For the *reconfigure* attribute that controls whether the server requires, allows, or disallows client reconfiguration, the server checks the embedded and named policies of all prefixes on the link that the client is allowed to use (based on selection tags). If any of the prefix policies have the *reconfigure* attribute set to **disallow** or **require**, the server uses that setting. Otherwise, if at least one policy has it set to **allow**, Reconfigure is allowed. Otherwise, the server checks the remaining policies in the hierarchy. (See the [Reconfiguring IPv6 Leases, on page 211](#) for details.)
- If the client requests Rapid Commit (see the [Editing DHCPv6 Server Attributes, on page 24](#)), the server checks the embedded and named policies of all prefixes on the link that the client is allowed to use (based on selection tags). If one of these policies has *allow-rapid-commit* disabled, the server processes the client request as if Rapid Commit were not part of the request. If at least one policy has *allow-rapid-commit*

enabled, the client can use Rapid Commit. If no prefix policy has the attribute configured, processing continues at step 7.

- For the options-related attributes (see [Setting DHCPv6 Options, on page 179](#)), the server also does special handling at steps 5 and 6. The server checks the embedded and then named policy of each prefix on the link. It then uses the first one with the configured *v6-reply-option* attribute, or the first one with the configured value for the *v6-options* or *v6-vendor-options*.
- The server checks the prefixes in case-insensitive alphabetical order.
- The server ignores any policies related to the location-independent and/or universal link and the prefixes under those. Only topological links (and prefixes under those links) are considered.



Tip In configurations with multiple prefixes on a link, avoid setting the Rapid Commit and option properties for the prefix policy, but rather set them on the link policy or other policy instead.

Creating and Applying DHCP Policies

This section describes how to create a policy at the DHCP server level and then allow specific scopes or prefixes to reference it. A policy can consist of a:

- **Name**—Not case-sensitive and must be unique.
- *permanent-leases* **attribute**—A permanent lease never expires.
- **Lease time**—How long a client can use an assigned lease before having to renew the lease with the DHCP server (the lease time attributes are not available for an embedded policy, only the option). The default lease time for both system default and default policies is seven days (604800 seconds). A policy contains two lease times—the client lease time and the server lease time:
 - **Client lease time**—Determines how long the client believes its lease is valid. (Set the client lease time using a DHCP option, not a policy attribute.)
 - **Server lease time**—Determines how long the server considers the lease valid. Note that the server lease time is independent of the lease grace period. The server does not allocate the lease to another client until after the lease time and grace period expire.



Caution Although Cisco Prime Network Registrar supports the use of two lease times for special situations, Cisco Systems generally recommends that you not use the *server-lease-time* attribute.

You can establish these two different lease times if you want to retain information about client DNS names and yet have them renew their leases frequently. When you use a single lease time and it expires, the server no longer keeps that client DNS name. However, if you use a short client lease time and a longer server lease time, the server retains the client information even after the client lease expires. For details on leases, see [Managing Leases, on page 185](#).

- **Lease grace period**—Time period after the lease expires that it is unavailable for reassignment (not available for an embedded policy).
- **DNS update configuration**—A DNS update configuration specifies the type of DNS updates to perform, the zones involved, the DNS server to be updated, and the related security. The policy determines the forward and reverse DNS update configuration objects, and can also specify the forward zone to use if

a DNS server hosts multiple zones. (For details on DNS update configurations, see [Creating DNS Update Configurations, on page 256](#).)

- **DHCP options**—To add option values, see [Setting DHCP Options and Attributes for Policies, on page 168](#).

Local Basic or Advanced and Regional Web UI

- Step 1** From the **Design** menu, choose **Policies** under the **DHCP Settings** submenu to open the List/Add DHCP Policies page.
- Step 2** The default policy and `system_default_policy` are already provided for you.
- Step 3** Click the **Add Policies** icon in the Policies pane, give the policy a unique name (required).
- Step 4** Set the offer timeout and grace period values or leave them blank.
- Step 5** Enter the DHCP Lease Time, if required and click **Add DHCP Policy** to add the named policy.
- Step 6** In the Edit DHCP Policy page, you can:
- Add the necessary DHCP options (see [Setting DHCP Options and Attributes for Policies, on page 168](#) like:
 - **Lease time**—Set the `dhcp-lease-time` (51) option.
 - **Limitation count**—See [Using Expressions, on page 315](#)
 - **Use client IDs for reservations**—See [Overriding Client Identifiers, on page 382](#).To set vendor-specific options, see [Using Standard Option Definition Sets, on page 172](#).
 - In Advanced mode, set the policy attributes, which include:
 - **Unavailable timeout**—See [Setting Timeouts for Unavailable Leases, on page 215](#).
 - **Inhibit all renews**—See [Inhibiting Lease Renewals, on page 212](#).
 - **Inhibit all renews at reboot**.
 - **Permanent leases** (not recommended).
 - **Lease retention limit**.
 - Set the DNS update configuration that determines which forward or reverse zones you want to include in a DNS update (**DNS Update Settings**). You can set:
 - ***forward-dnsupdate***—Name of the update configuration for the forward zone. Note that you can thereby set different update configurations for forward and reverse zones.
 - ***forward-zone-name***—If necessary, overrides the forward zone in the update configuration. Use this in case a DNS server is hosting multiple zones.
 - ***reverse-dnsupdate***—Name of the update configuration for the reverse zone. If not set on any policy in the policy hierarchy applicable to the client request (see [DHCPv4 Policy Hierarchy, on page 164](#)), the DHCP server uses the `forward-dnsupdate` configuration.
- Step 7** Click **Save**.
- Step 8** Reload the DHCP server.

In the regional web UI, you can also pull replica policies and push policies to local clusters. (See [Configuring DHCP Policies, on page 161](#) for regional policy management.)

CLI Commands

Use **policy name create** to create the policy. Then use **policy name set offer-timeout=value** and **policy name set grace-period=value** to set these two values.

To set policy options, use **policy name setOption <opt-name | id> value [-blob]**:

- **Lease time**—Use **policy name setLeaseTime time-val**.
- **Subnet mask**—Use a combination of **policy name setOption subnet-mask value** and **dhcp enable get-subnet-mask-from-policy**.

To confirm the option settings, use **policy name listOptions** or **policy name getOption <opt-name | id>**.

To enable permanent leases (not recommended), use **policy name enable permanent-leases**. Note that enabling permanent leases forces the *dhcp-lease-time* option (51) to be set to infinite.

Related Topics

[Types of Policies, on page 163](#)

[DHCPv4 Policy Hierarchy, on page 164](#)

[Cloning a Policy, on page 168](#)

[Setting DHCP Options and Attributes for Policies, on page 168](#)

[Creating and Editing Embedded Policies, on page 170](#)

[Creating DHCP Option Definition Sets and Option Definitions, on page 171](#)

Cloning a Policy

In the CLI, you can clone a policy from an existing one by using **policy clone-name create clone=policy, and** then make adjustments to the clone. For example:

```
nrcmd> policy cloned-policy create clone=example-policy-1 offer-timeout=4m
```

Setting DHCP Options and Attributes for Policies

DHCP options automatically supply DHCP clients with configuration parameters, such as domain, nameserver, and subnet router addresses (see [Creating DHCP Option Definition Sets and Option Definitions, on page 171](#)). Note that the Cisco Prime Network Registrar user interfaces allow you to set some option values on a policy that actually have no effect on the packet returned to the client (such as *hostname* and *dhcp-server-identifier*).

The server searches the policies, in order, for these BOOTP and DHCP attribute values and returns the first occurrence of these values in its reply packet:

- *packet-siaddr* returned in the *siaddr* packet field

- *packet-file-name* returned in the *file* field
- *packet-server-name* returned in the *sname* field

Related Topics

[Adding Option Values, on page 169](#)

[Adding Complex Values for Suboptions, on page 170](#)

Adding Option Values

You can view, set, unset, and edit DHCP option values. When you set an option value, the DHCP server replaces any existing value or creates a new one, as needed for the given option name. Cisco Prime Network Registrar DHCP options are grouped into categories to aid you in identifying options that you must set in various usage contexts. You can create custom option definitions to simplify entering custom option values (see [Creating Custom Option Definitions, on page 173](#)).

Local Basic or Advanced and Regional Web UI

Step 1 Create a policy, as described in [Creating and Applying DHCP Policies, on page 166](#).

Step 2 On the Edit DHCP Policy page, add each DHCP option to the policy by choosing its number and name in the drop-down list. The choices indicate the data type of the option value (see [Option Definition Data Types and Repeat Counts, on page 180](#)).

Tip You can sort the options by Name, Number, or (in the case of DHCPv4) Legacy (grouping).

Step 3 Add the appropriate option value in the Value field. The web UI does error checking based on the value entered. For example, to add the lease time for the policy, click the *[51] dhcp-lease-time (unsigned time)* option in the Number drop-down list, then add a lease time value in the Value field. (Options do not have preset values.)

Tip If you are configuring an option on a policy while another user is editing the option definition, log out of the session and log back in to get the new option definition.

Step 4 Click **Add Option** for each option. You must supply a value or you cannot add the option.

Step 5 Click **Save**.

Tip If you add new option values or edit existing ones, be sure to save the policy object by clicking **Save**.

CLI Commands

To view option values, use **policy name getOption** *<opt-name | id>* and **policy name listOptions**. To set option values, use **policy name setOption** *<opt-name | id>* *value* **[-blob]**. When you set an option value, the DHCP server replaces any existing value or creates a new one, as needed, for the given option name. To unset option values, use **policy name unsetOption** *<opt-name | id>*.

Adding Complex Values for Suboptions

If you are adding more complex option values such as for suboptions, use a parenthesized string format. The format requires that you:

- Enclose each option level (option, suboption, subsuboption) in parentheses.
- Separate multiple values with commas.
- Separate data fields for packed data (missing the suboption code or length) with semicolons.

For example, the *cablelabs-client-configuration* option (122) normally has 10 suboptions as well as some subsuboptions. This example shows the syntax to set the suboption 1, 2, 3, and 4 data values, and includes the two subsuboptions for suboption 3 and the three subsuboptions for suboption 4 (which are packed data and have no code numbers):

```
(primary-dhcp-server 1 10.1.1.10)
(secondary-dhcp-server 2 10.2.2.10)
(provisioning-server 3 (flag 0; provisioning-server server.example.com.))
(as-backoff-retry 4 (as-backoff-retry-initial-time-ms 10;
as-backoff-retry-max-time 10s; as-backoff-retry-count 100))
```

The suboption name (such as *primary-dhcp-server*) is optional. Hence, it is often safer to use just the code number and data value (or just the data value for packed data) to minimize typographical errors and parsing failures. The compacted (and preferred) version of the previous example that strips out the suboption names is:

```
(1 10.1.1.10) (2 10.2.2.10) (3 (0;server.example.com.)) (4 (10;10s;100))
```

Even if you use numerical code values, Cisco Prime Network Registrar always includes the equivalent names when it displays the suboptions (see [Creating DHCP Option Definition Sets and Option Definitions, on page 171](#)).

To include suboptions that include enterprise IDs (such as for option 125), use the following format, for example, when entering in the policy option value:

```
(enterprise-id 1((1 10.1.1.1) (2 10.2.2.2) (3 www.cisco.com)))
```

The parentheses surround the enterprise ID itself, the suboptions as a group, and each suboption.

Creating and Editing Embedded Policies

An embedded policy is embedded for a DHCPv4 scope or scope template, DHCPv6 prefix or prefix template, client, or client-class. You can create or edit an embedded policy.

Local Advanced Web and Regional UI

-
- Step 1** From the **Design** menu, choose one of the following that appear for DHCPv4 or DHCPv6 in the local web UI: **Scopes**, **Scope Templates**, **Clients**, **Client-Classes**, **Prefixes**, or **Links**. (The regional web UI can have the selections **Scope Templates**, **Client-Classes**, **Prefixes**, and **Links**.)
- Step 2** Click the name of the object on the left pane to open its Edit page.

- Step 3** Click **Create New Embedded Policy** or **Edit Existing Embedded Policy** under the Embedded Policy section of the page. This opens the Edit DHCP Embedded Policy page for the object.
- Step 4** Make changes to the values as needed, then click **Modify Embedded Policy**.
- Step 5** On the Edit page for the object, be sure to save the changes by clicking **Save**.
-

CLI Commands

Use the embedded commands, such as **client-class-policy** *client-class-name* **set attribute=value**, where the command starts with the object name followed by -policy.

Creating DHCP Option Definition Sets and Option Definitions

In Cisco Prime Network Registrar, you configure option values on policies for such things as lease times and router addresses. Numerous RFCs describe the formatting of DHCP option values, beginning with RFC 2132. Option definitions are used in the web UI and CLI to control formatting of option values in policies.

DHCPv6 options do not use DHCPv4 options; they are unique and separate. There are currently about 46 DHCPv6 options. Most of these options are the DHCPv6 protocol infrastructure options and are not user-definable. They use a 16-bit option code and 16-bit length (DHCPv4 uses only 8 bits for both of these). Configuring options and the behavior of configured options in policies are similar to those for DHCPv4. See [Setting DHCPv6 Options, on page 179](#) for details about client processing as it relates to the policy hierarchy.

You can define option definitions separately for the DHCPv4 and DHCPv6 address spaces, as:

- **Standard (built-in) options** -Defined by the RFCs. In the web UI, these are in the **dhcp-config** and **dhcp6-config** definition sets. The CLI includes additional **dhcp-default** and **dhcp6-default** definition sets that are hidden, but accessible if you call for them specifically. (See [Using Standard Option Definition Sets, on page 172](#).)
- **Custom options** -New or modified definitions in the supplied **dhcp-config** or **dhcp6-config** definition sets. Once you add or modify definitions in the web UI, they are added to the **dhcp-custom** or **dhcp6-custom** definition sets in the CLI. (See [Creating Custom Option Definitions, on page 173](#).)
- **Vendor-specific options** -Defined in their own definition sets. The CableLabs definition sets (**dhcp-cablelabs-config** and **dhcp6-cablelabs-config**) are preconfigured in Cisco Prime Network Registrar. The CLI also includes **dhcp-cablelabs-default**, **dhcp6-cablelabs-default**, **dhcp-cablelabs-custom**, and **dhcp6-cablelabs-custom** definition sets. (See [Using Standard Option Definition Sets, on page 172](#).)

Related Topics

- [Using Standard Option Definition Sets, on page 172](#)
- [Creating Custom Option Definitions, on page 173](#)
- [Creating Vendor-Specific Option Definitions, on page 173](#)
- [Option Definition Data Types and Repeat Counts, on page 180](#)
- [Adding Suboption Definitions, on page 181](#)
- [Importing and Exporting Option Definition Sets, on page 181](#)

[Pushing Option Definition Sets to Local Clusters, on page 182](#)

[Pulling Option Definition Sets from Replica Data, on page 182](#)

[Setting Option Values for Policies, on page 179](#)

Using Standard Option Definition Sets

Cisco Prime Network Registrar provides two standard, built-in option definition sets, **dhcp-config** and **dhcp6-config**, for DHCPv4 and DHCPv6 option definitions, respectively. You can create new options definitions in these sets or you can overwrite existing ones. New option definitions or ones that were overwritten are identified by an asterisk (*). You can delete these definitions and there is no deletion confirmation given. However, saving the set after deleting an overwritten definition causes the original definition to reappear in the set.



Caution Arbitrarily modifying the standard definitions (or adding suboption definitions) can adversely affect configurations.

Local Advanced and Regional Web UI

-
- Step 1** From the **Design** menu, choose **Options** under the **DHCPv4** or **DHCPv6** submenu to open the List/Add DHCP Option Definition Sets page. (DHCP option definition is not available in Basic mode.)
- Step 2** Click the **dhcp-config** (DHCPv4) or **dhcp6-config** (DHCPv6) link to open the Edit DHCP Option Definition Set page, then click the **Option Definitions** tab. View the predefined definitions on the List DHCP Option Definitions page. These are the definitions that control the formatting of the option values you add to policies. If there are suboption definitions, you can expand to show them.
- Step 3** To add a definition, click the **Add Option Definition** icon. On the Edit DHCP Option Definition page, give the option an number, name, description, type, and repeat count (whether more than one instance of the option is allowed or required). (For details on the data types and repeat count values, see [Option Definition Data Types and Repeat Counts, on page 180](#).)
- Note** You cannot add an option definition for an option number or name that already exists. However, you can modify any option definition that appears as a hyperlink on the page.
- Step 4** Click **Add Option Definition**. Then, on the List/Add DHCP Option Definition Sets page, click **Save**.
- Step 5** Click the **Revert** button if you want to revert to the original definitions in that standard set.
- Step 6** In the regional web UI, you can also pull replica definition sets and push definition sets to local clusters. (See [Pulling Option Definition Sets from Replica Data, on page 182](#) and [Pushing Option Definition Sets to Local Clusters, on page 182](#).)
-

CLI Commands

To view the entire list of standard DHCP option definitions, use **option-set dhcp-config [show]** or **option-set dhcp6-config [show]**, or **option {id | name} option-set show** to view a specific definition. For example:

```
nrcmd> option-set dhcp-config
nrcmd> option subnet-mask dhcp-config show
```

To add a definition to a set, use **option** *id option-set create option-name type [attribute=value]*. You cannot add a definition for an option ID (number) or name that already exists. For example, to add option number 222 with the name example-option in the dhcp-config option set, with a string type, use:

```
nrcmd> option 222 dhcp-config create example-option AT_STRING
```

To get a particular option attribute value, use **option** *{id | name} option-set get attribute*. To modify an option attribute, use **option** *{id | name} option-set set attribute=value*. You can also unset an option attribute.

Creating Custom Option Definitions

You can create custom option definitions in the standard sets. Click the **dhcp-config** or **dhcp6-config** set on the List/Add DHCP Option Definition Sets page. Then proceed with **Step 3** in [Using Standard Option Definition Sets](#), on page 172.

Creating Vendor-Specific Option Definitions

You can send vendor-specific option data to DHCP clients that request them.



Note

There are several option codes set aside for vendor-specific options, so that you must explicitly specify the option code number for which you are creating a vendor-specific option definition.

In Cisco Prime Network Registrar, you can create vendor-specific option definitions in the web UI, or in the CLI by using **option** *id option-set-name create*. (For details on the option data types, see [Option Definition Data Types and Repeat Counts](#), on page 180.)

Vendor-specific options are sent in the following DHCP options:

- **vendor-encapsulated-options (43)**—Set this to a binary data type, then add the vendor-specific suboption definitions. (The data type of the parent option definition is a placeholder only. The suboption definitions define the valid option value formatting.)
- **v-i-vendor-info (125) or vendor-options (17) for DHCPv6**—Set this to a vendor-opts data type, then add the vendor-specific suboption definitions.

You can create vendor-specific option definitions for DHCPv4 options 43 and 125, and DHCPv6 option 17. You add the vendor-specific option definitions into a vendor option definition set that you create.



Caution

Changing option definition properties, or deleting the option definition altogether, can have unexpected side effects on policies. If you delete a custom option definition, also check for the policies that include an option value. Changing an option definition changes the way that they are displayed, not what is stored, so that you do not need to modify the policy value unless you want the policy to return a differently formatted option value. Some option types are very similar, and changing between them can have side effects. For example, strings and DNS names are both entered as string values in the user interfaces, but the formatted option values are quite different.



Note Cisco Prime Network Registrar 7.0 preconfigures separate CableLabs (enterprise ID 4491) option definitions in the **dhcp-cablelabs-config** and **dhcp6-cablelabs-config** vendor-specific option definition sets.

Local Advanced and Regional Web UI

- Step 1** From the **Design** menu, choose **Options** under the **DHCPv4** or **DHCPv6** submenu to open the List/Add DHCP Option Definition Sets page. View the existing DHCPv4 or DHCPv6 options.
- Step 2** Click the **Add Options** icon in the Options pane to open the Add OptionDefinitionSet dialog box.
- Step 3** Enter a name for the option definition set, then choose DHCPv4 or DHCPv6 from the DHCP Type drop-down list.
- If you are creating vendor-specific option definitions using:
- Option 43, enter a value in the Vendor Option String field. (See the subsequent section for a sample procedure on creating a vendor option set and vendor option values for option 43.)
 - Option 125 for DHCPv4 or option 17 for DHCPv6, enter a valid Enterprise Option Enterprise ID value.
- Step 4** Click **Add OptionDefinitionSet**.
- Step 5** Click the added option definition set name on the left pane.
- Step 6** On the Edit DHCP Option Definition Set page, click the **Option Definitions** tab. Any existing option definitions will appear on this page (new or modified standard definitions are marked with an asterisk).
- Step 7** Click **Add Option Definition** icon. Enter the ID number of the option definition, along with its name and a description. The ID must be 43, 125, or 17 (for DHCPv6) for the client to recognize a vendor-specific option definition. The option name does not need to match the one specified in the RFC and can be of your own creation.
- Step 8** Choose a data type and repeat count (or enter an absolute repeat count in the next field). The data type must be:
- Binary (AT_BLOB) for option 43.
 - Vendor-opts (AT_VENDOR_OPTS) for option 125 (for DHCPv4) and option 17 (for DHCPv6).
- (For details on the data type and repeat count values, see [Option Definition Data Types and Repeat Counts, on page 180](#).)
- Step 9** Click **Add Option Definition**. Then, on the List DHCP Option Definitions page, click **Save**.

Local Advanced and Regional Web UI

Using the Local Advanced web UI to create vendor option set and vendor option values for option 43:

- Step 1** From the **Design** menu, choose **Options** under the **DHCPv4** or **DHCPv6** submenu to open the List/Add DHCP Option Definition Sets page.
- Step 2** Click the **Add Options** icon in the **Options** pane to open the Add OptionDefinitionSet dialog box.
- Step 3** Enter values for the following attributes:

Name	Name of the option definition set; for example, AP1130.
------	---

DHCP Type	Byte size of the type identifiers for all children in this set. You must choose DHCP v4 from the drop-down list.
Vendor Option String	Exact vendor class identifier string from option-60 that the DHCP client device vendor provides. For example, Cisco AP c1130.

Step 4 Click **Add OptionDefinitionSet**.

The List/Add DHCP Option Definition Sets page appears.

Step 5 Click AP1130, the name of the option definition set that appears.

The Edit DHCP Option Definition Set AP1130 page appears.

Step 6 Click the **Option Definitions** tab and then click **Add Option Definition**.

Step 7 Enter the values for the following attributes:

Number	Number of the option code. You must enter 43.
Name	Name of this attribute. For example, ap1130-option-43.
Type	Datatype for the option value. You must choose binary from the drop-down list.

Step 8 Click **Add Option Definition**.

Note that clicking this button does not save the changes that you make to the option definition set. It only lists the option definition set on the List DHCP Option Definitions page.

Step 9 In the Option Definitions tab, click the name of the new option definition (ap1130-option-43), then **Add Sub-Option Definition**.

Step 10 In the Add DHCP Option Definition page, enter values for the following attributes:

Number	The option code for this suboption. For this example, you must enter 241.
Name	Name of this attribute. For example, "ap1130-suboption-241".
Type	Datatype for the suboption value. For this example, you must choose IP Address from the drop-down list.
Repeat	The repeat count for this type. For this example, you must choose 1+ from the drop-down list.

Step 11 Click **Add Option Definition**, then **Save**.

Step 12 Click **Design**, then **Policies** under the **DHCP Settings** submenu to open the List/Add DHCP Policies page.

Step 13 Choose the policy for which to set this option; or, add a new policy in the Advanced mode.

Depending on your selection, the Edit DHCP Policy policy_name or the Add DHCP Policy page appears.

Step 14 From the DHCP v4 Vendor Options drop-down list, choose the name of the option definition set (AP1130), and click **Select**.

Step 15 Choose the option definition from the Name drop-down list (“ap1130-option-43”) and, in the Value field, enter, for example:

```
(241 3.3.3.3,4.4.4.4)
```

Step 16 Click **Add Option**, then click **Save**.

Step 17 Reload the DHCP server.

Example: Creating Vendor Option Set for Cisco AP Devices

You can create a vendor option set and vendor option values from the CLI for Cisco Access Point (AP) devices, SunRay devices, and Cisco 79xx IPPhones using the sample procedures described in this section.

Using option 43 for Lightweight Access Point Protocol (LWAPP) APs requires vendor option 43 if you are using Cisco Prime Network Registrar as the DHCP server. This example is specific to the Cisco Aironet 1130 series. You can modify the example to configure option 43 for other vendor options, such as Cisco Aironet 1200 series and Cisco Aironet 1240 series.

Step 1 Create a .txt file with the following content:

```
#
# Version: 1
# 6.2+ Option-set example for Option 43 with suboptions for Cisco APs
#
# NOTE: Need to edit vendor option string to Exact match AP Model string in Option-60.
#
# For compatibility with pre-6.2 vendor options ensure that
# name=vendor-option-string. (Not True in this test example.)
# =====
{
  ( id-range = 1 )
  ( vendor-option-string = Cisco AP c1130 )
  ( name = Aptest )
  ( children = [
    {
      ( id = 43 )
      ( name = pxe-sample )
      ( desc = )
      ( base-type = AT_BLOB )
      ( children = [
        {
          ( id = 241 )
          ( name = controller )
          ( desc = ap controller )
          ( base-type = AT_IPADDR )
          ( repeat = ONE_OR_MORE )
        } ]
      )
    } ]
  )
}
```

Step 2 Save the file as *OptionSetCiscoAP.txt* at the following location:

- Windows—\Program Files\Network Registrar\Local\bin
- Linux—/opt/nwreg2/local/usrbin

Step 3 Import the OptionSetCiscoAP.txt file from the CLI using the import option-set file command. For example:

```
nrcmd> import option-set OptionSetCiscoAP.txt
```

(For information on importing option definition sets, see [Importing and Exporting Option Definition Sets, on page 181.](#))

Step 4 Set the vendor-specific option data on a policy using the **policy name setVendorOption <opt-name | id> opt-set-name value [-blob]** command.

For example, to set vendor option 43 data for the optionset APtest with values (241 3.3.3.3,4.4.4.4), on an existing policy with the name test, use:

```
nrcmd> policy test setVendorOption 43 APtest "(241 3.3.3.3,4.4.4.4)"
nrcmd> save
```

Step 5 Reload the DHCP server.

```
nrcmd> dhcp reload
```

Example: Creating Vendor Option Set for SunRay Devices

Use this sample procedure to create vendor option set with multiple suboptions for SunRay Devices:

Step 1 Create a .txt file with the following content:

```
#
# Option Definition Set Export/Import Utility
# Version: 1
# 6.2 Option-set example for Option 43 with suboptions for Sun SunRay.
#
# NOTE: Need to edit vendor option string to match Option-60
#
# For compatibility with pre-6.2 vendor options ensure that
# name=vendor-option-string.
# =====
{
  ( id-range = 1 )
  ( vendor-option-string = sunray )
  ( name = sunray )
  ( children = [
  {
    ( id = 43 )
    ( name = option43 )
    ( desc = )
    ( base-type = AT_BLOB )
    ( children = [
    {
      ( id = 21 )
      ( name = AuthSrvr )
      ( desc = AuthSrvr )
      ( base-type = AT_IPADDR )
      ( repeat = ONE_OR_MORE )
    }
  ]
  }
  {
    ( id = 35 )
    ( name = AltAuth )
    ( desc = AltAuth )
    ( base-type = AT_IPADDR )
    ( repeat = ONE_OR_MORE )
  }
  ]
}
```

Example: Creating Option Set for Cisco 79xx IPPhones

```
{
  ( id = 36 )
  ( name = BarrierLevel )
  ( desc = BarrierLevel )
  ( base-type = AT_SHORT )
}
]
)
} ]
)
}
```

Step 2 Save the file as *OptionSetSunRay.txt* at the following location:

- Windows—\Program Files\Network Registrar\Local\bin
- Linux—/opt/nwreg2/local/usrbin

Step 3 Import the OptionSetSunRay.txt file from the CLI using the import option-set file command. For example:

```
nrcmd> import option-set OptionSetSunRay.txt
```

(For information on importing option definition sets, see [Importing and Exporting Option Definition Sets, on page 181](#).)

Step 4 Set the vendor-specific option data on a policy using the **policy name setVendorOption <opt-name | id> opt-set-name value [-blob]** command.

For example, to set vendor option 43 data for the optionset APtest with values (241 3.3.3.3,4.4.4.4), on an existing policy with the name test, use:

```
nrcmd> policy test setVendorOption 43 APtest "(241 3.3.3.3,4.4.4.4)"
nrcmd> save
```

Step 5 Reload the DHCP server.

```
nrcmd> dhcp reload
```

Example: Creating Option Set for Cisco 79xx IPPhones

Use this sample procedure to create option set for Cisco 79xx IPPhones:

Step 1 Define the option.

```
nrcmd> option 150 dhcp-custom create voip-tftp-server AT_IPADDR desc="VOIP Option-150 Server"
repeat=ONE_OR_MORE
```

Step 2 Display the configured option.

```
nrcmd> option dhcp-config list
```

Step 3 Set policy, by using **policy default setOption voip-tftp-server ip-address**. For example:

```
nrcmd> policy default setOption voip-tftp-server 192.168.1.254
```

Step 4 Confirm the policy setting.

```
nrcmd> policy default getOption voip-tftp-server
```

Step 5 Reload the DHCP server.

```
nrcmd> dhcp reload
```

Setting Option Values for Policies

You enter option values on a policy. The option definitions in your server configuration control the format and values that you enter.

Local Advanced and Regional Web UI

On the List/Add DHCP Policies page, click a policy to edit it. (Note that you cannot set options for policies in Basic mode.) On the Edit DHCP Policy page:

- To enter a standard DHCPv4 or DHCPv6 option value for a policy, choose it from the DHCPv4 Options or DHCPv6 Options drop-down list, then set a value for the option. Click **Add Option**.
- To enter a vendor-specific DHCPv4 or DHCPv6 option value for a policy, choose an option definition set in the DHCPv4 Vendor Options or DHCPv6 Vendor Options drop-down list, then click **Select**. The page changes to show the drop-down list that includes the option; choose it, then click **Add Option**.

Note that you can also edit policy attributes on this page. Be sure to click **Modify Policy**.

To edit a configured policy option, click the name of the configured option on the Edit DHCP Policy page to open the Edit DHCP Policy Option page. Enter a new value, then click **Modify Option**.

CLI Commands

Use one of these commands:

```
nrcmd> policy name setOption {opt-name | id} value [-blob]
nrcmd> policy name setV6Option {opt-name | id}[.instance] value [-blob]
nrcmd> policy name addV6Option {opt-name | id}[.instance] value [-blob]
nrcmd> policy name setVendorOption {opt-name | id} opt-set-name value [-blob]
nrcmd> policy name setV6VendorOption {opt-name | id} opt-set-name value [-blob]
```

To list the options in the policy, use one of these commands:

```
nrcmd> policy name listOptions
nrcmd> policy name listV6Options
nrcmd> policy name listVendorOptions
nrcmd> policy name listV6VendorOptions
```

To add suboption values, see [Adding Complex Values for Suboptions, on page 170](#).

Setting DHCPv6 Options

Set DHCPv6 options and vendor options when you create or edit policies (embedded or named) for prefixes. (See [DHCPv6 Policy Hierarchy, on page 165](#) for special handling of the *v6-options* and *v6-vendor-options* policy attributes when used in an embedded or named policy on a prefix.)

Local Advanced Web UI

The DHCPv6 options coexist along with the DHCPv4 options on the List/Add DHCP Policies or Edit DHCP Policy page. Note that the vendor options appear only if you create these options (see [Creating DHCP Option Definition Sets and Option Definitions, on page 171](#)).

You can select the options from the drop-down lists. If option descriptions exist, they appear under the Name and Number headings, which you can click to sort the entries.

CLI Commands

Use **policy name setV6Option** {*opt-name* | *id*}[*.instance*] *value* [-**blob**] or **policy name setV6VendorOption** {*opt-name* | *id*} *opt-set-name* *value* [-**blob**]. The option settings require an option name (or ID) and a value. For example:

```
nrcmd> policy dhcpv6-policy setV6Option dns-servers 2222::1,2222::2
```

```
nrcmd> policy foo setV6VendorOption 17 dhcp6-cablelabs-config "(32 2222::3,2222::4)"
```

Option Definition Data Types and Repeat Counts

The data type values that you can use appear in the following table.

Table 23: Option Definition Data Types

AT_INT8 unsigned 8-bit	AT_SHORT unsigned 16-bit	AT_INT unsigned 32-bit	AT_STRING string
AT_SINT8 signed 8-bit	AT_SSHORT signed 16-bit	AT_SINT signed 32-bit	AT_NSTRING string (no termination)
AT_DNSNAME DNS name	AT_SHRTI unsigned 16-bit (Intel)	AT_INTI unsigned 32-bit (Intel)	AT_BLOB binary
AT_RDNSNAME relative DNS name	AT_SSHRTI signed 16-bit (Intel)	AT_SINTI signed 32-bit (Intel)	AT_DATE date
AT_VENDOR-CLASS vendor-class	AT_IPADDR IP address	AT_BOOL boolean	AT_TIME unsigned time
AT_VENDOR_NOLEN vendor-nolen	AT_IP6ADDR IPv6 address	AT_MACADDR MAC address	AT_STIME signed time
AT_VENDOR_OPTS vendor-opts	AT_ZEROSIZE zero size	AT_VPREFIX IPv6 variable-length prefix	

You can view these types in the CLI by using **option listtypes**.

To set the repeat count, set the *repeat-count* attribute to one of the following, or enter an absolute number:

- **ZERO_OR_MORE**—0+ in the web UI
- **ONE_OR_MORE**—1+ in the web UI
- **EVEN_NUMBER**—2n in the web UI

In the CLI, for example, use:

```
nrcmd> option 200 ex-opt-def-set set repeat-count=ZERO_OR_MORE
```

```
nrcmd> save
```

Adding Suboption Definitions

You can set a suboption definition for the option definition by clicking **Add Suboption Definition** on the Edit DHCP Option Definition page. This opens the Add DHCP Option Definition page, where you can add the same values as for an option definition. The suboption definition you create is associated with its parent option (or parent suboption) definition. You can define up to six option and suboption levels.



Note You can add suboption definitions by using the web UI only. You currently cannot do so by using the CLI.

Suboption definition formats can be packed or type/length/value (TLV):

- **Packed**—A suboption with a zero ID value and an implicit data type. The option value is the only data in the packet. DHCPv6 options are virtually all defined with packed data. There are no markers for type or length and the layout of the data is inherent in the option definition. You cannot have further suboption definitions for packed suboptions.
- **TLV**—A suboption with a value of 1 through 255 (or 65535) that includes a type, length, and value. The data in the packet has the type and length preceding the value.

In most cases, you will not be mixing packed with TLV suboptions for the same option.



Note DHCP server does not support suboption 0 definitions (for the DHCPv4 vendor-encapsulated-options (43) and v-i-vendor-opts(125) options, and DHCPv6 vendor-opts (17) option). Suboption with a zero ID value is used by DHCP server to specify packed data as mentioned above.

To enter suboption values when editing policies, see [Adding Complex Values for Suboptions, on page 170](#).

Option Definition Set

Importing and Exporting Option Definition Sets

Importing and exporting option definition sets is a way to copy them between servers. In the CLI, you can import and export option sets by using **import option-set *file*** and **export option-set *name file***.

For example, to import an option set for Preboot Execution Environment (PXE) clients, modify and import a sample file located in the /examples/dhcp directory:

```
nrcmd> import option-set /examples/dhcp/OptionSetPXE.txt
```



Caution Do not export the built-in option definition sets (such as dhcp-config and dhcp-cablelabs-config) and then reimport them. Reimporting an edited option definition set without TAC assistance can cause the server to fail.

Some of the guidelines for the file format include:

- The version string in the file must match the version for the import utility.

- The utility imports just the first option definition set found in the file.
- Delimit objects using curly brackets ({}), attributes using parentheses (()), and lists of objects in attributes using square brackets ([]). Delimit string value attributes using quotes (" ").

Using some care, you can also edit the text file to make minor modifications to an option definition set. Cisco Prime Network Registrar provides two sample option definition set text files in the examples/dhcp directory, OptionSetJumpStart.txt and OptionSetPXE.txt:

- **OptionSetJumpStart.txt**—Edit the vendor-option-string to match the dhcp-class-identifier (option 60) that your JumpStart clients are sending.
- **OptionSetPXE.txt**—Edit the vendor-option-string to match the dhcp-class-identifier (option 60) that your Pre-boot Execution Environment (PXE) clients are sending.

Pushing Option Definition Sets to Local Clusters

You can push option definition sets you create from the regional cluster to any of the local clusters. If you want to push a specific option definition set to a cluster, click **Push Option Definition** sets on the List/Add DHCP Option Definition Sets page, which opens the Push DHCP Option Definition Set to Local Clusters page.

This page identifies the data to push, how to synchronize it with the local cluster, and the cluster or clusters to which to push it. The data synchronization modes are:

- **Ensure** (preset value)—Ensures that the local cluster has new data without affecting any existing data.
- **Replace**—Replaces data without affecting other objects unique to the local cluster.
- **Exact**—Available for “push all” operations only. Use this with caution, because it overwrites the data and deletes any other objects unique to the local cluster.

Choose the destination cluster or clusters in the Available field and move it or them to the Selected field.



Tip The synchronization mode and cluster choice settings are persistent for the duration of the current login session, so that they are in effect each time you access this page, unless you change them.

After making these choices, click **Push Data to Clusters**. This opens the View Push DHCP Option Definition Set Data Report page.

Pulling Option Definition Sets from Replica Data

You may choose to pull option definition sets from the replica data of the local clusters instead of explicitly creating them. (You may first want to update the option definition set replica data by clicking the **Replicate** icon next to the cluster name.) To pull the option definition sets in the web UI, click **Pull Replica Option Definition Sets** to open the Select Replica DHCP Option Definition Set Data to Pull page.

This page shows a tree view of the regional server replica data for the local clusters’ option definition sets. The tree has two levels, one for the local clusters and one for the scope templates in each cluster. You can pull individual option definition sets from the clusters, or you can pull all of their option definition sets. To pull individual ones, expand the tree for the cluster, then click **Pull Option Definition Set** next to its name. To pull all the ones from a cluster, click **Pull All Option Definition Sets from Cluster**. To pull the option definition sets, you must also choose a synchronization mode:

- **Ensure**—Ensures that the regional cluster has new data without affecting any existing data.

- **Replace** (preset value)—Replaces data without affecting other objects unique to the regional cluster.
- **Exact**—Available for “pull all” operations only. Use this with caution, because it overwrites the data and deletes any other objects unique to the regional cluster.



CHAPTER 8

Managing Leases

Leases are at the center of the Dynamic Host Configuration Protocol (DHCP). They are the IP addresses allocated to individual clients for a certain time period. The DHCP server automatically allocates these leases with properly configured scopes that include valid IP address ranges. No two clients can have the same leased address. Reservations are leases that always get the same IP address.

This chapter describes how to manage leases and reservations in a network.

- [Lease States, on page 185](#)
- [Guidelines for Lease Times, on page 186](#)
- [DHCPv6 Clients and Leases, on page 189](#)
- [Configuring Leases in Scopes, on page 191](#)
- [Viewing Leases, on page 191](#)
- [Using Client Reservations, on page 199](#)
- [Creating Lease Reservations, on page 202](#)
- [Setting Advanced Lease and Reservation Properties, on page 207](#)
- [Querying Leases, on page 216](#)
- [Running Address and Lease Reports, on page 223](#)
- [Dynamic Lease Notification, on page 231](#)
- [Sample Lease Notification Client, on page 232](#)
- [Lease History Database Compression Utility , on page 238](#)

Lease States

The tables below list the IPv4 or IPV6 lease states.

IPv4 Lease States

A IPv4 lease can be in one of the states described in the table below.

Table 24: IPv4 Lease States

State	Description
Available	IP address available to be leased.

State	Description
Unavailable	Not leasable. See Handling Leases Marked as Unavailable, on page 214 for ways the DHCP server might set a lease to unavailable.
Leased	Held by a client.
Offered	Offered to the client.
Expired	Available when the lease grace period expires.
Deactivated	Not renewable or leasable after the lease expires. See Deactivating Leases, on page 195 .
Pending available	Failover-related. A lease in the pending-available state is available as soon as the server synchronizes its state with the failover partner. See Managing DHCP Failover, on page 55 .

IPv6 Lease States

A lease can be in one of the states described in the table below.

Table 25: IPv6 Lease States

State	Description
Available	IP address available to be leased.
Offered	Offered to the client.
Leased	Held by a client.
Expired	Available when the lease grace period expires.
Unavailable	Not leasable. It was made unavailable because of some conflict.
Released	The client has released the lease, but the server is configured to apply a grace period to the lease. The lease will not be made available until the grace period expires.
Other available	Failover-related. Available for allocation by the failover partner but not available for allocation by this server.
Pending available	Failover-related. A lease in the pending-available state is available as soon as the server synchronizes its state with the failover partner. Used for only prefix delegation leases.
Pending delete	Failover-related. A lease in the pending-delete state is disassociated from its client as soon as the server synchronizes its state with the failover partner.

Guidelines for Lease Times

To define appropriate values for lease times, consider these events on your network:

- Frequency of changes to DHCP options and default values.

- Number of available IP addresses compared to clients requesting them.
- Number of network interface failures.
- Frequency at which computers are added to and removed from the network.
- Frequency of subnet changes by users.

All these events can cause clients to release IP addresses or the leases to expire at the DHCP server. Consequently, the addresses may return to the free-address pool for reuse. If many changes occur on your network, Cisco recommends a lease time between one and three days for active networks, and between four and ten days for inactive networks. Assigning such a lease time reassigns IP addresses more quickly as clients leave the subnet.

Another important factor is the ratio of available addresses to connected computers. For example, the demand for reusing addresses is low in a class C network having 254 available addresses, of which only 40 are used. A long lease time, such as two months, might be appropriate in such a situation. The demand would be much higher if there were 240 to 260 clients trying to connect at one time. In this situation, you should try to configure more address space. Until you do, keep the DHCP lease time to under an hour.



Tip Short lease periods increase the demand that the DHCP server be continuously available, because clients will be renewing their leases more frequently. The DHCP failover functionality can help guarantee such levels of availability.

Be careful when creating policies that have permanent leases. A certain amount of turnover among clients occurs, even in a stable environment. Portable hosts might be added and removed, desktop hosts moved, and network adapter cards replaced. If you remove a client with a permanent lease, it requires manual intervention in the server configuration to reclaim the IP address. It would be better to create a long lease, such as six months, to ensure that addresses are ultimately recovered without administrator intervention.

Recommendations for lease durations include:

- Set cable modem lease times to seven days (604800 seconds). The leases should come from private address space, and the cable modems should seldom move around.
- Leases for customer premises equipment (CPE) or laptops should come from public address space and should match the habits of the user population, with as long a lease as possible to reduce load on the server.
- Shorter lease times require more DHCP request and response buffers. Set the request and response buffers for optimal throughput (see [Setting DHCP Request and Response Packet Buffers, on page 69](#)).
- Allow the server to determine the lease period, by ensuring that the *allow-lease-time-override* policy attribute is disabled, which is its normal default. Even if enabled, clients can only request lease times that are shorter than you configure for the server. Some clients always request a fixed lease time (such as an hour) or the same one they had previously. These kinds of requests can cause problems in that the client never gets the full lease time, thereby generating more traffic for the server.
- Defer any lease extensions for clients trying to renew leases before the halfway mark in the lease. For details, see [Deferring Lease Extensions, on page 22](#).

Restricting Lease Dates

Lease date restrictions can be specified using the following attributes:

- lease-retention-max-age
- lease-retention-min-age

The *lease-retention-max-age* attribute specifies the longest time, in the past (from the current time), to which lease times are restricted. This can be used to meet data retention restrictions for privacy protection. If not specified, no restrictions are placed on how far back in time the lease times may be. In order for lease retention limitation to take place for a lease, not only does the *lease-retention-max-age* need to be non-zero, but the individual lease itself must fall under a policy where the *lease-retention-limit* attribute is set in that policy. This value, if configured, must be greater than 8 hours. If it is configured as non-zero and less than eight hours, it will be set to eight hours.

The *lease-retention-min-age* attribute specifies the shortest time, in the past, to which lease times may be restricted. Its value must be at least 6 hours less than the *lease-retention-max-age*. If this attribute is enabled and is configured to a non-zero value, lease times subject to retention limitation will not be allowed to grow older than *lease-retention-max-age*. As they progress toward *lease-retention-max-age*, they are periodically reset to *lease-retention-min-age* in the past. Configuring this attribute is optional as it will be six hours less than the *lease-retention-max-age*, by default. Also if the difference between the attribute values is less than six hours then *lease-retention-max-age* minus six hours is used.

Keeping older times on a lease between *lease-retention-min-age* and *lease-retention-max-age* involves some processing, and the closer these two values are, the more frequently this processing must take place, regardless of the absolute values of these attributes. Setting the *lease-retention-min-age* to several days before the *lease-retention-max-age* minimizes the additional server processing devoted to lease retention limitation.

You have to change one or more policies for the clients which are subject to these retention times. You can configure this in the *system_default_policy* to apply to all clients. But if there are some devices for which this does not matter, it might be best to configure it more selectively. The fewer the clients with this feature enabled, the lesser the impact on the performance of the server because of lesser work.

The policy attribute *lease-retention-limit* indicates whether the clients associated with that policy are subject to the lease date restrictions. If this attribute is enabled and the *lease-retention-max-age* of the DHCP server is configured to a non-zero value, lease times subject to this policy will not be allowed to grow older than *lease-retention-max-age*. As they progress toward *lease-retention-max-age*, they will periodically be reset to *lease-retention-min-age* in the past.

Some points to remember when considering to use the privacy protection feature are:

- When first enabled (or for certain reconfigurations), existing lease history records will not be subject to this feature because these records will not have the *lease-retention-limit* flag set.
- Detailed lease history is disabled if the limiting retention feature is enabled. This is not an issue if detailed lease history has not been used.
- The lease history trimming time will likely be adjusted. It is set to about two-thirds of the difference between the *lease-retention-max-age* and *lease-retention-min-age* values. For example, when the default value of six hours is taken, the trimming is done every 4 hours.
- Disk Input/Output rates go up on the system. This is because the server needs to update the older times in the active and historical lease records. The impact of this can be reduced to some extent by increasing the difference between the *lease-retention-max-age* and *lease-retention-min-age* values.
- When configuration changes such as removing scopes and prefixes, or adjusting ranges are made, the leases associated with the scopes or prefixes become orphaned leases. These orphaned leases are not trimmed and not processed for privacy protection time limits. You must remove the orphaned leases. For more information, see [Removing Orphaned Leases, on page 196](#).

DHCPv6 Clients and Leases

The DHCPv6 server supports clients and leases that are similar to those for DHCPv4. The key differences are:

- The server identifies DHCPv6 clients by their DHCP Unique Identifier (DUID), which is the DHCPv4 concept of hardware addresses and client IDs consolidated into one unique client identifier.
- DHCPv6 clients can have multiple leases. This means that if multiple prefixes are on a single link and are not grouped using the *allocation-group* attribute, the server assigns the client a lease from each prefix that it is allowed to use, not just from one scope, as in DHCPv4. If multiple prefixes on a single link are grouped using the *allocation-group* attribute, then the server assigns the client only one lease per allocation group from the prefix with highest priority within the prefix allocation group (see [Prefix Allocation Groups, on page 130](#)).
- The server first creates a DHCPv6 client when it associates the first lease with it, and deletes the client when it no longer has any leases associated with it. This is identical to DHCPv4 behavior, except that a DHCPv4 client can only have a single lease.
- DHCPv6 leases are dynamically created. The server does not create all leases that it can potentially use at configuration time, because there potentially could be billions of these leases.

Leases can be for:

- **Nontemporary addresses**—Standard IPv6 unicast addresses with likely long (and renewable) lifetimes.
- **Temporary addresses**—Standard IPv6 unicast addresses, but with very limited (and nonrenewable) lifetimes. Temporary addresses solve a privacy issue with IPv6 (see RFC 3041).
- **Delegated prefixes**—Used for prefix delegation (see RFC 3633).

Leases have both a preferred and valid lifetime:

- **Preferred lifetime**—Primarily for the use of the client, the length of time that a valid address is preferred. When the preferred lifetime expires, the address becomes deprecated.
- **Valid lifetime**—Used by both client and server, it is the length of time an address remains in the valid state. The valid lifetime must be greater than or equal to the preferred lifetime. When the valid lifetime expires, the address becomes invalid. A lease is eligible to be deleted once the valid lifetime expires. This is essentially the same as the DHCPv4 lease time.

Related Topics

[DHCPv6 Bindings, on page 189](#)

[Lease Affinity, on page 190](#)

[Lease Life Cycle, on page 190](#)

[DHCPv6 Lease Reservations, on page 204](#)

DHCPv6 Bindings

Bindings are new to DHCPv6 and allow multiple groups of addresses to be allocated to a client. A client binding consists of one of three types:

- Nontemporary (IA_NA)
- Temporary (IA_TA)

- Prefix delegation (IA_PD)

A binding also consists of a unique Identity Association Identifier (IAID). Leases always exist under a binding. Clients, therefore, have one or more bindings, and bindings have one or more leases. The server creates bindings when it first adds the lease, and removes the binding when it has no more leases. The server creates clients when adding the first binding, and removes them when it has no more bindings.

Lease Affinity

For DHCPv4, when a lease expires or the server releases it, the server remembers the client for an address as long as it is not assigned to another client. For DHCPv6, because of the large IPv6 address space and depending on the address generation technique, eons could pass before an address needs reassignment to another client. Therefore, Cisco Prime Network Registrar provides an *affinity-period* attribute so that the client can get the same address even if not requesting a renewal before expiration.

The affinity period is desirable in some environments, but not in others where the affinity time would be zero or very small. During the affinity period, the lease is in the AVAILABLE state and still associated with the client that last leased it. If the client requests a lease during this period, the server grants it the same lease (or, if renewals are inhibited, the client explicitly does not get that lease).

Lease Life Cycle

Leases have a life cycle controlled by states. A lease only exists while it is associated with a client and the server deletes it once it is no longer associated with that client. The life cycle and state transitions are:

1. A lease is born and associated with an address when the server:
 1. Creates a reservation for a lease, which puts the lease in the AVAILABLE state and marks it as RESERVED. No timer is associated with this state and the server does not delete the lease as long as it is RESERVED.
 2. Sends an ADVERTISE message to a client, which puts the lease in OFFERED state. The lease transitions to DELETED state after the offer timeout.
 3. Sends a REPLY message to a client (for a REQUEST, RENEW, or REBIND), which puts the lease in LEASED state. The lease transitions to EXPIRED state after the valid lifetime for the lease elapses.
2. An OFFERED lease transitions to:
 1. LEASED state when the server receives a REQUEST message, and then transitions to EXPIRED state after the valid lifetime for the lease elapses.
 2. DELETED state if the offered-time expires.
3. A LEASED lease:
 1. Is renewed when the server receives a REQUEST, RENEW, or REBIND message. The lease transitions to EXPIRED state after the new valid lifetime for the lease elapses (note that the new valid lifetime could be 0).
 2. Transitions to RELEASED state when the server receives a RELEASE message. The lease transitions to AVAILABLE state after the release-grace-period elapses.
 3. Transitions to UNAVAILABLE state when the server receives a DECLINE message. The server deletes the lease after the unavailable timeout period elapses.
4. An EXPIRED lease transitions to either AVAILABLE state after the grace-period. The server deletes the lease after the affinity-period elapses.

5. An AVAILABLE lease:
 1. Transitions to DELETE state and the server deletes it from memory and the lease database after the affinity-period elapses.
 2. Cannot be deleted if it is RESERVED, and it remains AVAILABLE.
6. The server can reoffer a LEASED, EXPIRED, RELEASED, or AVAILABLE lease to a client, but it remains in its current state, although the server extends the timeout to at least the *offer-timeout*.

The DHCP failover complicates some of the state transitions as these transitions can generally not occur until the partner acknowledges them. The additional life cycle and state transitions (failover related) are as follows:

- Transitioning into the AVAILABLE (or OTHER AVAILABLE) state requires that the partner to acknowledge the transition and hence the PENDING AVAILABLE state is used until the acknowledgement is received from the partner.
- Disassociating a lease from a client also requires an acknowledgement from the partner and hence the PENDING DELETE state is used until the partner has acknowledged the state change.

Configuring Leases in Scopes

After setting the IP address ranges for a scope, you can monitor and adjust the leases that result from DHCP assignments.

Viewing Leases

To view leases, you must first create a range of IP addresses for them in a scope, as described in the “Set Up DHCP” chapter of the *Cisco Prime Network Registrar 9.0 Quick Start Guide* or the [Managing Scopes, on page 111](#), then wait for the DHCP server to generate leases based on these addresses.

Local Basic Web UI

From the **Design** menu, choose **Scopes** under the **DHCPv4** submenu to open the List/Add DHCP Scopes page, then click the **Leases** tab for the scope. This opens the page, where you can click each lease to manage it.

See [Lease States, on page 185](#) for a description of the values in the State column. For guidelines as to the lease expiration time, see [Guidelines for Lease Times, on page 186](#).

To open the Manage DHCP Lease page, click the lease IP address.

Local Advanced Web UI

From the **Design** menu, choose **Scopes** under the **DHCPv4** submenu to open the List/Add DHCP Scopes page. You can then click the **Leases** tab for the scope; or you can click the name of the scope to open the Edit DHCP Scope page, then click **Leases** tab in the page.

CLI Commands

Use **lease** *[vpn-name]/ipaddr* **show** to show the properties of a particular lease based on its IP address. Use **scope name listLeases** to show all the leases for a named scope. The output is nearly identical for both commands. Note that you cannot list leases in a particular virtual private network (VPN); all the leases in all the VPNs appear in the list.

You can show the most recent MAC address associated with a lease or what lease is associated with a MAC address. The **lease** *[vpn-name]/addr macaddr* **macaddr** command shows the MAC address of the lease, whether or not that lease is reserved or active. The **lease list -macaddr** *addr [-vpn=vpn-name]* command lists the lease data only if the IP address for that MAC address was actively leased (and not reserved). You can also list leases by LAN segment and subnet by using **lease list -lansegment** *addr mask* and **lease list -subnet** *addr mask* commands.

Importing and Exporting Lease Data

You can use the CLI to import lease data to, and export from, text files.

Import Prerequisites

Before you can import leases, you must perform several configuration steps:

1. Configure a scope or scopes in the DHCP server for the leases that you plan to import.
2. If you want the hostnames for the leases dynamically entered into DNS as part of the import, configure zones in the DNS server to allow dynamic updates from the DHCP server.
3. Set the DHCP server to import mode so that it does not respond to other lease requests during the lease importing.
4. For all the time fields, use either the number of seconds since midnight GMT January 1, 1970, or a day, month, date, time, year format (Mon Apr 15 16:35:48 2002).
5. After you import the leases, take the DHCP server out of import mode so that it can respond to other lease requests.



Note Importing permanent leases will fail if you disable the permanent leases option. Enable this option using **policy name enable permanent-leases**, as necessary.

Import and Export Commands

The **import leases** and **export leases** commands use a special file format. Each record, or line, in the file represents one DHCP client:

```
field-1|field-2|field-3|...|field-13
```

Do not use spaces between the vertical line (|) delimiter and the field values. You must include at least the first four required fields. If you include more, you must delimit all the remaining null fields with the vertical line (|) so that there are 13 fields. The fields are, in order:

1. MAC address in *aa:bb:cc:dd:ee:ff* format (required)
2. MAC address type (required)
3. MAC address length (required)

4. IP address in dotted decimal format, *a.b.c.d* (required)
5. Start of lease time (Greenwich Mean Time, GMT) (optional)
6. Lease expiration time (GMT) (optional)
7. Allowable extension time (GMT) (optional)
8. Last transaction time (GMT) (optional)
9. IP address of the DHCP server (optional)
10. Hostname (without domain) (optional)
11. Domain name (optional)
12. Client ID (optional)
13. VPN name (optional; if omitted, the global VPN is used)

For all the time fields, use either the number of seconds since 1970, or the *day-month-date-time-year* format (such as Mon Apr 9 16:35:48 2007).

When importing leases, the DHCP server might not accept a lease, or a communication failure might drop the lease packet. In the latter case, the server retries the import several times, and after about a minute, reports a failure. If the import fails, check the DHCP server log file to find the lease that caused the error. Then go back to the import file, delete all lease entries up to and including the offending one, and repeat the lease import.

When you use **export leases**, you can choose between writing the state of all current and expired leases, or just the current leases, to the output file. The example below shows part of a lease data export from a Cisco Prime Network Registrar DHCP server. The blank lines between records appear in the example for clarity; they are not in the actual output.

Example: Lease Data Export

```
00:60:97:40:c1:96|1|6|204.253.96.103|Wed Aug 30 08:36:57 2000|Fri Sep 01 13:34:05 2000|
Wed Aug 30 08:36:57 2000|Fri Sep 01 09:34:05 2000|204.253.96.57|nomad|cisco.com|
00:d0:ba:d3:bd:3b|blue-vpn
00:d0:ba:d3:bd:3b|1|6|204.253.96.77|Thu Aug 17 13:10:11 2000|Fri Sep 01 14:24:46 2000|
Thu Aug 17 13:10:11 2000|Fri Sep 01 10:09:46 2000|
204.253.96.57|NPI9F6AF8|cisco.com|blue-vpn
00:d0:ba:d3:bd:3b|1|6|204.253.96.78|Fri Jun 23 15:02:18 2000|Fri Sep 01 14:11:40 2000|
Fri Jun 23 15:02:18 2000|Fri Sep 01 09:56:40 2000|
204.253.96.57|JTB-LOCAL|cisco.com|blue-vpn
```

Lease Times in Import Files

For a lease import request, if the DHCP server is:

- Enabled for *import-mode* and the lease is not already leased to the client, the server accepts any lease time the client specifies.
- Enabled for *import-mode*, the lease is already leased to the client, *defer-lease-extensions* is enabled for the server (the default), and the request arrives before the renewal time (T1), the server uses the existing lease time.

If the request arrives after T1, the server gives the client whatever it asks for. Within about two minutes of the expiration time, *defer-lease-extensions* is inoperative.

- Not enabled for *import-mode*, it never accepts a lease time longer than the server-configured one.
 - If *allow-lease-time-override* is enabled for a policy applicable to the request, the server accepts a shorter lease time from the client. The shorter lease time is acceptable to the server, even though

you can set a server expert mode *client-requested- min-lease-time* attribute that creates a floor for the lease time.

- If *allow-lease-time-override* is not enabled for any applicable policy, the server ignores the *dhcp-lease-time* request in the incoming packet and uses the server setting.

If your import file specifies a DNS zone name, the server does not use the zone name when it updates the DNS. If the file specifies a hostname, then the server uses the hostname when updating the DNS, unless hostname specification in a client or client-class entry overrides the hostname.

The client hostname should be in a zone other than the zone associated with the DNS update configuration object used for the DNS update. This can be indicated to the DHCP server, only by specifying that zone in a client or client-class entry.

Pinging Hosts Before Offering Addresses

You can have the DHCP server use the Internet Control Message Protocol (ICMP) echo message capability (also known as **ping**) to see if anyone responds to an IP address, before assigning it (using the *ping-clients* attribute). The *ping-clients* attribute controls whether the server should attempt to ping an address before offering a lease. If enabled, then the *ping-timeout* attribute may also need to be set. This test allows the DHCP server to check whether an address is not in use before assigning it.

Using **ping** can help prevent two clients from using the same address. If a client responds to ping, the DHCP server marks that address as *unavailable* and offers a different address. This test works only for powered-up clients; it is possible for clients to have a lease and be powered down.

You can also configure the *ping-clients* attribute at the DHCP server. This attribute controls the default value of the *ping-clients* attribute of a scope, if not explicitly configured on a scope.



Note

If you have configured scopes, the scope-specific configuration takes precedence; scopes without explicit configurations assume the global setting.

The ping timeout period is important. Because pinging helps to ensure that no client is using a particular IP address, each ping must wait the entire timeout period. This ping timeout period comes before an offer, so the time specified has a considerable effect on server performance.

- If you set this time too long, it slows down the lease offering process.
- If you set this time too short, it reduces the effectiveness of the ping packet to detect another client using the IP address.

To implement pinging hosts before offering IP addresses, modify the scope by:

- Enabling the *ping-clients* attribute. It is disabled by default.
- Setting the *ping-timeout* attribute. It is 300 milliseconds by default.

The server makes unavailable any IP address for which it receives a successful ECHO reply. You can control this action by enabling the DHCP server attribute *ignore-icmp-errors* (the preset value). If disabled, the DHCP server also uses ICMP DEST_UNREACHABLE and TTL_EXPIRED error messages that it receives after sending ICMP ECHO requests as reasons for making an IP address unavailable.

Deactivating Leases

Deactivating a lease moves a client off of it. If the lease is available, deactivating it prevents the DHCP server from giving it to a client. If the lease is active (held by a client), deactivating it prevents the client from renewing it and the server from giving the lease to another client. You can deactivate a lease only if the server is running. The DHCP server deactivates the lease immediately.



Tip To force a Windows client to release its lease, run **ipconfig /release** on the client machine.



Note For DHCPv4 leases, the lease will remain deactivated until it is reactivated. For DHCPv6 leases (address or prefix delegation), the behavior is a bit different in that the lease will automatically be activated when the client is removed from the lease. Therefore, there is no need to activate DHCPv6 deactivated leases. However, this also means that the lease is available after the current lease ends and you cannot deactivate leases that are not associated with a client. If a DHCPv6 reservation is deactivated, it must be activated to be used by that client again.

Local Basic or Advance Web UI

To deactivate a lease, click the address of the lease on the **Leases** tab for the Scope (see [Viewing Leases, on page 191](#)) and click **Deactivate**. The lease now shows as deactivated. To reactivate the lease, click **Activate**. You can also deactivate DHCPv6 leases in a similar manner.

CLI Commands

To deactivate a lease, use **lease [vpn-name/]ipaddr deactivate**. To reactivate a lease, use **lease [vpn-name/]ipaddr activate**.

To deactivate a DHCPv6 lease, use **lease6 [vpn-name/]addr deactivate**. To reactivate a DHCPv6 lease, use **lease6 [vpn-name/]addr activate** (though see the note above as DHCPv6 leases generally need not be re-activated as this happens automatically when the client is removed from the lease).

Excluding Leases from Ranges

IP address ranges, by definition, must be contiguous. To exclude a lease from an existing range, you must divide the range into two smaller ones. The new ranges consist of the addresses between the original starting and ending range addresses and the address that you want to exclude.



Caution If the excluded address currently has an active lease, you should first follow the steps in [Deactivating Leases, on page 195](#), otherwise you will get a warning message. Deleting an active lease can result in a duplicate IP address if the deleted address is subsequently reconfigured and then reassigned. Information about that lease will no longer exist after you reload the server.

Local Basic Web UI

To exclude a lease from a scope address range, do the following:

-
- Step 1** From the **Design** menu, choose **Scopes** under the **DHCPv4** submenu to open the List/Add DHCP Scopes (Address Pool) page.
 - Step 2** Click the name of the scope in the Scopes pane to open the Edit DHCP Scope (Address Pool) page.
 - Step 3** In the Ranges area, click the **Delete** icon next to the IP address range you want to remove.
 - Step 4** Add a range that ends just before the excluded IP address.
 - Step 5** Add another range that begins just after the excluded IP address.
 - Step 6** Save the scope.
 - Step 7** Reload the DHCP server.
-

Local Advanced Web UI

To exclude a lease from a scope address range, the same operations exist as in Basic mode, except that you click the name of the scope on the List/Add DHCP Scopes page, which opens the Edit DHCP Scope page.

CLI Commands

To exclude a lease from a scope address range, discover the lease range (**scope name listRanges**), deactivate the lease (**lease [vpn-name/]ipaddr deactivate**), then remove the range of just that IP address (**scope name removeRange start end**). The resulting ranges are then split appropriately.

The following example removes the 192.168.1.55 address from the range. Note that if the lease is in a scope with a defined VPN, you must explicitly define that VPN for the session, or you can include the VPN prefix in the **lease** command:

```
nrcmd> session set current-vpn=red
nrcmd> scope examplescope1 listRanges
nrcmd> lease red/192.168.1.55 deactivate
nrcmd> scope examplescope1 removeRange 192.168.1.55 192.168.1.55
nrcmd> scope examplescope1 listRanges
```

Removing Orphaned Leases

To remove the orphaned leases:

Before you begin

When configuration changes such as removing scopes and prefixes, or adjusting ranges are made, the leases associated with the scopes or prefixes become orphaned leases. These orphaned leases are not updated periodically to assure that they do not violate the date restrictions.

When you use the lease date restriction feature, ensure that no orphaned-leases are present (or clean them out periodically).

-
- Step 1** Enable the DHCP attribute delete-orphaned-leases:

```
nrcmd> dhcp enable delete-orphaned-leases
```

Step 2 Reload the DHCP server:

```
nrcmd> dhcp reload
```

Step 3 Unset the DHCP attribute delete-orphaned-leases:

```
nrcmd> dhcp unset delete-orphaned-leases
```

Step 4 Reload the DHCP server:

```
nrcmd> dhcp reload
```

Searching Server-Wide for Leases

Using Cisco Prime Network Registrar, you can search for leases, server-wide. The search is a filter mechanism whereby you can specify a combination of lease attributes to target one or more leases configured for the network. The lease history search function is available at both local and regional cluster whereas the active lease search function is available only at the local cluster. The search function is provided separately for DHCPv4 and DHCPv6 leases.

You can also search for the active leases using Cisco Prime Network Registrar.

Local Advanced Web UI

To search for DHCPv4 leases, do the following:

Step 1 From the **Operate** menu, choose **DHCPv4 Current Leases** under the **Reports** submenu to open the DHCP Lease Search page.

Note You can open the DHCP Lease Search page by clicking the Search button in the DHCP Lease History Search page (choose **DHCPv4 Lease History** under the **Reports** submenu to open the DHCP Lease History Search page). This button helps you to toggle between lease history search page and active leases search page.

Step 2 Choose a Filter Attribute from the drop-down list, such as address. DHCPv4 and DHCPv6 have separate lists of filter attributes. Also, the set of filter attributes are different for active and historical leases.

Attributes are greyed out after you select them as elements.

Step 3 Choose a filter Type from the drop-down list. You can choose at least Binary or Regular Expression, but the list can contain one or more of the following, depending on the Filter Attribute selected:

- Binary—Value is in binary notation.
- Date Range—Range of date values, From a date and time To a date and time.
- Integer—Value is an integer.
- Integer Range—Integer From value to an integer To value.
- IP Address—Value is an IP address.
- IP Range—IP address From value to an IP address To value.

- IP Subnet—Value is an IP subnet.
- Regular Expression—Value is a Regular Expression in regex syntax. (For common regex usage, see the "Configuring Administrators" chapter in *Cisco Prime Network Registrar 9.0 Administrator Guide*.)

Step 4 Enter a Value, based on the Type selected. To clear the filter, click **Clear Filter**.

Step 5 Click **Add Element** to add the search element to the Filter Elements list. You can delete the element by expanding the filter display, then clicking the **Delete** icon next to the element.

Step 6 Once you assemble a list of elements, you can search on them, so that the elements are ANDed together for the result. Click **Search**.

Step 7 Check the table of resulting leases from the search, which shows for each an address, state, MAC address, hostname, flags, and expiration date. If necessary, change the page size to see more entries. The leases are ordered by IP address.

Tip The filter elements are ANDed together for the search. If you find that the search results do not yield what you expect, look at the Filter Elements list again and delete elements that can obstruct the results.

Local Advanced Web UI

To search for DHCPv6 leases, do the following:

Step 1 From the **Operate** menu, choose **DHCPv6 Current Leases** under the **Reports** submenu to open the DHCP v6 Lease Search page.

You can also go to the DHCP v6 Lease Search page if you choose **DHCPv6 Lease History** under the **Reports** submenu. If you choose **DHCPv6 Lease History** under the **Reports** submenu, the DHCP v6 Lease History Search page is displayed. You have to click the Search button to go to the DHCP v6 Lease Search page.

Step 2 Choose a Filter Attribute from the drop-down list, such as address.

Step 3 Choose a filter Type from the drop-down list. You can choose at least Binary or Regular Expression, but the list can contain one or more of the following, depending on the Filter Attribute selected:

- Binary—Value is in binary notation.
- Date Range—Range of date values, From a date and time To a date and time.
- Integer—Value is an integer.
- Integer Range—Integer From value to an integer To value.
- IPv6 Address—Value is an IPv6 address.
- IPv6 Prefix—Value is an IPv6 prefix.
- Regular Expression—Value is a Regular Expression in regex syntax. (For common regex usage, see the "Configuring Administrators" chapter in *Cisco Prime Network Registrar 9.0 Administrator Guide*.)
- Contains—Value is an IPv6 address or prefix (available for only IPv6 address). The query will list the leases that contain the specified address or prefix.

Step 4 Enter a Value, based on the Type selected. To clear the filter, click **Clear Filter**.

Step 5 Click **Add Element** to add the search element to the Filter Elements list. You can delete the element by expanding the filter display, then clicking the **Delete** icon next to the element.

- Step 6** Once you assemble a list of elements, you can search on them, so that the elements are ANDed together for the result. Click **Search**.
- Step 7** Check the table of resulting leases from the search, which shows for each an address, state, MAC address, hostname, flags, and expiration date. If necessary, change the page size to see more entries. The leases are ordered by IP address.

CLI Commands

Use `lease list -macaddr mac-addr [-vpn=vpn-name]` to find leases in the DHCPv4 space. Specify the MAC address of the lease. If you omit the VPN designation, you base the search on the current VPN.

For leases in the DHCPv4 space, use the following `lease list` syntax:

```
nrcmd> lease list [-macaddr=mac-addr] [-cm-macaddr=cm-mac-addr]
[-reservation-lookup-key=key] [-mac | -blob | -string]]
[-vpn=vpn-name] [-count-only]
```

For leases in the DHCPv4 space, use the following `lease listbrief` syntax:

```
nrcmd> lease listbrief [-macaddr=mac-addr] [-cm-macaddr=cm-mac-addr]
[-reservation-lookup-key=key] [-mac | -blob | -string]]
[-vpn=vpn-name] [-count-only]
```

For leases in the DHCPv6 space, use the following `lease6 list` syntax:

```
nrcmd> lease6 list[-duid= client-id]
[-lookup-key=key] [-blob | -string]]
[-reservation-lookup-key=key] [-blob | -string]]
[-macaddr=mac-addr]
[-cm-macaddr=cm-mac-addr]
[-vpn=vpn-name] [-count-only]
```

For leases in the DHCPv6 space, use the following `lease6 listbrief` syntax:

```
nrcmd> lease6 listbrief[-duid= client-id]
[-lookup-key=key] [-blob | -string]]
[-reservation-lookup-key=key] [-blob | -string]]
[-macaddr=mac-addr]
[-cm-macaddr=cm-mac-addr]
[-vpn=vpn-name] [-count-only]
```

The `-macaddr` and `-cm-macaddr` options are to search for leases identified by the CableLabs DOCSIS `vendor-opts` option (DHCPv6 option 17). For example, for these two commands:

```
nrcmd> lease6 listbrief -macaddr=01:02:03:04:05:06
nrcmd> lease6 listbrief -cm-macaddr=01:02:03:04:05:06
```

The `-macaddr` line lists leases where the option 17 device-id suboption (36) contains the requested MAC address. The `-cm-macaddr` line lists leases where the option 17 cm-mac-address suboption (1026) matches the requested MAC address. (See [DHCPv6 Options, on page 420](#) for details on these suboptions.)

Using Client Reservations

In earlier versions of Cisco Prime Network Registrar versions, the only option for clients to get the lease they want was to create a lease reservation (see [Creating Lease Reservations, on page 202](#)). It may not always be

easy to create reservations for each client, which may come up to millions of reservations. Also, the process to update and synchronize the Cisco Prime Network Registrar reservations with databases is very complex. The client reservation feature helps in reducing this complexity.

The current functionality supported by Cisco Prime Network Registrar DHCP server in assigning an IP address to a DHCPv4 client is as follows:

- If a lease based reservation for the client exists and the lease is available, it is used.
- Otherwise, if the client requested an address and it is available, it is used.
- Otherwise, a random address from one of the scopes available to the client is used.

Client reservations feature enables you to supply addresses and delegate prefixes through client entries (either stored directly in Cisco Prime Network Registrar or in LDAP) or through extensions. Also, a client can be located on more than a single scope or prefix and the server will select the address appropriate to the location of the client.

Client-reserved leases are essentially reserved leases. The major difference is that the client for which the lease is reserved is not known to the server in case of client reservations. Client reservations are used when you want to configure leases for many clients or configure many leases for a single client.

Client reservations can be provided to Cisco Prime Network Registrar using one of the following three primary mechanisms:

- Using internal client database—This has some of the same issues as with lease reservations, but may be a better option if Cisco Prime Network Registrar internal client database is already being used for other purposes. The fact that the internal client database has to maintain the client alone and not the reservations makes it more advantageous when compared to lease reservations.
- Using LDAP—Cisco Prime Network Registrar can look up clients in an LDAP repository (external to Cisco Prime Network Registrar) and these clients may specify client reservations.
- Using extensions—Cisco Prime Network Registrar can be set up to communicate with external servers or databases using extensions.

The client entries, maintained either within the Cisco Prime Network Registrar client database or LDAP, can include the addresses and prefixes a client is supposed to use. The attributes to specify the client reservations are:

1. **reserved-addresses**—Specifies the list of addresses reserved for the client. The first available address to match a usable Scope (which must have restrict-to-reservations enabled) are assigned to the client.
2. **reserved-ip6addresses**—Specifies the list of addresses reserved for the client. All available addresses to match a usable Prefix (which must have restrict-to-reservations enabled) are assigned to the client.
3. **reserved-prefixes**—Specifies the list of prefixes reserved for the client. All available prefixes to match a usable Prefix (which must have restrict-to-reservations enabled) are assigned to the client.

The attribute restrict-to-reservations is added to Scope, Scope template, Prefix, and Prefix template objects to specify the client reservations.

For a client in LDAP, you must set up a mapping between the LDAP attribute name and the corresponding client attribute name.

If the LDAP addresses attribute contained a list of the IPv4 addresses for the client, use **ldap servername setEntry query-dictionary ldap-attribute=cnr-client-attribute** to map it to the reserved-addresses attribute. For example:

```
nrcmd> ldap ldap-1 setEntry query-dictionary addresses=reserved-addresses
```

Local Advanced Web UI

To restrict a scope to client reservations, do the following:

1. From the **Design** menu, choose **Scopes** under the **DHCPv4** submenu to open the List/Add DHCP Scopes page. See [Creating Scopes, on page 112](#) to create a scope.
2. Click **enabled** for *restrict-to-reservations* attribute in Miscellaneous Settings group in the List/Add DHCP Scopes page.

To modify an existing scope to specify client reservations, click the required scope name to open the Edit DHCP Scope page. Click **enabled** for *restrict-to-reservations* attribute in Miscellaneous Settings group.

The flag client-reserved shows that a scope is restricted to client reservations.



Note All the IP addresses configured in a scope which has the *restrict-to-reservations* attribute enabled will be counted in the active lease count, and will count against the licensed IP addresses for this DHCP server. If you are a heavy user of client reservations, you might want to configure only the addresses that you expect to need in the near term, and increase the size of the scope later if you need more addresses.

To restrict a scope template to client reservations, do the following:

1. From the **Design** menu, choose **Scope Templates** under the **DHCPv4** submenu to open the List/Add DHCP Scope Templates page. See [Creating and Applying Scope Templates, on page 139](#) to create a scope template.
2. Click **enabled** for *restrict-to-reservations* attribute in Miscellaneous Settings group in the List/Add DHCP Scope Template page.

To modify an existing scope template to specify client reservations, click the required scope template name to open the Edit DHCP Scope Template page. Click **enabled** for *restrict-to-reservations* attribute in Miscellaneous Settings group.

To restrict a prefix to client reservations, do the following:

1. From the **Design** menu, choose **Prefixes** under the **DHCPv6** submenu to open the List/Add DHCP v6 Prefixes page.
2. Click the **Add Prefixes** icon in the Prefixes pane, enter the prefix name and address and click the Add IPv6 Prefix.
3. Click the prefix name on the Prefixes pane to open the Edit DHCPv6 Prefix page. Click **enabled** for *restrict-to-reservations* attribute in Non-Parent Settings group.



Note Prefixes which have the *restrict-to-reservations* attribute enabled are not counted in the total of active leases which must be licensed. Any client which receives a client reservation will have that active lease counted, but that will happen only when the lease is actually held by a client.

To restrict a prefix template to client reservations, do the following

1. To restrict a prefix to client reservations, from the **Design** menu, choose **Prefix Templates** under the **DHCPv6** submenu to open the List/Add DHCP v6 Prefix Templates page.
2. Click the **Add Prefix Templates** icon in the Prefix Templates pane, to open the Add Prefix Template dialog box.
3. Enter the prefix template name and click the **Add Prefix Template** button.
4. Click **enabled** for *restrict-to-reservations* attribute.

To modify an existing prefix template to specify client reservations, click the prefix template name that you want to restrict to client reservations. Click **enabled** for *restrict-to-reservations* attribute.

Differences Between Client Reservations And Lease Reservations

Client reservations have the following significant differences over lease reservations:

- There is **no** validation to assure that there is only a single client reservation for any address. If there are two different clients that specify the same address or prefix, whichever client request arrives first is granted that lease.
- A client reservation really exists only after the client completes DHCP configuration. Lease reservations are known even if a client transaction never occurs and thus can also be used for clients that do not provide DHCP services at all.

Cisco Prime Network Registrar supports:

- Creating a lease reservation for a particular IP address.
- Configuring the correct cable modem MAC address for the IP address such that Cable Source Verify will work correctly with a Cable Modem Termination System (CMTS).

This works because the Cisco Prime Network Registrar DHCP server knows about the lease reservation before any DHCP client transaction and will respond correctly to a leasequery request from a CMTS for those addresses. Client reservations are, in contrast, not known to the DHCP server before the arrival of a DHCP client packet at the DHCP server. A leasequery for an IP address which is configured as client-reserved due to some client registration will not (in general) know that the IP address is client reserved.

Thus, any leasequery to which the DHCP server is supposed to respond with a positive result that includes the proper cable modem MAC address, even when no client has actively requested the lease, will not work with client reservations.

Creating Lease Reservations

To ensure that a client always gets the same lease, you can create a lease reservation. Managing lease reservations is available only to administrators having the `dhcp-admin` role at the local cluster, or the `central-cfg-admin` role with the `dhcp-management` subrole at the regional cluster.

You can query DHCPv4 and DHCPv6 reservations from the server.



Note All lease reservations are counted in the total of active leases that is compared to the number of IP addresses licensed.

DHCPv4 Reservations

When the DHCP edit mode is synchronous, reservation changes are automatically forwarded to the DHCP server, and take immediate effect.

When the edit mode is staged, any change you make to the reservation list on a local cluster modifies the parent scope to indicate that a server reload is required. Any change to the regional reservation list modifies the parent subnet.

Local Basic Web UI

To view lease reservations, from the **Design** menu, choose **Scopes** under the **DHCPv4** submenu to open the List/Add DHCP Scopes (Address Pools) page, then click the **Reservations** tab.

To create a reservation on this page, enter the IP address you want to reserve for lease, and enter a lookup key in the Lookup Key field. Click the MAC address (the default) or string or binary radio button, as appropriate for the lookup key entry. Click **Add Reservation**. The lease IP address, Lookup Key and Scope details are displayed in the List/Add DHCP Reservations page.

Local Advanced Web UI

To view the lease reservations for DHCPv4 scopes, from the **Design** menu, choose **Scopes** under the **DHCPv4** submenu to open the List/Add DHCP Scopes page. Proceed as for the Basic web UI.

Advanced mode also provides a mechanism to create reservations independent of scopes. To configure reservations directly for DHCPv4 scopes, do the following:

-
- Step 1** From the **Design** menu, choose **Reservations** to open the List/Add DHCP Reservations page.
- Step 2** Click the **Add DHCP Reservations** icon in the Reservations pane, enter the IP address you want to reserve for lease, and enter a lookup key in the Lookup Key field, then click **Add Reservation**.
- Step 3** Click the MAC address (the default) or string or binary radio button, as appropriate for the lookup key entry. Click **Save**.
- Tip** You can use a filter to reduce the size of the list that is displayed. To do this, choose a filter type from the Filter Type drop-down list. The Filter Value is set as for the selection of the Filter Type. Click **Set Filter**. To set Filter Type as None, click **Clear Filter**. The lease IP address, Lookup Key and Scope details are displayed in the List/Add DHCP Reservations page.
- Note** Multiple DHCP servers should not distribute IP addresses on the same subnet, unless they are DHCP Failover partners. When using Failover, the client reservations must be identical on each server. If not, a client for whom a lease reservation exists can receive offers of different IP addresses from different servers. The Failover synchronization function helps you assure that the partner configuration is consistent.
-

CLI Commands

The **reservation** command lets you access the global list of DHCPv4 reservations of Cisco Prime Network Registrar.

Create a new address by using, **reservation** [vpn-name/]address **create** {macaddr | lookup-key} [-mac | -blob | -string] [attribute=value ...]

For example:

```
nrcmd> reservation white/192.168.1.110 create 00:d0:ba:d3:bd:3b
```

Delete an address by using, **reservation** *[vpn-name/]address delete*

For example:

```
nrcmd> reservation white/192.168.1.110 delete
```

Get an attribute by using, **reservation** *[vpn-name/]address get attribute*

For example:

```
nrcmd> reservation white/192.168.1.110 get value
```

Set an attribute by using, **reservation** *[vpn-name/]address set attribute=value*

For example:

```
nrcmd> reservation white/192.168.1.110 prefix=cm_prefix
```

Unset an attribute by using, **reservation** *[vpn-name/]address unset attribute*

For example:

```
nrcmd> reservation white/192.168.1.110 unset value
```

Show an address by using, **reservation** *[vpn-name/]address show*

For example:

```
nrcmd> reservation white/192.168.1.110 show
```

Display the reservations by using, **reservation list** *[[vpn-name/]address [-mac | -key]*. This command displays the reservations in *address* order unless *-key* is specified to change the sort order.

For example:

```
nrcmd> reservation list white/192.168.1.110
```

Display the brief details of the reservations by using, **reservation listbrief** *[-macaddr=mac-addr] [-lookup-key=lookup-key [-mac | -blob | -string]] [-vpn=vpn-name] [-count-only]*.

For example:

```
nrcmd> reservation listbrief -lookup-key=d4:6a:a8:d3:e2:ea -mac
```

DHCPv6 Lease Reservations

Reservations apply to nontemporary addresses and delegated prefixes only. They are associated with a prefix in the configuration, and must always be for an address (or prefix) under a configured prefix object.

The reservation can be outside the object range of the prefix, provided it is not in object range of another prefix. However, this has implications when you add a new prefix object. If a reservation that is contained in the new range of the prefix exists, the prefix will not be added. This results in an EX_CONFLICT status. For details, see [Creating Lease Reservations, on page 202](#).



Note The operations for DHCPv4 reservations are similar to DHCPv6 reservations, except that the addresses are v6 addresses, not v4 addresses. Also, the main identity for a DHCPv6 client is a client DUID, and not the mac-address. DHCPv6 reservations include addresses and delegated prefixes.

Any change you make in the v6 reservation list modifies the parent prefix to indicate that a server reload is required. On the regional server, if the DHCP edit mode is synchronous and the parent prefix has been assigned to a local cluster, changes are automatically forwarded to the local cluster. A server reload is required, before these changes take effect.



Caution If multiple DHCP servers distribute IP addresses on the same prefix, the reservations must be identical. If not, a client for whom a reservation exists can receive offers of different IP addresses from different servers.

A lease reservation pairs an IP address with a lookup key. A lookup key can be a string value or binary blob.



Note If a new prefix delegation reservation is added that has a shorter or longer prefix that conflicts (is contained by or contained in) an existing lease when the server is reloaded, the reservation will prevent the existing leases from being loaded.

Local Advanced Web UI

To view the reservations for DHCPv6 prefixes, do the following:

-
- Step 1** To view the DHCPv6 lease reservations, from the **Design** menu, choose **Prefixes** under the **DHCPv6** submenu to open the List/Add DHCPv6 Prefixes page.
 - Step 2** Select the prefix on the Prefixes pane and click the **Reservations** tab.
-

Local Advanced Web UI

To configure the reservations directly for DHCPv6 prefixes, do the following:

In the advanced mode, if a valid parent prefix is not specified, the CCM server automatically sets the appropriate parent prefix.

-
- Step 1** From the **Design** menu, choose **Reservations** under the DHCPv6 submenu to open the List/Add DHCP v6 Reservations page.
 - Step 2** To create a reservation, click the **Add DHCP v6 Reservations** icon in the Reservations pane, enter the IP address you want to reserve for lease, and enter a lookup key in the Lookup Key field.
 - Step 3** Click the String radio button, if you entered string value or click the Binary radio button, if you entered binary value in the Lookup Key field.
 - Step 4** Click **Add Reservation**.

Step 5 On the Reservations pane, choose a filter type from the Filter Type drop-down list. Enter a value in the Filter Value field. Click **Set Filter**. To set Filter Type as None, click **Clear Filter**. The lease IP address, Lookup Key and Prefix details are displayed in the List/Add DHCP v6 Reservations page.

CLI Commands

The **reservation6** command lets you access the global list of DHCPv6 reservations of Cisco Prime Network Registrar.

A matching prefix must exist for each reservation in the global list, otherwise the edit is rejected as invalid.

Create a new address by using, **reservation6** *[vpn-name/]address* **create** *lookup-key* **[-blob | -string]** *[attribute=value ...]*

For example:

```
nrcmd> reservation6 white/2001:db8::1 create 00:03:00:01:01:02:03:04:05:06
```

Delete an address by using, **reservation6** *[vpn-name/]address* **delete**

For example:

```
nrcmd> reservation6 white/2001:DB8::1 delete
```

Get an attribute by using, **reservation6** *[vpn-name/]address* **get** *attribute*

For example:

```
nrcmd> reservation6 white/2001:DB8::1 get value
```

Set an attribute by using, **reservation6** *[vpn-name/]address* **set** *attribute=value*

For example:

```
nrcmd> reservation6 white/2001:DB8::1 set prefix=cm_prefix
```

Unset an attribute by using, **reservation6** *[vpn-name/]address* **unset** *attribute*

For example:

```
nrcmd> reservation6 white/2001:DB8::1 unset value
```

Show an address by using, **reservation6** *[vpn-name/]address* **show**

For example:

```
nrcmd> reservation6 white/2001:DB8::1 show
```

Display the reservations by using, **reservation6 list** *[[vpn-name/]address | -key]*. This command displays the reservations in *address* order unless *-key* is specified to change the sort order.

For example:

```
nrcmd> reservation6 list white/2001:DB8::1
```

Display the brief details of the reservations by using, **reservation6 listbrief** **[-lookup-key=lookup-key [-blob | -string]] [-vpn=vpn-name] [-count-only]**.

For example:

```
nrcmd> reservation6 listbrief -lookup-key=def -string -vpn=vpn1
```

Setting Advanced Lease and Reservation Properties

Setting advanced lease and reservation properties can include:

- Reserving currently leased IP addresses—See [Reserving Currently Leased Addresses, on page 207](#)
- Unreserving leases—See [Unreserving Leases, on page 208](#)
- Extending leases to non-MAC addresses—See [Extending Reservations to Non-MAC Addresses, on page 209](#)
- Forcing lease availability—See [Forcing Lease Availability, on page 211](#)
- Inhibiting lease renewals—See [Inhibiting Lease Renewals, on page 212](#)
- Handling leases marked as unavailable—See [Handling Leases Marked as Unavailable, on page 214](#)
- Setting timeouts for unavailable leases—See [Setting Timeouts for Unavailable Leases, on page 215](#)

Reserving Currently Leased Addresses

You can delete a reservation for one client while reusing it for another one, even though the first client still has the lease.

Local Advanced Web UI

To reserve an existing lease, do the following:

-
- Step 1** From the **Design** menu, choose **Scopes** under the **DHCPv4** submenu, then select the name of the scope to open the Edit DHCP Scope page.
 - Step 2** Click the **Leases** tab.
 - Step 3** Click the IP address of the lease.
 - Step 4** If the IP address is not leased (in available state), enter the lookup key or MAC address for the reservation.
 - Step 5** Click **Make Reservation**. On the Edit DHCP Scope page, the lease will appear as reserved.
 - Step 6** Save the scope.
 - Step 7** To remove the reservation, click **Remove Reservation** on the Manage DHCP Lease page, then modify the scope. The lease no longer appears as reserved.
-

Example of Reserving an Existing Lease

This CLI command example creates a reservation from an existing lease. It assumes that the `dhcp-edit-mode` has been set to `synchronous` to allow the reservations to be added to the server dynamically:

```
nrcmd> reservation 192.168.1.110 create 1,6,00:d0:ba:d3:bd:3b
nrcmd> lease 192.168.1.110 activate
```

Client 1,6,00:d0:ba:d3:bd:3b does a DHCPDISCOVER and gets an offer for 192.168.96.110. The client then does a DHCPREQUEST and gets an ACK message for the same IP address.

As time passes, client 1,6,00:d0:ba:d3:bd:3b does several DHCPREQUESTs that are renewals, which the server acknowledges. Then, at some time before the client lease expiration time, you terminate the reservation:

```
nrcmd> lease 192.168.1.110 deactivate
nrcmd> reservation 192.168.1.110 delete
```

You then add a reservation for a different client for that IP address, even though the address is still leased to the first client:

```
nrcmd> reservation 192.168.1.110 create 1,6,02:01:02:01:02:01
nrcmd> lease 192.168.1.110 activate
```

This action results in an IP address that is leased to one client, but reserved for another. If the new client (1,6,02:01:02:01:02:01) does a DHCPDISCOVER before the original client (1,6,00:d0:ba:d3:bd:3b) does, the new client does not get 192.168.96.110, but gets a random IP address from the dynamic pool.

When the original client (1,6,00:d0:ba:d3:bd:3b) sends its next DHCPREQUEST/RENEW for the lease on 192.168.96.110, it gets a NAK message. Generally, upon receipt of the not-acknowledged message, the client immediately sends a DHCPDISCOVER. On receiving that DHCPDISCOVER, the server cancels the remaining lease time for 192.168.96.110.

The server then gives client 1,6,00:d0:ba:d3:bd:3b whatever lease is appropriate for it—some reservation other than 192.168.96.110, some dynamic lease (if one is available), or nothing (if no dynamic leases are available). When the new client (1,6,02:01:02:01:02:01) tries to renew the random IP address it received, the server sends it a NAK, because it wants to give it the reserved address. When the new client then does a DHCPDISCOVER, it gets the 192.168.96.110 reserved address.

You could also force availability of a lease (see [Forcing Lease Availability, on page 211](#)). However, doing so does not stop the original client (1,6,00:d0:ba:d3:bd:3b) from using 192.168.96.110. Also, it does not prevent the new client (1,6,02:01:02:01:02:01) from getting 192.168.96.110. In other words, this means that making a reservation for a client is independent of the lease state (and actual lease client) of the IP address for which the reservation is made.

Thus, making a reservation for one client does not cause another client to lose that lease right away, although that client receives a NAK response the next time it contacts the DHCP server (which could be seconds or days). Additionally, the client that reserved the IP address does not get that address if some other client already has it. Instead, it gets another IP address until the:

- IP address it is supposed to receive is free.
- Client sends a DHCPREQUEST as a renewal and receives a NAK response.
- Client sends a DHCPDISCOVER.

Unreserving Leases

You can remove lease reservations at any time. However, if the lease is still active, the client continues to use the lease until it expires. If you try to reserve the lease for a different client, you will get a warning.

Once you delete the last reservation from regional, it is not possible to select the reservation and push the change to the local cluster. You must push the parent subnet, which will then synchronize the reservation list and thus delete the local copy of the reservation.

There is no push function for DHCPv6 reservations in regional. You always need to push the parent prefix to resynchronize the reservations. This is the preferred method when synchronizing regional delete actions.

Local Advanced Web UI

To unreserve a lease, choose **Reservations** from the **Design** menu to open the List/Add DHCP Reservations page, then click the **Delete Reservation** icon after selecting the reservation you want to remove. This removes the reservation immediately, with no confirmation.

CLI Commands

To unreserve a lease, use **reservation** *[vpn/]ipaddr delete* or **scope name removeReservation** *{ipaddr | macaddr | lookupkey} [-mac | -blob | -string]*. However:

- Ensure that the reservation is gone from the nrcmd internal database.
- If you use failover on the scope containing the reservation:
 1. Use **reservation** *[vpn/]ipaddr delete*, or **scope name removeReservation**, on both servers.
 2. On the backup server, if you are in staged dhcp edit mode, use **lease** *[vpn/]ipaddr delete-reservation*.
 3. Use the same command on the main server.

Save the result of this operation to preserve it across server reloads, because issuing **lease ipaddr delete-reservation** alone affects only the server internal memory.

Extending Reservations to Non-MAC Addresses

You might need to create lease reservations based on something other than the MAC address from the incoming client packet. Often, DHCP client devices attached to a switch port need to get the same IP address, regardless of the MAC address. This approach helps when you replace factory floor devices with identical devices (with different MAC addresses), but want to maintain the same IP address.

Overriding Client IDs

You can set an expression in a client-class *override-client-id* attribute that extracts the MAC address and port of a switch from the relay-agent-info option (82) and creates a client identity from it. Regardless of the client-id in the incoming packet, the identity that allocates an IP address is the same for any device coming in through the same switch port. The expression you use for the attribute depends on the option 82 format. The DHCP server calculates the expression when it assigns the packet to the client-class. The *override-client-id* value becomes the identity of the client from that point onward.



Note When using *[v6-]override-client-id* expressions, leasequery by client-id requests may need to specify the *override-client-id* attribute to correctly retrieve the information on the lease(s) for the client.

However, when you enable the *use-client-id-for-reservations* attribute in a policy, the server turns the client-id of that request into a string of the form *nn:nn:nn ... nn:nn*, and uses that string to look up the reservation.

The *add-to-environment-dictionary* attribute for a client or client-class also serves to send attribute values to the DHCP extension environment dictionary (see [Using Extension Points, on page 351](#)), specified as name-value pairs. You can configure an *add-to-environment-dictionary* attribute on either a client or a client-class. If you choose to configure this attribute on both a client and client-class, you should ensure that the name-value pairs that you configure on the client have different names than the name-value pairs that you configure on the client-class, because they are all going to be put into the same environment dictionary (which can have only one value for a particular name). Generally, it is best to configure this attribute on a client or a client-class only, but not on both.

Local Advanced Web UI

You can find the *override-client-id* attribute on the Edit DHCP Client Class page (from the **Design** menu, choose **Client Classes** under **DHCP Settings** submenu, then the name of the client-class).

You also need to configure a client-class lookup ID for the DHCP server, to put every packet into a particular client-class where you configure the *override-client-id* expression. From the **Operate** menu, choose **Manager Servers** under the **Servers** submenu, then the Local DHCP Server link to open the Edit Local DHCP Server page. In the Client Class attributes, enter a *client-class-lookup-id* expression.

To use the client ID for the reservation, configure the policy to enable the *use-client-id-for-reservations* attribute on the Add DHCP Policy page (from the **Design** menu, choose **Policies** under the **DHCP Settings** submenu, then **Add Policies**) or Edit DHCP Policy page (from the **Design** menu, choose **Policies** under the **DHCP Settings** submenu, then the name of the policy).

CLI Commands

The syntax for setting the *override-client-id* attribute is **client-class name set override-client-id="expression"**.
 The syntax for setting the *client-class-lookup-id* attribute is **dhcp set client-class-lookup-id="expression"**.
 The syntax for setting the *use-client-id-for-reservations* attribute is **policy name enable use-client-id-for-reservations**.

Reservation Override Example

The following example shows how to override a client ID for a reservation:

-
- Step 1** Create a scope for the reservation:
- a) Enter a subnet address.
 - b) If you want dynamic reservations, add an IP address range.
- Step 2** Add the reservation for the scope:
- a) Include a value for the lookup key.
 - b) Specify the lookup key type as binary.
- Step 3** Create a policy for the purpose, enabling the *use-client-id-reservations* attribute.
- Step 4** Create a client-class for the purpose:
- a) Specify the policy created in the previous step.
 - b) Include an expression for the *override-client-id* attribute that returns a blob value with the client ID you want, based on the contents of the packet.
- Step 5** Get a lease for a client with the MAC address. This client will then get the override ID.
-

Reconfiguring IPv6 Leases

For DHCPv6 leases, you can send a RECONFIGURE message to a client to inform the client that the server has new or updated configuration parameters. If so authorized and through proper authentication, the client then immediately initiates a Renew, Rebind, or Information-request reply transaction with the server so that the client can retrieve the new data.

For more information on enabling the DHCPv6 policy reconfiguration, see [Configuring DHCPv6 Policies, on page 162](#).

Local Advanced Web UI

The List/Add DHCP Leases for Prefix page includes a **Reconfigure** button for each lease so that you can initiate a reconfiguration request for that particular lease.

CLI Commands

To support Reconfigure, Cisco Prime Network Registrar includes the following syntax for the **lease6** command:

```
lease6 [<vpn-name>/]<ipaddr> reconfigure [renew | rebind | information-request] [-unicast | -via-relay]
```

The options determine whether to have the client respond to the Reconfigure message with a Renew, Rebind, or Information-request packet, and whether the server should unicast or go through a relay agent. The **lease6 list** and **lease6 show** commands also display values for these related attributes:

- **client-reconfigure-key**—128-bit key that the server generates for Reconfigure messages to the client.
- **client-reconfigure-key-generation-time**—Time at which the server generated the *client-reconfigure-key*.

The **policy** command includes two related attribute settings:

- **reconfigure**—Whether to allow (1), disallow (2), or require (3) Reconfigure support; the preset value is allow (1).
- **reconfigure-via-relay**—Whether to allow reconfiguration over a relay agent; the preset value is false, whereby reconfiguration notification is by unicasting from the server.

Forcing Lease Availability

You can force a current lease to become available. You should request that the user release the lease, or do so yourself, before forcing its availability. Forcing lease availability does not require a server reload.



Note After a lease is forced to be available, the client continues to use it until the client contacts the DHCP server.

Local Advanced Web UI

To force lease availability, do the following:

-
- Step 1** From the **Design** menu, choose **Scopes** under the **DHCPv4** submenu to open the List/Add DHCP Scopes page.
 - Step 2** Click the **Lease** tab for the scope that has leases.

- Step 3** Click the IP address of the lease on the Edit DHCP Scope page.
- Step 4** Click **Force Available**. On the List DHCP Leases for Scope page, the lease will now show an empty value in the Flags column
-

CLI Commands

To force lease availability, use `lease [vpn/]ipaddr force-available`. Use `scope name clearUnavailable` to force all "Unavailable" leases in the scope to change to "Available" states.

Inhibiting Lease Renewals

Normally, the Cisco Prime Network Registrar DHCP server retains the association between a client and its leased IP address. The DHCP protocol explicitly recommends this association and it is a usually desirable feature. However, for some customers, such as ISPs, clients with long-lived lease associations may be undesirable, because these clients should change their IP addresses periodically. Cisco Prime Network Registrar includes a feature that allows customers to force lease associations to change when DHCP clients attempt to renew their leases or reboot.

A server can never force a client to change its lease, but can compel the client to do so based on a DHCPRENEW or DHCPDISCOVER request. Cisco Prime Network Registrar offers configuration options to allow customers to choose which interactions to use to force a client to change its IP address:

- **Inhibiting all lease renewals**—While a client is using a leased address, it periodically tries to extend its lease. At each renewal attempt, the server can reject the lease, forcing the client to stop using the IP address. The client might have active connections that are terminated when the lease terminates, so that renewal inhibition at this point in the DHCP interaction is likely to be user-visible.
- **Inhibiting renewals at reboot**—When a DHCP client reboots, it might have recorded a valid lease binding that did not expire, or it might not have a valid lease. If it does not have a lease, you can prevent the server from granting the last held lease. If the client has a valid lease, the server rejects it, forcing the client to obtain a new one. In either case, no active connections can use the leased address, so that the inhibition does not have a visible impact.
- **Effect on reservations**—Reservations take precedence over renewal inhibition. If a client has a reservation, it can continue to use the reserved IP address, whether or not renewal inhibition is configured.
- **Effect on client-classes**—Client-class testing takes place after renewal inhibition testing. If a client is forced to change IP addresses by renewal inhibition, then client-class processing might influence which address the server offers to the client.

You can enable or disable lease renewal inhibition for a policy, which you can set system wide, for a scope or on a client-by-client basis. The *inhibit-all-renews* attribute causes the server to reject all renewal requests, forcing the client to obtain a new IP address any time it contacts the DHCP server. The *inhibit-renews-at-reboot* attribute permits clients to renew their leases, but the server forces them to obtain new addresses each time they reboot.

The DHCP server needs to distinguish between a client message that it should reject (such as a renewal request) and one that represents a retransmission. When the server processes a message, it records the time the packet arrived. It also records the time at which it made a lease binding to a client, and the last time it processed a message from the client about that binding. It then compares the packet arrival time with the lease binding time (the start-time-of-state) and processes packets from the client within a certain time interval from the start time of the binding. By default, this time interval is one minute.

Local Advanced Web UI

To inhibit lease renewals, create a policy on the Edit DHCP Policy page (click **Design**, then **Policies** under the **DHCP Settings** submenu, then the name of the policy), then enable the *inhibit-all-renews* or *inhibit-renews-at-reboot* attribute. (Both attributes are preset to disabled). Then, modify the policy.

Moving Leases Between Servers

There may be a need to move leases to a new DHCP server such as, the configuration of the server grows sufficiently large to exceed the recommended limits. There are different ways to accomplish this task depending on whether the leases are being moved to a new server or an existing server. Either of these techniques requires special considerations and careful execution. A new server is often the simplest way to accomplish by moving the entire configuration and the state database. To move the leases to another server, the leaseadmin utility is used. This utility allows you to export all or a selected set of leases and also to import this exported lease set.



Caution

The Leaseadmin utility must only be used on a local cluster (exporting or importing) and the DHCP server must be stopped before running the leaseadmin utility.

The leaseadmin utility was added to Cisco Prime Network Registrar to allow leases to be moved from one server to another. You must run this utility on the same machine as the DHCP server and you must have superuser/root privileges to read and modify the database file. This utility requires direct access to the lease state database; however, stopping the DHCP server is not sufficient as the stopped server still holds the lease state database open. If the utility is run when the database is still in use, the leaseadmin utility will report the error "Failed to obtain exclusive access to lease state database". The default location is:

- **Windows**—\Program Files\Cisco Prime Network Registrar\bin
- **Linux**—/opt/nwreg2/local/bin

From the command prompt, change to the above location and run the utility using the syntax:

```
# leaseadmin <options>
```

The table below describes the qualifying options for the **leaseadmin** utility.

Table 26: leaseadmin Command Options

Option	Description
To export lease(s)	
-e filename	Exports to a file.
-x	Sends raw output format (required to import).
-t {current history detail all v6leases v6history}	Specifies the record types to be exported. Valid values are: current, history, detail, all, v6leases, and v6history
-s subnet prefix	Restrict the lease records to be exported to a subnet or a prefix.
To import lease(s)	

Option	Description
<code>-i filename</code>	Import from a file. When used with <code>-n</code> option, specifies the VPN.
<code>-o</code>	When used with the <code>-i</code> (import) option, overwrites the existing data.
<code>-c</code>	Compress records.
To delete lease(s) or the server DHCP Unique Identifier (DUID)	
<code>-d address subnet prefix</code>	<p>Specifies the address, subnet, prefix, or server DUID to be deleted.</p> <p>Note When you specify the server DUID, the auto-generated DHCPv6 server DUID is deleted.</p> <p>While not recommended (see Things to Avoid When Troubleshooting Issues Related to Failover, on page 88), if lease databases are ever copied to another local cluster, it is critical that any server-duid that may be present in the copied database is deleted from it using this operation.</p> <p>If the server-duid is deleted, when the DHCP server starts, it will generate a new server-duid. This will cause all DHCPv6 Renew requests specifying the older server-duid as the server-id option to be dropped until the client starts sending DHCPv6 Rebind requests.</p>
General Options	
<code>-n vpn</code>	When used with <code>-e</code> (export), <code>-i</code> (import), or <code>-d</code> (delete) option, specifies the VPN. To include all VPNs specify "all".
<code>-h path</code>	Overrides the default path to the database.
<code>-v</code>	Displays the database version.
<code>-z {letters}=level</code>	Sets the debug output levels.

Handling Leases Marked as Unavailable

One of the aspects of effective lease maintenance is determining the number of unavailable leases in a scope. This number is sometimes higher than expected. Each unavailable lease is probably an indication of a serious problem. Possible causes for an unavailable lease are:

- **The DHCP server is configured for a ping before an offer, and the ICMP echo message is returned successfully**—A currently active client is using that IP address, causing the DHCP server to mark it as *unavailable*. To prevent the server from doing so, disable pinging an address before offering it to a client. See [Pinging Hosts Before Offering Addresses, on page 194](#).
- **The server receives a DHCPDECLINE message from a client to which it leased what it considered to be a good IP address**—The client does an address resolution (ARP) request for the IP address on its local LAN segment, and another client responds to it. The client then returns the address to the server with a DHCPDECLINE packet and sends another DHCPDISCOVER packet to get a new address. The server marks as *unavailable* the address that the client returns. To prevent the server from reacting to DHCPDECLINE messages, you can set a scope attribute, *ignore-declines*.
- **The server receives “other server” requests from the client**—Because all DHCPREQUEST messages that follow DHCPDISCOVER messages are broadcast, the server can see messages directed to other DHCP servers. A server knows that a message is directed to it by the value of the *server-id* option in the packet. If the Cisco Prime Network Registrar recognizes a message directed at another server, in that its own IP address does not appear in the *server-id* option, but the address leased in the message is one that the server controls, it believes that two servers must be trying to manage the address simultaneously. It then marks the local address as *unavailable*. This behavior does not apply in a DHCP failover configuration. Either the two servers are configured with some or all of the same IP addresses, or (in rare cases) the DHCP client placed a wrong *server-id* option value in the packet.

If you have reason to believe that the client is sending bad *server-id* options (rather than packets actually directed to other servers), Cisco Prime Network Registrar has a server attribute you can enable that turns this behavior off, the *ignore-requests-for-other-servers* attribute.

- **Inconsistent lease data**—Extremely rare and occurring only during server startup when, while configuring a lease, the server reads the lease data from disk during a refresh of the internal cache. The lease state appears as *leased*, but there is incomplete data to construct a client for that lease, such as that the lease might not yet have a *client-id* option value. The server considers the data to be inconsistent and marks the IP address as *unavailable*. Forcing the lease to be available (such as by using the **lease ipaddr force-available** command in the CLI) should clear up this problem.

Setting Timeouts for Unavailable Leases

During the times when leases become unavailable, as described in [Handling Leases Marked as Unavailable, on page 214](#), all unavailable leases remain in that state for a configured time only, after which time they again become available. A policy attribute, *unavailable-timeout*, controls this time. The *system_default_policy* policy sets this value to one day by default.

To handle upgrades from previous releases of Cisco Prime Network Registrar that do not have this timeout feature, a special upgrade timeout attribute, *upgrade-unavailable-timeout* (which is preset to one day) is included at the server level. The *upgrade-unavailable-timeout* value is the timeout given to leases set to unavailable before the Cisco Prime Network Registrar upgrade. This setting affects the running server only and does not rewrite the database. If the server stays up for one day without reloading, all the unavailable leases that were present at the last reload will time out. If the server reloads in less than a day, the entire process restarts with the next reload. Note that this process occurs only for leases that were set unavailable before the upgrade. Leases that become unavailable after the upgrade receive the *unavailable-timeout* value from the policy, as previously described.

If a Cisco Prime Network Registrar failover server receives an update from a Cisco Prime Network Registrar DHCP server running prior to Cisco Prime Network Registrar 6.0, the unavailable leases do not have a timeout value. In this case, the upgraded Cisco Prime Network Registrar server uses the *unavailable-timeout* value

configured in the scope policy or *system_default_policy* policy as the timeout for the unavailable lease. When the lease times out, the policy causes the lease to transition to available in both failover partners.

Querying Leases

Cisco Prime Network Registrar can work together with Cisco routers to provide enhanced provisioning capabilities. This function is described in the DHCP Leasequery specification (RFC 4388), with which Cisco Prime Network Registrar conforms. Part of the implementation of the Cisco uBR access concentrator relay agent is to capture and glean information from DHCP lease requests and responses. It uses this information to:

- Associate subscriber cable modem and client MAC addresses with server-assigned IP addresses.
- Verify source IP addresses in upstream datagrams.
- Encrypt unicast downstream traffic through the DOCSIS Baseline Privacy protocol.
- Avoid broadcasting downstream Address Resolution Protocol (ARP) requests, which can burden the the uBR as well as the subscriber hosts, and which malicious clients can compromise.

The uBR device does not capture all DHCP state information through gleaning. The uBR device cannot glean from unicast messages (particularly renewals and releases) because capturing them requires special processing that would degrade its forwarding performance. Also, this data does not persist across uBR reboots or replacements. Therefore, the only reliable source of DHCP state information for the uBR device is the DHCP server itself.

For this reason the DHCP server supports the DHCPLEASEQUERY message, which is similar to a DHCPINFORM message. Access concentrators and relay agents can thereby obtain client location data directly from the DHCP server, for DHCPv4 and DHCPv6 addresses.

Related Topics

[Leasequery Implementations, on page 216](#)

[Pre-RFC Leasequery for DHCPv4, on page 217](#)

[RFC 4388 Leasequery for DHCPv4, on page 218](#)

[Leasequery for DHCPv6, on page 218](#)

[Leasequery Statistics, on page 219](#)

[Leasequery Example, on page 220](#)

Leasequery Implementations

Cisco Prime Network Registrar provides three Leasequery implementations:

- DHCPv4 Cisco-proprietary for pre-RFC 4388—See [Pre-RFC Leasequery for DHCPv4, on page 217](#)
- DHCPv4 compliant with RFC 4388—See [RFC 4388 Leasequery for DHCPv4, on page 218](#)
- DHCPv6—See [Leasequery for DHCPv6, on page 218](#)

The Cisco-proprietary and the more recent RFC-compliant implementations for DHCPv4 differ in only minor ways and will coexist. The DHCP server accepts Leasequery requests at the same port and returns the specified data for both implementations. The DHCPv6 implementation conforms with RFC 5007 and RFC 5460.

The DHCP server can include lease reservation data in Leasequery responses for DHCPv4 and DHCPv6. Cisco Prime Network Registrar returns a default lease time of one year (31536000 seconds) for reserved DHCPv4 and lifetime of the leases for DHCPv6 leases in a response. If the IP address is actually leased, Cisco Prime Network Registrar returns its remaining lease time.

Leasequery is preset to be enabled for all the implementations. To disable it, disable an Expert mode attribute, *leasequery*.

Pre-RFC Leasequery for DHCPv4

Leasequery messages usually contain request fields and options. To illustrate, suppose that after a relay agent reboot or replacement, the relay agent receives a request to forward a datagram downstream to the public broadband access network. Because the relay agent no longer has the downstream location data, it sends a LEASEQUERY message to the DHCP server that includes the gateway IP address (*giaddr*) of the relay agent and the MAC address or *dhcp-client-identifier* (option 61) of the target client. If the DHCP server finds the client, it returns the client IP address in the client address (*ciaddr*) field in the response to the leasequery. If the server cannot find the client address, it returns a DHCPNACK.

In the pre-RFC implementation for DHCPv4, the requestor can query by IP address, client ID option (61), or MAC address, and receives from the server a DHCPACK (with the returned data) or a DHCPNACK message, or the server drops the packet. If the request includes multiple query types, the DHCP server responds to the first one it can find. The *giaddr* value from the requestor is independent of the *ciaddr* searched and is simply the return IP address for any responses from the server. The three possible query types are:

- **IP address (*ciaddr*)**—The request packet includes an IP address in the *ciaddr* field. The DHCP server returns data for the most recent client to use that address. A packet that includes a *ciaddr* value must be a request by IP address, despite the values in the MAC address fields (*htype*, *hlen*, and *chaddr*) or *dhcp-client-identifier* option. Querying by IP address is the most efficient method and the one most widely used, in that the other two methods can put more load on the DHCP server.
- **dhcp-client-identifier option (61)**—The request packet includes a *dhcp-client-identifier* option value. The DHCP server returns a DHCPACK packet containing the IP address data for the most recently accessed client. If the request omits a MAC address, the server returns all IP addresses and their data for the requested client ID in the *cisco-leased-ip* (also called the *associated-ip*) option. If the request includes the MAC address, the server matches the *dhcp-client-identifier* and MAC address with the client data for the IP address and returns that data in the *ciaddr* field or *cisco-leased-ip* (also called the *associated-ip*) option.
- **MAC address**—The request packet includes a MAC address in the hardware type (*htype*), address length (*hlen*), and client hardware address (*chaddr*) fields, and a blank *ciaddr* field. The server returns all the IP addresses and most recent lease data for the MAC address in the *cisco-leased-ip* (also called the *associated-ip*) option of the reply packet.

The DHCPLEASEQUERY message number in the *dhcp-message-type* option (53) for the pre-RFC implementation is 13. A server that does not support this type of message is likely to drop the packet. The DHCPACK message reply always contains the physical address of the lease owner in the *htype*, *hlen*, and *chaddr* fields. If the request contains the *ciaddr*, the data returned is always based on the *ciaddr* and never the client ID or MAC address.

The requestor can include the *parameter-request-list* option (55) to request specific options about an address. The reply often contains the *dhcp-lease-time* option (51) and the original content of the *relay-agent-info* option (82) that the client sent. If the server does not detect a valid lease for a client, it does not return option 51, and the requestor needs to determine if there is a valid lease.

A DHCPACK from the server can also contain the following Leasequery options:

- **cisco-leased-ip (161)**—Data on all the IP addresses associated with the client; also known as (and later renamed) the *associated-ip* option.
- **cisco-client-requested-host-name (162)**—Hostname that the client requested in the *host-name* option (12) or *client-fqdn* option (81). The requested hostname was dropped in the RFC 4388 implementation.
- **cisco-client-last-transaction-time (163)**—Most recent time duration that a DHCP server contacted the client.

RFC 4388 Leasequery for DHCPv4

Leasequery became an official RFC 4388 for DHCPv4 in February 2006. Cisco Prime Network Registrar provides the RFC 4388 implementation alongside the pre-RFC one (see [Pre-RFC Leasequery for DHCPv4, on page 217](#)) and there are no conflicts between them. However, the RFC 4388 implementation includes a few notable changes:

- The DHCPLEASEQUERY message type contained in the *dhcp-message-type* option (53) changed its message ID to 10 (the ID 13 was given to the DHCPLEASEACTIVE message), and the reply messages were changed from just DHCPACK and DHCPNACK to be more specific:
 - DHCPLEASEQUERY (10) for queries
 - DHCPLEASEUNASSIGNED (11) for replies of unassigned addresses
 - DHCPLEASEUNKNOWN (12) for replies of unknown addresses
 - DHCPLEASEACTIVE (13) for replies of active addresses
- The reply option names and IDs changed, and the *cisco-client-requested-host-name* option was dropped so that there are only two reply options:
 - **client-last-transaction-time (91)**—Most recent time duration that a DHCP server contacted the client.
 - **associated-ip (92)**—Data on all the IP addresses associated with the client.
- If querying by client ID or MAC address, the request can contain only the *dhcp-client-identifier* option (61) or MAC address; if the packet contains both, the server drops it.

Leasequery for DHCPv6

Cisco Prime Network Registrar supports both the RFC 5007 (UDP) and RFC 5460 (TCP, Bulk) DHCPv6 leasequery capabilities.



Note To use the RFC 5460 (TCP, Bulk) leasequery support, you must create a DHCP Listener for IPv6 (see [DHCP Listener Configuration, on page 237](#)).

The message types for DHCPv6 Leasequery are:

- LEASEQUERY (14)
- LEASEQUERY_REPLY (15)
- LEASEQUERY_DATA (17)
- LEASEQUERY_DONE (16)
- ACTIVELEASEQUERY(240)

A query can be by:

- QUERY_BY_ADDRESS (1)
- QUERY_BY_CLIENTID (2)
- QUERY_BY_RELAY_ID(3)
- QUERY_BY_LINK_ADDRESS(4)
- QUERY_BY_REMOTE_ID(5)

A DHCPv6 LEASEQUERY_REPLY message can contain one or more of the following options:

- **lq-query (44)**—Query being performed. The option, used in a request only, includes the query type, link-address (or 0::0), and options to provide data needed for the query.
- **client-data (45)**—Encapsulates the data for a single client on a single link. The client data can include any number of these or other requested options.
- **clt-time (46)**—Client last transaction time encapsulated in a *client-data* option (45); identifies how long ago (in seconds) the server last communicated with the client.
- **lq-relay-data (47)**—Relay agent data used when the client last communicated with the server. Fields are the peer-address and the relay-message. This option can include further options.
- **lq-client-link (48)**—Links on which the client has any bindings. Used in reply to a client query when the link-address is omitted and the client is found to be on more than one link.
- **option_lq_base_time**—Specifies the current absolute time on DHCPv6 server at the time of sending binding information.

A DHCPv6 LEASEQUERY_REQUEST message can contain one or more of the following options:

- **option_lq_start_time**—Bindings updated since the specified time. This option, used for the list of binding updates happened during the offline period.
- **option_lq_end_time**—Bindings updated during the specified time period.

DHCPv6 uses the Option Request option (*oro*) to request a list of options in the Leasequery reply.


Note

Leasequery by client-id requests may need to specify the *override-client-id* attribute when using `[v6-override-client-id]` expressions to correctly retrieve the information on the lease(s) for the client.

Leasequery Statistics

Lease queries provide statistics attributes, in the web UI, on the DHCP Server Statistics page (see the *"Displaying Statistics"* section in *Cisco Prime Network Registrar 9.0 Administrator Guide*), and in the CLI by using **dhcp getStats**. The Leasequery statistics are:

- **lease-queries**—Number of RFC 4388 message ID 10 (or pre-RFC message ID 13) DHCPv4 Leasequery packets received in the given time interval.
- **lease-queries-active**—Number of RFC 4388 DHCPLEASEACTIVE packets.
- **lease-queries-unassigned**—Number of RFC 4388 DHCPLEASEUNASSIGNED packets.
- **lease-queries-unknown**—Number of RFC 4388 DHCPLEASEUNKNOWN packets.
- **leasequeries**—Number of DHCPv6 Leasequery packets received.
- **leasequery-replies**—Number of responses to DHCPv6 Leasequery packets that might or might not be successful.
- **tcp-current-connections**—Number of currently open TCP connections to the DHCP server for DHCPv6 Active and Bulk Leasequery.

- **tcp-total-connections**—Number of TCP connections that were opened to the DHCP server for DHCPv6 Active and Bulk Leasequery in this time interval.
- **bulk-leasequeries**—Number of LEASEQUERY packets received over all TCP connections in this timebulk-leasequeries interval.
- **bulk-leasequery-replies**—Number of LEASEQUERY-REPLY packets sent over all TCP connections in this time interval.
- **bulk-leasequery-data**—Number of LEASEQUERY-DATA packets sent over all TCP connections in this time interval.
- **bulk-leasequery-done**—Number of LEASEQUERY-DONE packets sent over all TCP connections in this time interval.
- **tcp-lq-status-unspec-fail**—Number of LEASEQUERY-REPLY packets with a status code of UnspecFail(1) sent over TCP in this time interval.
- **tcp-lq-status-unknown-query**—Number of LEASEQUERY-REPLY packets with a status code of UnknownQueryType(7) sent over TCP in this time interval.
- **tcp-lq-status-malformed-query**—Number of LEASEQUERY-REPLY packets with a status code of MalformedQuery(8) sent over TCP in this time interval.
- **tcp-lq-status-not-configured**—Number of LEASEQUERY-REPLY packets with a status code of NotConfigured(9) sent over TCP in this time interval.
- **tcp-lq-status-not-allowed**—Number of LEASEQUERY-REPLY packets with a status code of NotAllowed(10) sent over TCP in this time interval.
- **tcp-lq-status-query-terminated**—Number of LEASEQUERY-REPLY/LEASEQUERY-DONE packets with a status code of QueryTerminated(11) sent over TCP in this time interval.
- **tcp-connections-dropped**—Number of TCP requests that were terminated in this time interval because the TCP connection was closed (or reset) by the DHCPv6 requester. This excludes normal connection closes or server reloads.
- **active-leasequeries**—Number of ACTIVELEASEQUERY packets received over all TCP connections in this time interval.
- **active-leasequery-replies**—Number of LEASEQUERY-REPLY packets sent over all TCP connections in this time interval for active leasequery.
- **active-leasequery-data**—Number of LEASEQUERY-DATA packets sent over all TCP connections in this time interval for active leasequery.
- **active-leasequery-done**—Number of LEASEQUERY-DONE packets sent over all TCP connections in this time interval for active leasequery.
- **tcp-lq-status-data-missing**—Number of LEASEQUERY-REPLY packets with a status code of DataMissing(240) sent over TCP in this time interval.
- **tcp-lq-status-catch-up-complete**—Number of LEASEQUERY-DATA packets with a status code of CatchUpComplete(241) sent over TCP in this time interval.

Leasequery Example

The example below shows a packet trace of a DHCPv6 UDP query by client ID without a link-address, but with addresses on more than one link. The first part of the output shows the query message and the second part shows the reply data. The *lq-query* option identifies the type of query. Note the list of requested options via the Option Request option (*oro*) in the request, and the two addresses returned in the *lq-client-links* option in the reply.

Example: Packet Trace of Sample UDP Lease Query

```
+- Start of LEASEQUERY (14) message (113 bytes)
```

```

| transaction-id 22
| lq-query (44) option (37 bytes)
| (query-type 2, link-address ::)
| client-identifier (1) option (10 bytes)
| 00:03:00:01:01:02:03:04:05:06
| oro (6) option (2 bytes)
| 47
| server-identifier (2) option (14 bytes)
| 00:01:00:01:13:06:6a:67:00:23:7d:53:e5:e3
| client-identifier (1) option (10 bytes)
| 00:03:00:01:01:03:05:07:09:11
| vendor-class (16) option (14 bytes)
| (enterprise-id 1760,
| ((00:08:41:49:43:20:45:63:68:6f)))
| vendor-class (16) option (14 bytes)
| (enterprise-id 1760,
| ((00:08:41:49:43:20:45:63:68:6f)))
+- End of LEASEQUERY message
+- Start of LEASEQUERY-REPLY (15) message (72 bytes)
| transaction-id 22
| server-identifier (2) option (14 bytes)
| 00:01:00:01:13:06:6a:67:00:23:7d:53:e5:e3
| client-identifier (1) option (10 bytes)
| 00:03:00:01:01:03:05:07:09:11
| lq-client-links (48) option (32 bytes)
| 2001:4f8:ffff:0:8125:ef1b:bdcb:4b4e,2001:4f8:ff00:0:e400:f92:1bfd:60fa
+- End of LEASEQUERY-REPLY message

```

The example below shows a packet trace of a DHCPv6 TCP query by client ID. The first part of the output shows the request message, the second part shows the response message with the binding data of the first client, and the last part will show that the query has ended successfully. The third part will follow the second part if there are more than a single client to be returned.



Note The LEASEQUERY-DONE message will not be present in a packet if the LEASEQUERY-REPLY message does not have any binding data.

Example: Packet Trace of Sample TCP Lease Query

```

+- Start of LEASEQUERY (14) message (59 bytes)
| transaction-id 2
| lq-query (44) option (37 bytes)
| (query-type 2, link-address ::)
| client-identifier (1) option (10 bytes)
| 00:03:00:01:01:02:03:04:05:06
| oro (6) option (2 bytes)
| 47
| client-identifier (1) option (10 bytes)
| 00:03:00:01:01:03:05:07:09:11
+- End of LEASEQUERY message

+- Start of LEASEQUERY-REPLY (15) message (162 bytes)
| transaction-id 2
| server-identifier (2) option (14 bytes)
| 00:01:00:01:13:06:6a:67:00:23:7d:53:e5:e3
| client-identifier (1) option (10 bytes)
| 00:03:00:01:01:03:05:07:09:11
| client-data (45) option (122 bytes)
| client-identifier (1) option (10 bytes)
| 00:03:00:01:01:02:03:04:05:06

```

```

| clt-time (46) option (4 bytes)
| 5m54s
| iaaddr (5) option (24 bytes)
| (address 2001:4f8:ffff:0:8125:ef1b:bdcb:4b4e,
| preferred-lifetime 6d23h54m6s,
| valid-lifetime 1w6d23h54m6s)
| lq-relay-data (47) option (68 bytes)
| peer-address fcc0:a803::214:4fff:fecl:226a
| +- Start of RELAY-FORW (12) message (52 bytes)
| | hop-count 0,
| | link-address 2001:4f8:ffff::,
| | peer-address fe80::302:3ff:fe04:506
| | vendor-class (16) option (14 bytes)
| | (enterprise-id 1760,
| | ((00:08:41:49:43:20:45:63:68:6f)))
| +- End of RELAY-FORW message
+- End of LEASEQUERY-REPLY message
+- Start of LEASEQUERY-DATA (17) message (130 bytes)
| transaction-id 2
| client-data (45) option (122 bytes)
| client-identifier (1) option (10 bytes)
| 00:03:00:01:01:02:03:04:05:06
| clt-time (46) option (4 bytes)
| 5m33s
| iaaddr (5) option (24 bytes)
| (address 2001:4f8:ff00:0:e400:f92:1bfd:60fa,
| preferred-lifetime 6d23h54m27s,
| valid-lifetime 1w6d23h54m27s)
| lq-relay-data (47) option (68 bytes)
| peer-address fcc0:a803::214:4fff:fecl:226a
| +- Start of RELAY-FORW (12) message (52 bytes)
| | hop-count 0,
| | link-address 2001:4f8:ff00::,
| | peer-address fe80::302:3ff:fe04:506
| | vendor-class (16) option (14 bytes)
| | (enterprise-id 1760,
| | ((00:08:41:49:43:20:45:63:68:6f)))
| +- End of RELAY-FORW message
+- End of LEASEQUERY-DATA message

+- Start of LEASEQUERY-DONE (16) message (4 bytes)
| transaction-id 2
+- End of LEASEQUERY-DONE message

```

Difference between TCP bulk leasequery and UDP leasequery

The following are the differences between TCP bulk leasequery and UDP leasequery:

- UDP leasequery supports Query by IPv6 Address and Query by Client Identifier. However, TCP Bulk Leasequery supports all the five query types; that is, Query by IPv6 Address, Query by Client Identifier, Query by Relay Identifier, Query by Link Address, and Query by Remote ID.
- In UDP Leasequery, if the server finds bindings for the relay agent on multiple links, then DHCP server will send an option `OPTION_CLIENT_LINK` in the reply message. The relay agent will need to resend `LEASEQUERY` messages using each of the returned link-addresses to obtain the all client's bindings. Whereas in TCP Bulk Leasequery, the server returns multiple bindings of a client on different links; however `OPTION_CLIENT_LINK` is not supported in Bulk Leasequery reply.

Running Address and Lease Reports

You can run these reports on IP addresses and leases:

- **Address Usage**—See [Running Address Usage Reports, on page 223](#)
- **Lease History**—See [Running IP Lease Histories, on page 223](#)
- **Current Utilization**—See [Running Lease Utilization Reports, on page 229](#)
- **Lease Notification**—See [Receiving Lease Notification, on page 229](#)

Running Address Usage Reports

The address usage reports show the IP addresses that are assigned leases.

Local Advanced Web UI

To view the leases for IP addresses, on the Edit DHCP Scope page, click the **Leases** tab, to open the List DHCP Leases for the scope. To manage a specific lease, click its IP address on the page. This opens the Manage DHCP Lease page.

CLI Commands

To view the IP address usage for specified servers, use **report**.



Tip If you are not already using **lease-notification** in an automated way, try **lease-notification available=100%** for a concise scope-by-scope summary of the state of the servers.

Running IP Lease Histories

You can extract IP lease history data from a special database so that you can determine past allocation information for a given IP address. You can get a historical view of when a client was issued a lease, for how long, when the client or server released the lease before it expired, and if and when the server renewed the lease and for how long.

Cisco Prime Network Registrar provides a client to control querying IP history data. Through this client, you can:

- Get the MAC addresses associated with a given IP address over a given time.
- See the entire IP history database as a comma-separated file.
- View the attributes of the lease history (the lease history detail report)—See [Querying IP Lease History, on page 224](#).

You must use additional administrative functions to trim the IP history database of records, to keep the size of the database from growing without bounds.



Note When the state of an existing lease changes (for example, when it is configured as a reserved IP address or it is deactivated), the change does not appear as a lease history change at the regional. With detail collection disabled, a lease history change appears only when the lease transitions from leased to not leased or is assigned to another client.

Enabling Lease History Recording at the Local Cluster

You must explicitly enable lease history recording for the local cluster DHCP server. The DHCP server logs IP history recording errors in the usual DHCP log files.

When the lease history is enabled on a local cluster it impacts the performance of the server and the size of the lease state database. A history record is created for the lease whenever a lease ends (expires or is released); a lease that a client renews over a long period does not cause a history record to be created. The size of each lease history record depends on many factors, but a good estimate is about 1 KB per record. Depending on the rate at which the lease ends and the duration over which lease history is kept, this could result in a sizeable number of lease history records being created and thus requires a considerable disk space. This could be many orders larger than the space needed for the active leases.

Local Advanced Web UI

To enable lease history recording, do the following:

-
- Step 1** From the **Deploy** menu, choose **DHCP Server** under the **DHCP** submenu to open the Manage DHCP Server page.
 - Step 2** Click the **Local DHCP Server** on the DHCP Server pane.
 - Step 3** On the Edit Local DHCP Server page, look for the Lease History attributes:
 - **Lease History (*ip-history*)**—Enable or disable the lease history database for v4-only (DHCPv4), v6-only (DHCPv6), or both.
 - ***ip-history-max-age***—Maximum age of the lease history to collect. With lease history set to v4 only, v6 only, or both the DHCP server periodically examines the lease history records and deletes any records with lease history bindings older than this age threshold.
 - Step 4** Click **Save**.
 - Step 5** Reload the server.
-

CLI Commands

To enable lease history recording, you must explicitly enable recording IP (lease) history for IP addresses by using `dhcp set ip-history=<value>` (**v4-only**, **v6-only**, **both**, or **disable**).

Querying IP Lease History

Once you have leases, you can query for their history. You can query IP lease history either from a local or a regional cluster. Set up the local cluster containing the DHCP server as part of the regional cluster, and

enable polling for the lease history data from the regional cluster (see the *"Enabling Lease History Collection"* section in *Cisco Prime Network Registrar 9.0 Administrator Guide*).

You can adjust the polling criteria for the cluster in the regional cluster web UI by using the attributes described in the *"Polling Subnet Utilization and Lease History Data"* section in *Cisco Prime Network Registrar 9.0 Administrator Guide*.

You must also set the selection criteria for querying the lease history data, as described in the following sections.

Local and Regional Advanced Web UI

To query the IPv4 lease history, do the following:

Step 1 From the **Operate** menu, choose **DHCPv4 Lease History** under the **Reports** submenu to open the DHCP Lease History Search page.

Note You can use the Search button in the Local Advanced Web UI to move to DHCP Lease Search page. This button helps you to toggle between lease history search page and active leases search page.

Step 2 Choose the Filter attribute and the Type from the drop down lists and enter the value of the filter type selected in the Value field.

Step 3 Click **Search** to display the list of leases.

Local and Regional Advanced Web UI

To query the IPv6 lease history, do the following:

Step 1 From the **Operate** menu, choose **DHCPv6 Lease History** under the **Reports** submenu to open the DHCP v6 Lease History Search page.

Note You can use the Search button in the Local Advanced Web UI to move to DHCP v6 Lease Search page. This button helps you to toggle between lease history search page and active leases search page.

Step 2 Choose the Filter attribute and the Type from the drop down lists and enter the value of the filter type selected in the Value field.

Step 3 Click **Search** to display the list of leases.

Local and Regional Advanced Web UI



Note The regional server only searches its version of the lease history which is as recent as the latest poll. For the most up-to-date data, this might require performing an explicit lease history poll for the regional to retrieve the latest lease history data.

Using the iphist Utility

You can query the IP history database at the local as well as regional clusters and direct the results to standard output or a file by using the **iphist** utility. The default location is:

- **Windows**—\Program Files\Cisco Prime Network Registrar\bin
- **Linux**—/opt/nwreg2/local/usrbin

From the command prompt, change to the above location and run the utility using the syntax:

iphist [*options*] {*ipaddr* | **all**} [*start-date* | **start** [*end-date* | **end**]]

The IP address is a single address or the keyword **all**, the start date is in local time or the keyword **start** for the earliest date in the database, and the end date is in local time or the keyword **end** for the last date in the database. However, the output is in Greenwich Mean Time (GMT) by default, unless you use the **-l** option to specify local time.

The full list of command options appears in the table below.

Table 27: iphist Command Options

Option	Description
-N <i>username</i>	Administrator username. If omitted, you are prompted for the username.
-P <i>password</i>	Administrator password. If omitted, you are prompted for the password.
-C <i>cluster</i> [<i>:port</i>]	Destination server and optional SCP port.
-6	Output DHCPv6 leases
-a	Shows the lease attributes, visibility 3.
-f <i>format</i>	Format of the output lines. The default format is: "address,client-mac-addr,binding-start-time,binding-end-time"
-t	Print format as title line.
-n <i>namespace</i>	Specify the namespace for the address.
-o <i>file</i>	Sends output to a file.
-l	Displays output in local time rather than the default UTC/GMT.
-i	Displays output for delegated prefix that includes specified IPv6 address (only with -6).
-s { <i>self</i> <i>partner</i> }	Restricts the leases to the self or partner.
-v	Displays the output version.
-z <i>debug-args</i>	Sets the debug output levels.

Dates can use this syntax (quotation marks are required if space characters are included):

- *month /day /year @hour :min :sec* (for example, 8/28/2007@10:01:15), with the time optional
- *month /day /year hour :min :sec* (for example, "8/28/2007 10:01:15"), with the time optional

- *month day hour :min :sec year* (for example, “Aug 28 10:01:15 2007”), with the seconds optional
- Keywords **start**, **end**, or **now** (for the current time)

The date filtering is intended to limit the output to leases that were active during that time. This means that they can begin before the specified start date, as long as they do not end before the start date. They can also not begin after the specified end date. For example, invoking the command:

```
# ./iphist -N user -P password all "Aug 28 00:00 2008" "Dec 31 23:59:59 2008"
```

for the following leases:

Lease 1	Begin	Jan 01 2008	End	Jun 30 2008
Lease 2	Begin	Mar 10 2008	End	Sep 01 2008
Lease 3	Begin	Jun 01 2008	End	Sep 30 2008
Lease 4	Begin	Jan 01 2009	End	Mar 10 2009

would return just Lease 2 and Lease 3, because they both end after the specified start date of the query, even though they both begin before that date. The other two are out of range, because they either end before the specified start date or begin after the specified end date of the query.

The values on each line depend on the specific lease object that the DHCP server stores. You can specify the values to include using the **iphist -f format** command.

The *format* argument is a list of lease attribute names, enclosed in quotation marks with the names separated by commas, that provides the template for the output lines. The default output is *ipaddress, client-mac-addr, binding-start-time, binding-end-time*.

For example:

```
# ./iphist -f "address,client-mac-addr,binding-start-time,binding-end-time" all
```

The output is a sequence of lines terminated with a newline sequence appropriate to the operating system (\n on UNIX or \r\n on Windows). Each line contains data on a single lease record. The format of the lines is generally comma-separated values enclosed in quotation marks. To use a literal backslash (\) or quotation mark (") inside quotation marks, precede each with a single backslash (\). New lines in attributes are printed as \n.

The table below lists some of the common lease object attributes you can include in the output. Also, see the help for the **lease** command. To get a full list, use **iphist -a**.

Table 28: IP History Query Output Attributes

Lease Attribute	Description
address	IP address of the lease.
binding-start-time	Start time of the lease binding.
binding-end-time	End time of the lease binding.
client-binary-client-id	Binary form of the client MAC address.
client-dns-name	Latest DNS name of the client known by the DHCP server.

Lease Attribute	Description
client-domain-name	Domain where the client resides.
client-flags	A number of client flags.
client-host-name	Hostname that the client requested.
client-id	Client ID requested by or synthesized for the client.
client-last-transaction-time	Date and time when the client most recently contacted the server.
client-mac-addr	MAC address that the client presented to the DHCP server.
client-os-type	Operating system of the leased client.
expiration	Date and time when the lease expires.
flags	Either reserved or deactivated.
lease-renewal-time	Minimal time that the client is expected to issue a lease renewal.
relay-agent-circuit-id	Contents of the <i>circuit-id</i> suboption (1).
relay-agent-option	Contents of the option from the most recent client interaction.
relay-agent-remote-id	Contents of the <i>remote-id</i> suboption (2).
relay-agent-server-id-override	IP address in the <i>server-id-override</i> suboption.
relay-agent-subnet-selection	IP address in the <i>subnet-selection</i> suboption.
relay-agent-vpn-id	Contents of the <i>vpn-id</i> suboption.
start-time-of-state	Date and time when the lease changed its state.
state	One of available, expired, leased, offered, or unavailable.
vendor-class-id	Vendor class ID requested by the client.
vpn-id	Identifier for the VPN, if any.

Trimming Lease History Data

If you enabled IP history trimming at the regional cluster, the IP history database is automatically trimmed so that you can reclaim disk space. Each history record has an expiration time. Trimming is necessary for the DHCP server itself, as well as for the CCM regional server that polls the DHCP server for history data.

The CCM server performs background trimming at the regional cluster, which trims off the lease history data older than a certain age at regular intervals. The trimming interval is set by default to 24 hours, and the age (how far back to go in time before trimming) to 24 weeks. The DHCP server at the local cluster performs daily automatic trimming (at 3:00 A.M. local time), and stores four weeks of data by default.

Regional Web UI

To trim lease history data, you must be a central configuration administrator:

-
- Step 1** From the **Operate** menu, choose **Manage Servers** under the **Servers** submenu to open the Manage Servers page.
- Step 2** Click the **Local CCM Server** on the Manage Servers pane.
- Step 3** On the Edit Local CCM Server page, under Lease History Settings, set the following attributes (you can use the **s**, **m**, **h**, **d**, **w**, **m**, or **y** suffix with values you enter):
- ***trim-lease-hist-interval***—How often to trim the old lease history data automatically, the default being daily. If set to 0, no automatic lease trimming occurs, which is not recommended due to the increasing disk space used. The bounded values are 0 to one year.
 - ***trim-lease-hist-age***—Provided that the *trim-lease-hist-interval* is not set to 0, how far back in time to trim the old lease history data automatically, the default being 24 weeks. The bounded values are one day to one year.
- Step 4** To force immediate trimming, at the bottom of the page find the Trim/Compact Inputs section (compacting is available only for subnet utilization data). Set the Trim/Compact age to a desired value. This age is how far in time to go back to trim the lease history data. There are no bounds to this value. However, if you set a very small value (such as 1m), it trims or compacts very recent data, which can be undesirable. In fact, if you set it to zero, you lose all of the collected data. Setting the value too high (such as 10y) may end up not trimming or compacting any data.
- Step 5** If you are trimming immediately, click **Trim All Lease History**.
- You can adjust the trimming that the DHCP server itself performs by setting the *ip-history-max-age* attribute. If *ip-history* is set, the DHCP server accumulates database records over time as lease bindings change. This parameter establishes a limit on the age of the history records kept in the database. The server periodically examines the lease history records, establishes an age threshold based on this parameter, and deletes any records that represent bindings that ended before the threshold. The preset value is four weeks.
-

Running Lease Utilization Reports

Lease utilization reports show the current utilization of address blocks, subnets, and scopes. For both user interfaces, see [Generating Subnet Utilization History Reports, on page 105](#).

Local Advanced Web UI

View the current utilization for address blocks, subnets, and scopes from pages in the Address Space function.

CLI Commands

To view lease utilization reports, use **report**.

Receiving Lease Notification

The CLI provides the feature of sending notifications if the number of available IP addresses equals or falls below a certain threshold. The **lease-notification** command specifies, through an *available* attribute, when the notification should occur if the number of available leases reaches or falls below a certain threshold. You

can e-mail the report to a user. Although you can use the command interactively, its primary use is in an automated procedure such as a UNIX **cron** task or Windows Scheduled Task.

The following example sets up lease notification for examplescope for when its free addresses fall to 10%. It sends the report to recipients billy, joe, and jane, on a specific Windows mail host:

```
nrcmd> lease-notification available=10% scopes=examplescope recipients=billy,joe,jane
mail-host=mailhost
```

The output consists of an explanatory header, a table containing a row for each scope in which the number of free addresses is equal to or less than the threshold, and possible warnings related to the scopes and clusters requested.

Cisco Prime Network Registrar uses the default cluster and the `.nrconfig` file by default, unless you specify otherwise. For the command syntax, see the help for the **lease-notification** command.

Related Topics

[Running Lease Notification Automatically in Linux, on page 230](#)

[Running Lease Notification Automatically in Windows, on page 231](#)

[Specifying Configuration Files for Lease Notification, on page 231](#)

Running Lease Notification Automatically in Linux

You can run **lease-notification** periodically by means of the **cron(1)** command by supplying **crontab(1)** with the command to run.

This example, specified to **crontab**, runs **lease-notification** at 00:15 and 12:15 (15 minutes after midnight and noon), Monday through Friday (note that this encompasses a single command line):

```
15 0,12 * * 1-5 . .profile; /opt/nwreg2/local/usrbin/nrcmd lease-notification available=10%\%
config=/home/jsmith/.nrconfig addresses=192.32.1.0-192.32.128.0
recipients=jsmith,jdoe@example.com >/dev/null 2>&1
```

You can perform **crontab** editing by running the UNIX **crontab -e** command. Set your EDITOR environment variable before running the command, unless you want to use **ed(1)**. See the **crontab(1)** man page for additional details.

Note that you must supply the full path of the CLI command on the **crontab** command line. You can determine the full path in your environment with the UNIX **which nrcmd** command.

Also, when you run the **lease-notification** *command* by means of **crontab**, the **nrcmd** command ignores the user environment variables **CNR_CLUSTER**, **CNR_NAME**, and **CNR_PASSWORD**. Because other viewers can view the command being run, do not provide the password through the **-P** option on the command line, for security reasons.

Supply the cluster name, user, and password information for the cluster you want the **nrcmd** command to run from in a `.profile` or other file in the home directory of the user running **crontab -e**. For example:

```
CNR_CLUSTER=host1
export CNR_CLUSTER
CNR_NAME=admin1
export CNR_NAME
CNR_PASSWORD=passwd1
export CNR_PASSWORD
```

The **.profile** specification in the **crontab** entry explicitly reads the file. The first dot (.) is the shell command that reads the file and you must follow it with at least one space character. For notification on a different cluster (or clusters) than where **nrcmd** is running, specify this information:

- Clusters to check in a config file (see [Specifying Configuration Files for Lease Notification, on page 231](#)).
- Fully specified path as in the sample **crontab** entry at the beginning of this section.

You can prevent others from examining or changing the contents of the **.profile** and the configuration file that you create by changing its permissions with the **chmod go-rwx config-file** UNIX command.

Running Lease Notification Automatically in Windows

Use the Scheduled Tasks service available in Windows Explorer under My Computer to schedule the **lease-notification command**. If you do not find a Scheduled Tasks folder under My Computer, you need to add this optional component from Microsoft Internet Explorer 4.0 or later, or use some third-party task scheduler. You can also use the **at** command to schedule the **nrcmd lease-notification command**. Put multiple entries in the **at** queue, one for each time of day at which you want to run the job.

Specifying Configuration Files for Lease Notification

If you omit a configuration file, **lease-notification** looks for a default **.nrconfig** file in your current directory, then in your home directory, and finally in the **CNR_INSTALL_PATH/conf** directory. Cisco Prime Network Registrar uses the first file it encounters. Each line of the file must either begin with the character **#** (comment), a section header enclosed in square brackets, or a parameter/value pair or its continuation. Cisco Prime Network Registrar strips leading space characters from each line and ignores blank lines.

Dynamic Lease Notification

The DHCPv4 and DHCPv6 dynamic lease notification feature allows an external client application to receive updates about the IP address binding activity of the DHCP server. This feature can be used to update an external database with lease activity or trigger actions, such as lawful intercept, when specific lease activity takes place.



Note Dynamic Lease Notification provides only the current lease state information. It does not guarantee that all the lease state changes are reported. Lease state changes are lost under certain conditions, such as when the connection to the DHCP server is down or congested.

The dynamic lease notification feature extends the DHCP server to support additional capabilities and includes a sample client (written in Java), which demonstrates the features by storing the lease state information into a MySQL database.

Using Dynamic Lease Notification

To use Dynamic Lease Notification:

1. You must create a **dhcp-listener** object on the local cluster. The **dhcp-listener** object specifies the port on which the server listens for incoming TCP connections and other attributes for these connections (see

[DHCP Listener Configuration, on page 237](#)). You must reload the DHCP server after creating the `dhcp-listener` object.

2. A dynamic lease notification client must establish a TCP connection with the DHCP server and make any of these requests:
 - Bulk leasequery—This request is made to obtain the current state of all leases in the DHCP server that have changed state since a specific point in time. The current state of all leases is sent when no time is specified (or zero is specified for the time). This is similar to the UDP-based DHCPv4 leasequery (RFC 4388) and DHCPv6 leasequery (RFC 5460), except that the DHCP server delivers all leases to the client in response to a single request. Typically, a bulk leasequery is used to initialize an external database. It is also used to bring that database up to date after some interruption of an active leasequery, where the catch-up time was too great for the active leasequery to return the missed data.
 - Active leasequery—This request is made to obtain lease state information for all future significant lease changes that the DHCP server will make. When the DHCP server writes significant lease state information to its database, the lease state information will be sent over the TCP connection.
 - Active leasequery with catchup—This request is made to obtain future lease state changes and the latest data from recently changed leases. It allows the dynamic lease notification client to retrieve the latest data on recently changed leases that were missed during a short period of connection loss, such as during a restart of the dynamic lease notification client or DHCP server. The active leasequery with catchup fetches only the current state of a lease; it does not fetch the data on all intermediate lease state changes that might have been missed.

The DHCP server sends the lease state information to the dynamic lease notification client in a stream of leasequery messages. For a bulk leasequery, the lease state information is sent as soon as the DHCP server has time for processing. For an active leasequery, the lease state information is sent as lease state changes occur. The dynamic lease notification client can process these messages to take appropriate actions such as updating its database.



Note

While the DHCP server supports multiple dynamic lease notification clients, it is recommended to keep the number of clients to a minimum as multiple clients can impact the DHCP server's leasing performance.

In a failover configuration, only the active failover partner which interacts with the DHCP client sends dynamic lease notification updates to the dynamic lease notification clients with an active leasequery request. Therefore, to receive complete information, a dynamic lease notification client must connect to both the failover partners.

The server determines whether a lease is queued for active leasequery notifications based on the *leasequery-send-all* attribute of the `dhcp-listener`. If this attribute is enabled, the DHCP server always sends a notification to an active leasequery client. If this attribute is disabled or unset, the DHCP server only sends notifications which are necessary to maintain accurate state in the active leasequery client.

You can also control the leasequery notifications using extensions. Extensions can decide whether a lease is queued for active leasequery notifications using the *active-leasequery-control* request and response data dictionary items as described in [Using Extension Points, on page 351](#).

Sample Lease Notification Client

Cisco Prime Network Registrar provides a standalone sample Java client. The standalone sample Java client collects the lease state data from one or more DHCP servers, and updates the SQL database with the most

current lease data. The sample Java client is designed to accept lease state updates from both failover partners and ensures that the latest lease state information is in the SQL database (even when updates are received out of order). If you use the sample Java client, it is not necessary to know the complete details of the bulk and active leasequery protocols. The sample Java client sources are provided; thus if the sample Java client does not meet your needs, it is recommended you modify it rather than implementing your own.

The sample Java client performs a bulk leasequery when it connects to a server for the first time to obtain the state of all leases. If the sample Java client has communicated with the server before, it attempts an active leasequery with catchup. The sample Java client performs a bulk leasequery only if the active leasequery with catchup indicates that catch-up data is not available, such as if the client was down for a while or the DHCP server was reloaded.

The sample Java client supports configurations with multiple VPNs and multiple servers. However, the sample Java client assumes that the leases across these servers are unique with respect to VPN and IP address. If two servers lease out the same IP address in a VPN or global namespace, the SQL database will contain a record of only one of the two leases. This does not apply to failover pairs, but rather to two independent DHCP servers. The sample Java client must also be configured to communicate with both the failover partners of a failover-pair to keep the SQL database up-to-date.



Note The sample Java client is available at *install-path/examples/dhcp/cnrnotify.jar*. A text readme file named *cnrnotify-readme.txt* file is also provided in that directory and must be read first.

The *examples/dhcp/cnrnotify.jar* is a zip file, which contains:

- The sample Java client source code and Javadoc documentation.
- For example *inc.properties* and *inc6.properties* files. (Run the client with *-listprops* option for details on the available properties.)
- The Bulk and Active Leasequery Internet Drafts for the Cisco Prime Network Registrar implementation.
- A document that details the message values, option codes, and vendor-specific data used for Cisco Prime Network Registrar proprietary lease information. As the Internet Assigned Numbers Authority (IANA) has not yet assigned values to the messages and option codes used by the Bulk and Active Leasequery Internet Drafts, this document describes the values that are used in the Cisco Prime Network Registrar.

To extract these items, open the *cnrnotify.jar* file using a zip tool such as Winzip. (See the *cnrnotify-readme.txt* file.) To extract the Javadoc, we recommend you use:

```
jar xvf cnrnotify.jar docs_notify
```

The above command is used to extract the documentation.

DHCPv4 Sub-sub Option Codes

Following table lists the sub-sub option codes used while requesting for DHCPv4 leasequery. These codes are present in the *cnrnotify-protocol-numbers.txt* file which is available in the *cnrnotify.jar* zip file.

Table 29: DHCPv4 Sub-sub Option Codes

Sub-sub Option Code	Option Name	Option Type
1	oro	one or more bytes of sub-sub option numbers
2	dhcp-state	byte

Sub-sub Option Code	Option Name	Option Type
3	data-source	byte
4	start-time-of-state	duration in past from base-time
5	base-time	absolute time (secs from 1970)
8	client-class-name	string (without zero termination)
9	partner-last-transaction-time	duration in past from base-time
10 0xa	client-creation-time	duration in past from base-time
11 0xb	limitation-id	blob containing limitation-id
12 0xc	binding-start-time	duration in past from base-time
13 0xd	binding-end-time	negative/positive value representing duration in future/past from base-time
14 0xe	fwd-dns-config-name	string (without zero termination)
15 0xf	rev-dns-config-name	string (without zero termination)
16 0x10	client-override-client-id	blob containing client-id for client
17 0x11	user-defined-data	string (without zero termination)
18 0x12	scope-name	string (without zero termination)
19 0x13	failover-state-serial-number	4 byte integer, network order
20 0x14	reservation-key	blob, starting with type byte: <ul style="list-style-type: none"> • 0x2e, 46: string without zero termination • 0x7, 7: blob
21 0x15	client-prl	client's parameter request list, blob of DHCPv4 option code

DHCPv6 Sub-sub Option Codes

Following table lists the sub-sub option codes used while requesting for DHCPv6 leasequery. These codes are also present in the `cnnotify-protocol6-numbers.txt` file which is available in the `cnnotify.jar` zip file.

Table 30: DHCPv6 Sub-sub Option Codes

Sub-sub Option Code	Option Name	Option Type
1	oro	one or more bytes of sub-sub option numbers
2	dhcp-state	byte
3	data-source	byte

Sub-sub Option Code	Option Name	Option Type
4	start-time-of-state	duration in past from base-time
5	base-time	absolute time (secs from 1970)
8	client-class-name	string (without zero termination)
9	partner-last-transaction-time	duration in past from base-time
10 0xa	client-creation-time	duration in past from base-time
12 0xc	binding-start-time	duration in past from base-time
13 0xd	binding-end-time	negative/positive value representing duration in future/past from base-time
14 0xe	fwd-dns-config-name	string (without zero termination)
15 0xf	rev-dns-config-name	string (without zero termination)
16 0x10	lookup-key	blob containing client-id for client
17 0x11	user-defined-data	string (without zero termination)
18 0x12	prefix-name	string (without zero termination)
19 0x13	failover-state-serial-number	4 byte integer, network order
20 0x14	reservation-key	blob, starting with type byte: <ul style="list-style-type: none"> • 0x2e, 46: string without zero termination • 0x7, 7: blob
21 0x15	failover-partner-lifetime	negative/positive value representing duration in future/past from base-time
22 0x16	failover-next-partner-lifetime	negative/positive value representing duration in future/past from base-time
23 0x17	failover-expiration-time	negative/positive value representing duration in future/past from base-time
24 0x18	client-oro	client's ORO, blob of DHCPv6 two byte option codes

Requirements for Sample Java Client

The requirements for the sample Java client are:

- JDK 1.6 or later.
- The java.sql package from JDK 1.6 or later.
- Installation of a JDBC driver and a compatible database. A specific table (that contains a pre-defined set of columns) must exist in the database.



Tip If the tables do not exist, run the client with `-c` option. The tables are thus created.

The requirements for MySQL are:

- The latest version of MySQL server.
- The JDBC connector for MySQL.
- The log4j package for logging the sample Java client status and errors.



Note We recommend that you use MySQL-5.5.23 database, `mysql-connector-java-5.1.19.jar` and `log4j-1.2.16.jar`.

Once extracted and the `lnc.properties` file is configured, the sample Java client can be run using:

Step 1 Place all three `.jar` files (`cnrnotify.jar`, `mysql-connector-java-5.1.19.jar`, `log4j-1.2.16.jar`) in same directory.

Step 2 Extract `lnc.properties`/`lnc6.properties` file in same directory:

For DHCPv4 client:

```
jar xvf cnrnotify.jar com/cisco/cnr/notify/lnc.properties
```

For DHCPv6 client:

```
jar xvf cnrnotify.jar com/cisco/cnr/notify/lnc6.properties
```

Step 3 Configure `lnc.properties`/`lnc6.properties` file.

Step 4 Assuming Java executable directory is in the current path, the sample client is run by:

For DHCPv4:

```
java -cp ../cnrnotify.jar:mysql-connector-java-5.1.19-bin.jar:log4j-1.2.16.jar
com/cisco/cnr/notify/LeaseNotificationClient
```

For DHCPv6:

```
java -cp ../cnrnotify.jar:mysql-connector-java-5.1.19-bin.jar:log4j-1.2.16.jar
com/cisco/cnr/notify/LeaseNotificationClient6
```

Local Basic or Advanced Web UI

The web UI displays and manages the configuration attributes, and displays the related servers' information. The statistics about the lease queries are available on the DHCP Server Statistics page.

Step 1 From the **Deploy** menu, choose **DHCP Server** under the **DHCP** submenu to open the Manage DHCP Server page.

Step 2 Click the **Statistics** tab to open the DHCP Server Statistics page.

The Server Statistics details are displayed in this page.

CLI Command

The existing nrcmd **dhcp getRelatedServers** command is extended to supply information about the DHCP listeners and any active connections.

```
nrcmd> dhcp getrelatedservers
```



Note You can use this command only on a local cluster.

DHCP Listener Configuration

Using DHCP Listener Configuration, you can configure objects to enable active and bulk leasequery to the DHCP server over TCP connections. A single object is sufficient, unless you want the DHCP server to support listening for connections on multiple TCP ports or need to restrict the addresses on which the server will accept incoming connections.

Local Advanced Web UI

- Step 1** From the **Deploy** menu, choose **Listeners** under the **DHCP** submenu, to open the List/Add DHCP TCP Listener page.
- Step 2** Click the **Add Listeners** icon in the Listeners pane, enter a name in the Name field, then click Add TCP Listener.
- Step 3** Enter an IP address in the address/ip6address field, to restrict the interface over which the server will accept connections. This is usually unspecified. If you want to configure a IPv6 listener, then enter ip6address. If both address and ip6address are not specified, then the IPv4 address 0.0.0.0 is used.

To restrict the address on which TCP connections are accepted, enter the address in the address (for IPv4) or ip6address (for IPv6) attribute. If no value is entered in either attribute, IPv4 connections to any IPv4 address of the host are accepted. To specify connections over IPv6, you must enter a value in the ip6address attribute (0::0 can be used to accept connections to any IPv6 address of the host). You can only enter a value in either, not both, attributes.

Note You cannot specify both IPv4 and IPv6 listeners for a DHCP server.
- Step 4** Enter a value for the port in the Port field, if the default value is not appropriate. The default port is the server-port for DHCPv4 and DHCPv6-server-port for DHCPv6.
- Step 5** For Enable attribute, click true or false radio button. The default value is true.
- Step 6** Enter a value for Max-connections, if the default value 10 is not appropriate.
- Step 7** Enter a value for Leasequery-backlog-time, if the default value 120 is not appropriate.
- Step 8** For leasequery-send-all attribute, click true or false radio button. The default value is false.
- Step 9** Click **Save**.

CLI Commands

The DHCP Listener commands are shown in the table below.

Table 31: DHCP Listener Commands

Action	Command
Create	dhcp-listener <name> create [<attribute>=<value>]
Delete	dhcp-listener <name> delete
List	dhcp-listener list
List the names	dhcp-listener listnames
Show	dhcp-listener <name> show
Set	dhcp-listener <name> set <attribute>=<value> [<attribute>=<value> ...]
Get	dhcp-listener <name> get <attribute>
Unset	dhcp-listener <name> unset <attribute>
Enable	dhcp-listener <name> enable <attribute>
Disable	dhcp-listener <name> disable <attribute>

Lease History Database Compression Utility

The `cnr_leasehist_compress` utility was added to Cisco Prime Network Registrar to compress regional cluster (DHCPv4) lease history databases. This utility does not compress the data directly in the databases, but copies the existing data into new databases that are optimized to be as compact as possible. You can download this utility from the Cisco Prime Network Registrar download section on the Cisco website.



Caution Use the `cnr_leasehist_compress` utility only with the regional cluster lease history database, and when you suspect that the database grew significantly, particularly because of DHCPRELEASE packets.

During the copy operation, you can use this utility to:

- Trim records that are older than a certain interval of time—You would typically use the `-t` option. The interval that you specify with this option uses the Network Registrar time interval format; for example, **30d** for 30 days or **1y** for 1 year.
- Merge records that belong to the same lease and client—You use the `cnr_leasehist_compress` utility to merge records that belong to clients who have reclaimed the lease on an IP address after releasing it. You would typically use the `-m` option. The interval that you specify with this option uses the Network Registrar time interval format; for example, **120s** for 120 seconds or **2m** for 2 minutes.

While merging records, the utility also corrects lease history records that were terminated abruptly or have an incorrect binding end time (that may have resulted from a subsequent lease operation). This option of merging records also addresses the vast number of records that are created by certain router configurations that introduce an additional load on the servers.

Before you run the `cnr_leasehist_compress` utility:

- Stop the Network Registrar regional cluster; it does not operate on an active regional cluster database.
- Note that you can use it to compress existing lease history data alone; it does not alter how the regional cluster collects future lease history records. If you suspect chatty clients, ensure that the DHCP server does not process DHCPRELEASE messages, because this results in rapid growth of lease history data. In such instances, you may need to run the utility periodically.
- Note that you can use it if you are a service provider and suspect that the regional lease history in your network grew because some devices have known issues, such as repeatedly generating a sequence of DHCPDISCOVER, DHCPDISCOVER, DHCPDISCOVER, DHCPDISCOVER, and after 30 seconds, DHCPRELEASE messages. You can choose to drop all DHCPRELEASE messages or those that belong to clients that exceed a configured threshold.
- Note that it writes the new database in the most optimal manner. The new database can initially grow at a considerable rate, but it tapers back to normal after the additional lease history records are collected.

General Comments on Running `cnr_leasehist_compress`



Caution

Follow every step in this procedure carefully. If you skip any step, you might lose lease history data. Note the lease history database that each task involves. Depending on the number of lease history records and the time taken to trim or merge the records, this utility may take several hours or days to run. You can interrupt the utility while it is running if the server reboots before the run is completed. You can resume it later; however, you must specify the same options that you have used in the previous run.

The *install-path* is the path in which you install Network Registrar.

The table below describes the qualifying options for the `cnr_leasehist_compress` utility.

Table 32: `cnr_leasehist_compress` Options

Option	Description
<code>-a</code>	Appends all the lease history records in the temporary active database to those in the new database.
<code>-c limit</code>	Generates a report when more than a specified number of records (<i>limit</i>) merged for a client. When used with the <code>-f</code> option, these records are transferred to a log file.
<code>-C</code>	Compresses lease records on write (for details, see CCM's <i>lease-hist-compression</i> attribute.
<code>-d path</code>	Specifies the path to a new destination database that contains the compressed lease history records.
<code>-e attrlist</code>	Overwrites the excluded merge attribute list.
<code>-f file</code>	Redirects most lease history record warnings to a log file.
<code>-g</code>	Uses the <code>dbtxn-seq</code> and <code>dbtxn-generation</code> attributes to generate a new sequence in the numbers that are assigned to all lease history records, which are written into the destination database.
<code>-i ipaddr</code>	Transfers the records of a particular IP address to a log file.
<code>-l limit</code>	Purges log files after the database reaches the preset limit of 20 files.

Option	Description
<code>-m time-int</code>	Merges lease records where the binding-start-time of a particular lease falls in the duration of the binding-end-time of the previous lease. The recommended value for this option is 120s .
<code>-n</code>	Compares adjacent records without merging them.
<code>-p</code>	Drops detailed lease history records. You can use this option only if you have enabled detailed lease history.
<code>-q records</code>	Sets an interval for a periodic progress report that is generated while the utility runs. The default value is 100000 . For example: +00:00:18 Read 100000 records (0 bad); trimmed 6717; merged 73370; 19912 written (19.91%)
<code>-r records</code>	Limits the number of records that are read from the source database.
<code>-s path</code>	Specifies the source database from where the data is copied to a new database.
<code>-t age</code>	Specifies a value for trimming records that are older than a certain interval of time. Use the standard Network Registrar time interval for this option, such as 1y for 1 year or 30d for 30 days.
<code>-v</code>	Emits the version and exits.
<code>-w records</code>	Limits the number of records that are written into the destination database.
<code>-y "line attr"</code>	Alters the width of the dump of lease history records. This option is not recommended; however, you can use the value 132 30 for a 132-column output.
<code>-z {letters}=level</code>	Debugs the database, specified by using the standard Network Registrar debug tracing syntax.

Running Compression on Linux

To run the `cnr_leasehist_compress` utility on Linux, do the following:

Step 1 Add `install-path/lib` to the `LD_LIBRARY_PATH` to provide the utility with access to the Network Registrar libraries:

```
$ bash
# export LD_LIBRARY_PATH=install-path/lib:$LD_LIBRARY_PATH
```

Step 2 Stop the Network Registrar regional cluster:

- RHEL/CentOS 6.x:
`/etc/init.d/nwregregion stop`
- RHEL/CentOS 7.x:
`systemctl stop nwregregion`

Step 3 Rename the original *install-path* /data/leasehist directory as *install-path* /data/oldleasehist. The /leasehist directory becomes the original database:

```
# mv install-path/data/leasehist \  
# install-path/data/oldleasehist
```

Step 4 Create a new leasehist directory, which becomes the temporary active database:

```
# mkdir install-path/data/leasehist
```

Step 5 Run the **cnr_leasehist_compress** utility to allow the regional cluster to resume activity:

```
# install-path/bin/cnr_leasehist_compress \  
> -r 0 \  
> -s install-path/data/oldleasehist \  
> -d install-path/data/leasehist \  
> -p
```

Caution Running these commands does not compress the original database. The **-r 0** option is critical as it instructs the utility to create the temporary active database. The regional cluster remains active while the utility compresses the original database.

Step 6 Restart the Network Registrar regional cluster.

- RHEL/CentOS 6.x:
/etc/init.d/nwregregion start
- RHEL/CentOS 7.x:
systemctl start nwregregion

You cannot, however, obtain lease history data from the original database at this time. The regional cluster collects new lease history data and transfers it to the temporary active database. The utility then merges the new lease history data into the new database.

Step 7 Create a new directory called *install-path* /data/newleasehist. This /newleasehist directory becomes the new lease history database:

```
# mkdir install-path/data/newleasehist
```

Tip After the regional cluster populates the new database, you can optionally create this new directory on a different partition and copy it to the final location.

Step 8 Run the **cnr_leasehist_compress** utility to trim, merge, and compress the original database into the new database:

```
# install-path/bin/cnr_leasehist_compress \  
> -s install-path/data/oldleasehist \  
> -d install-path/data/newleasehist \  
> -t trim-time-interval \  
> -m merge-time-interval \  
> -f /tmp/cnr-compress.log
```

If the original database contains any detailed lease history records, you must use the **-p** option to acknowledge that it is acceptable for the utility to not transfer these records into the new database. Otherwise, the utility does not run.

Step 9 Perform the following tasks to append any fresh lease history records to the new database after the utility processes the entire original database.

Note Do not restart the regional cluster until you have completed the following procedure. If the system reboots during the following procedure, repeat these steps.

a) Stop the Network Registrar regional cluster:

- RHEL/CentOS 6.x:
`/etc/init.d/nwregregion stop`
- RHEL/CentOS 7.x:
`systemctl stop nwregregion`

b) Run the `cnr_leasehist_compress` utility to append new lease history records to the new database:

```
# install-path/bin/cnr_leasehist_compress \  
> -a \  
> -s install-path/data/leasehist \  
> -d install-path/data/newleasehist \  
> -m merge-time-interval \  
> -f /tmp/cnr-append.log
```

Caution The `-a` option is critical as it indicates that the utility should append the lease history records in the temporary active database to those in the new database. We recommend that you use the same `merge-time-interval` value that you used for the original database.

c) After the utility completes the task of appending the newly collected lease history records, rename the temporary active database directory, `install-path /data/leasehist`, as `install-path /data/tmpleasehist`:

```
# mv install-path/data/leasehist \  
# install-path/data/tmpleasehist
```

d) Rename the new database directory, `install-path /data/newleasehist`, as `install-path /data/leasehist`:

```
# mv install-path/data/newleasehist \  
# install-path/data/leasehist
```

Step 10 Start the Network Registrar regional cluster:

- RHEL/CentOS 6.x:
`/etc/init.d/nwregregion start`
- RHEL/CentOS 7.x:
`systemctl start nwregregion`

Step 11 Verify the regional lease history data by using the Network Registrar web UI.

Step 12 Archive the original database, in `install-path /data/oldleasehist`, and the temporary active database, in `install-path /data/tmpleasehist`. Ensure that you include all subdirectories and files when you archive the database.

Step 13 Delete the original database and temporary active database:

```
# rm -rf install-path\data\oldleasehist
# rm -rf install-path\data\tmpleasehist
```

Running Compression on Windows

To run the `cnr_leasehist_compress` utility on Windows, do the following:

Step 1 Stop the Network Registrar regional cluster:

```
> net stop nwregregion
```

Step 2 Rename the original `install-path\data\leasehist` directory as `install-path\data\oldleasehist`. This leasehist directory becomes the original database:

```
> rename install-path\data\leasehist install-path\data\oldleasehist
```

Tip You can move the original database to a different partition. Ensure that you copy the entire original `/leasehist` directory (including all its subdirectories and files) before you remove it.

Step 3 Create a new leasehist directory, this new leasehist directory becomes the temporary active database:

```
> mkdir install-path\data\leasehist
```

Step 4 Run the `cnr_leasehist_compress` utility to allow the regional cluster to resume activity:

```
> install-path\cnr_leasehist_compress -r 0 -s install-path\data\oldleasehist
-d install-path\data\leasehist -p
```

Caution Running these commands, however, does not compress the original database. The `-r 0` option is critical as it instructs the utility to create the temporary active database. The regional cluster remains active while the utility compresses the original database.

Step 5 Restart the Network Registrar regional cluster. However, you cannot obtain lease history data from the original database at this time. The regional cluster collects any new lease history data and transfers it to the temporary active database. The utility then merges the new lease history data into the new database:

```
> net start nwregregion
```

Step 6 Create a new directory called `install-path\data\newleasehist`. This newleasehist directory becomes the new lease history database:

```
> mkdir install-path\data\newleasehist
```

Tip After the regional cluster populates the new database, you can optionally create this new directory on a different partition and copy it to the final location.

Step 7 Run the `cnr_leasehist_compress` utility to trim, merge, and compress the original database into the new database:

```
> install-path\cnr_leasehist_compress -s install-path\data\oldleasehist
-d install-path\data\newleasehist -t trim-time-interval
-m merge-time-interval
-f c:\temp\cnr-compress.log
```

If the original database contains any detailed lease history records, you must use the `-p` option to acknowledge that it is acceptable for the utility to not transfer these records into the new database. Otherwise, the utility does not run.

Step 8

Perform the following tasks to append any fresh lease history records to the new database after the utility processes the entire original database.

Note Do not restart the regional cluster until you have completed the following procedure. If the system reboots during the following procedure, repeat these steps:

- a) Stop the Network Registrar regional cluster:

```
> net stop nwregregion
```

- b) Run the `cnr_leasehist_compress` utility to append new lease history records to the new database:

```
> install-path\cnr_leasehist_compress -a -s install-path\data\leasehist
-d install-path\data\newleasehist -m merge-time-interval
-f c:\temp\cnr-append.log
```

Caution The `-a` option is critical as it indicates that the utility should append the lease history records in the temporary active database to those in the new database. We recommend that you use the same `merge-time-interval` value that you used for the original database.

- c) After the utility completes the task of appending the newly collected lease history records, rename the temporary active database directory, `install-path\data\leasehist`, as `install-path\data\tmpleasehist`:

```
> rename install-path\data\leasehist
install-path\data\tmpleasehist
```

- d) Rename the new database directory, `install-path\data\newleasehist`, as `install-path\data\leasehist`:

```
> rename install-path\data\newleasehist install-path
\data\leasehist
```

Step 9

Start the Network Registrar regional cluster:

```
> net start nwregregion
```

Step 10

Verify the regional lease history data by using the Network Registrar web UI.

Step 11

Archive the original database, in `install-path\data\oldleasehist`, and the temporary active database, in `install-path\data\tmpleasehist`. Ensure that you include all subdirectories and files when you archive the database.

Step 12

Delete the original database and temporary active database:

```
> del/s install-path\data\oldleasehist
> del/s install-path\data\tmpleasehist
```



CHAPTER 9

Managing DNS Update

The DNS Update protocol (RFC 2136) integrates DNS with DHCP. The latter two protocols are complementary; DHCP centralizes and automates IP address allocation, while DNS automatically records the association between assigned addresses and hostnames. When you use DHCP with DNS update, this configures a host automatically for network access whenever it attaches to the IP network. You can locate and reach the host using its unique DNS hostname. Mobile hosts, for example, can move freely without user or administrator intervention.

This chapter explains how to use DNS update with Cisco Prime Network Registrar servers, and its special relevance to Windows client systems.

- [DNS Update Process, on page 245](#)
- [DNS Updates for DHCPv6, on page 246](#)
- [Configuring Access Control Lists and Transaction Security, on page 249](#)
- [Transaction Security, on page 251](#)
- [GSS-TSIG, on page 253](#)
- [Creating DNS Update Configurations, on page 256](#)
- [Configuring DNS Update Policies, on page 258](#)
- [Creating DNS Update Maps, on page 263](#)
- [Confirming Dynamic Records, on page 264](#)
- [Scavenging Dynamic Records, on page 264](#)
- [Transitioning to DHCID RR for DHCPv4, on page 266](#)
- [Configuring DNS Update for Windows Clients, on page 267](#)
- [Configuring GSS-TSIG, on page 279](#)
- [Troubleshooting DNS Update, on page 282](#)

DNS Update Process

To configure DNS updates, you must:

1. Create a DNS update configuration for a forward or reverse zone or both. See [Creating DNS Update Configurations, on page 256](#).
2. Use this DNS update configuration in either of two ways:
 - Specify the DNS update configuration on a named, embedded, or default DHCP policy. See [Creating and Applying DHCP Policies, on page 166](#).

- Define a DNS update map to autoconfigure a single DNS update relationship between a Cisco Prime Network Registrar DHCP server or failover pair and a DNS server or High-Availability (HA) pair. Specify the update configuration in the DNS update map. See [Creating DNS Update Maps, on page 263](#)
3. Optionally define access control lists (ACLs) or transaction signatures (TSIGs) for the DNS update. See [Configuring Access Control Lists and Transaction Security, on page 249](#).
 4. Optionally create one or more DNS update policies based on these ACLs or TSIGs and apply them to the zones. See [Configuring DNS Update Policies, on page 258](#).
 5. Optionally configure the DNS update to transition from TXT RR to DHCID RR for DHCPv4. See [Transitioning to DHCID RR for DHCPv4, on page 266](#).
 6. Adjust the DNS update configuration for Windows clients, if necessary; for example, for dual zone updates. See [Configuring DNS Update for Windows Clients, on page 267](#).
 7. Configure DHCP clients to supply hostnames or request that Cisco Prime Network Registrar generate them.
 8. Reload the DHCP and DNS servers, if necessary based on the edit mode.

Special DNS Update Considerations

Consider these two issues when configuring DNS updates:

- For security purposes, the Cisco Prime Network Registrar DNS update process does not modify or delete a name an administrator manually enters in the DNS database.
- If you enable DNS update for large deployments, and you are not using HA DNS (see the *"Deploying High Availability DNS Pair"* chapter in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide*), divide primary DNS and DHCP servers across multiple clusters. DNS update generates an additional load on the servers.

DNS Updates for DHCPv6

Cisco Prime Network Registrar currently supports DHCPv6 DNS update over IPv4 and IPv6. For DHCPv6, DNS update applies to nontemporary stateful addresses and delegated prefixes.

DNS Updates for Non-Temporary Stateful Addresses

DNS Updates for DHCPv6 involves AAAA and PTR RR mappings for leases. Cisco Prime Network Registrar supports server- or extension-synthesizing fully qualified domain names and the DHCPv6 *client-fqdn* option (39).

Because Cisco Prime Network Registrar is compliant with RFCs 4701, 4703, and 4704, it supports the DHCID resource record (RR). All RFC-4703-compliant updaters can generate DHCID RRs and result in data that is a hash of the client identifier (DUID) and the FQDN (per RFC 4701). Nevertheless, you can use AAAA and DHCID RRs in update policy rules.

DNS update processing for DHCPv6 is similar to that for DHCPv4 except that a single FQDN can have more than one lease, resulting in multiple AAAA and PTR RRs for a single client. The multiple AAAA RRs can be under the same name or a different name; however, PTR RRs are always under a different name, based on the lease address. RFC-4703-compliant updaters use the DHCID RR to avoid collisions among multiple clients.



Note If the DNS server is down and the DHCP server can not complete the DNS updates to remove RRs added for a DHCPv6 lease, the lease continues to exist in the AVAILABLE state. Only the same client reuses the lease.

DNS Updates for Delegated Prefixes

DNS Updates for delegated prefixes can be enabled to update AAAA and/or PTR mappings for delegated prefix leases. However, this only updates DNS for the all 0's address for the delegated prefix. For example, if a prefix of 2001:db8:3333:3333::/64 is delegated, only the PTR and/or AAAA for 2001:db8:3333:3333::0 is updated in DNS. This feature does not provide a means for doing DNS delegations for delegated prefix.

Updates for delegated prefixes are only enabled if the DNS Update Configuration has the *prefix-delegation-updates* attribute enabled. This attribute is disabled by default. Updates for delegated prefixes most likely occur to different zones than the address updates, and therefore you may need to create new DNS update configurations and associate them with the corresponding prefixes.

Since the standard name generation rules apply, a client that includes an FQDN option with a hint can impact the resulting name (if the configuration allows this). Clients are never returned the name used for prefix delegation updates if they request the FQDN option.



Note If using this feature, you **MUST** assure that both the failover partners are running a version which supports this feature. Otherwise, updates will only be done when serviced by the server that has been upgraded. Therefore, do not enable this feature until both partners have been upgraded.

Related Topics

[DHCPv6 Upgrade Considerations, on page 247](#)

[Generating Synthetic Names in DHCPv4 and DHCPv6, on page 248](#)

[Determining Reverse Zones for DNS Updates, on page 248](#)

[Using the Client FQDN, on page 249](#)

DHCPv6 Upgrade Considerations

If you use any policy configured prior to Cisco Prime Network Registrar that references a DNS update object for DHCPv6 processing (see [DHCPv6 Policy Hierarchy, on page 165](#)), after the upgrade, the server begins queuing DNS updates to the specified DNS server or servers. This means that DNS updates might automatically (and unexpectedly) start for DHCPv6 leases.



Caution If you use earlier versions of Cisco Prime Network Registrar or other DNS servers, you might experience interoperability issues for zone transfers and DNS updates, because of recent DHCID RR standards changes. You might need to upgrade DNS servers to support DHCPv6 DNS updates.

Generating Synthetic Names in DHCPv4 and DHCPv6

If clients do not supply hostnames, DHCPv4 and DHCPv6 includes a synthetic name generator. The *v6-synthetic-name-generator* attribute for the DNS update configuration allows appending a generated name to the *synthetic-name-stem* based on the:

- Hash of the client DHCP Unique Identifier (DUID) value (the preset value).
- Raw client DUID value (as a hex string with no separators).
- CableLabs *cablelabs-17* option *device-id* suboption value (as a hex string with no separators, or the hash of the client DUID if not found).
- CableLabs *cablelabs-17* option *cm-mac-address* suboption value (as a hex string with no separators, or the hash of the client DUID if not found).



Caution

Some generation methods might cause privacy issues if the domain is accessible from the Internet.

The *v4-synthetic-name-generator* attribute for the DNS update configuration allows appending a generated name to the *synthetic-name-stem* based on the:

- **address**—Identifies the v4 address of client.
- **client-id**—Client-id or DUID given by DHCPv4 client in its request (Option 61).
- **hashed-client-id**—The hashed client-id which is a 13-character base 32 encoded string formed of the right part 64-bits of the SHA-256 hash appended with the forward zone name.

See [Creating DNS Update Configurations, on page 256](#) for how to create a DNS update configuration with synthetic name generation.

In the CLI, an example of this setting is:

```
nrcmd> dhcp-dns-update example-update-config set v6-synthetic-name-generator=hashed-duid
nrcmd> dhcp-dns-update example-update-config set v4-synthetic-name-generator=client-id
```

Determining Reverse Zones for DNS Updates

The DNS update configuration uses the prefix length value in the specified *reverse-zone-prefix-length* attribute to generate a reverse zone in the ip6.arpa domain. You do not need to specify the full reverse zone, because you can synthesize it by using the ip6.arpa domain. You set this attribute for the reverse DNS update configuration (see [Creating DNS Update Configurations, on page 256](#)). Here are some rules for *reverse-zone-prefix-length*:

- Use a multiple of 4 for the value, because ip6.arpa zones are on 4-bit boundaries. If not a multiple of 4, the value is rounded up to the next multiple of 4.
- The maximum value is 124, because specifying 128 would create a zone name without any possible hostnames contained therein.
- A value of 0 means none of the bits are used for the zone name, hence ip6.arpa is used.
- If you omit the value from the DNS update configuration, the server uses the value from the prefix or, as a last resort, the prefix length derived from the *address* value of the prefix (see [Configuring Prefixes and Links, on page 130](#)).

Note that to synthesize the reverse zone name, the *synthesize-reverse-zone* attribute must remain enabled for the DHCP server. Thus, the order in which a reverse zone name is synthesized for DHCPv6 is:

1. Use the full *reverse-zone-name* in the reverse DNS update configuration.
2. Base it on the ip6.arpa zone from the *reverse-zone-prefix-length* in the reverse DNS update configuration.
3. Base it on the ip6.arpa zone from the *reverse-zone-prefix-length* in the prefix definition.
4. Base it on the ip6.arpa zone from the prefix length for the *address* in the prefix definition.

In the CLI, an example of setting the reverse zone prefix length is:

```
nrcmd> dhcp-dns-update example-update-config set reverse-zone-prefix-length=32
```

To create a reverse zone for a prefix in the web UI, the List/Add Prefixes page includes a **Create Reverse Zone** button for each prefix. (See [Creating and Editing Prefixes, on page 131](#).)

The CLI also provides the **prefix name createReverseZone** [-range] command to create a reverse zone for a prefix (from its address or range value). Delete the reverse zone by using **prefix name deleteReverseZone** [-range].

You can also create a reverse zone from a DHCPv4 subnet or DHCPv6 prefix by entering the subnet or prefix value when directly configuring the reverse zone. See the "*Configuring Primary Reverse Zones*" section in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide* for details.

Using the Client FQDN

The existing DHCP server *use-client-fqdn* attribute controls whether the server pays attention to the DHCPv6 client FQDN option in the request. The rules that the server uses to determine which name to return when multiple names exist for a client are in the following order of preference:

1. The server FQDN that uses the client requested FQDN if it is in use for any lease (even if not considered to be in DNS).
2. The FQDN with the longest valid lifetime considered to be in DNS.
3. The FQDN with the longest valid lifetime that is not yet considered to be in DNS.

Configuring Access Control Lists and Transaction Security

ACLs are authorization lists, while transaction signatures (TSIG) is an authentication mechanism:

- ACLs enable the server to allow or disallow the request or action defined in a packet.
- TSIG ensures that DNS messages come from a trusted source and are not tampered with.

For each DNS query, update, or zone transfer that is to be secured, you must set up an ACL to provide permission control. TSIG processing is performed only on messages that contain TSIG information. A message that does not contain, or is stripped of, this information bypasses the authentication process.

For a totally secure solution, messages should be authorized by the same authentication key. For example, if the DHCP server is configured to use TSIG for DNS updates and the same TSIG key is included in the ACL for the zones to be updated, then any packet that does not contain TSIG information fails the authorization step. This secures the update transactions and ensures that messages are both authenticated and authorized before making zone changes.

ACLs and TSIG play a role in setting up DNS update policies for the server or zones, as described in [Configuring DNS Update Policies, on page 258](#).

Related Topics

[Assigning ACLs on DNS Caching Servers or Zones](#), on page 250

[Configuring Zones for ACLs](#), on page 251

[Transaction Security](#), on page 251

Assigning ACLs on DNS Caching Servers or Zones

You assign ACLs on the DNS Caching server or zone level. ACLs can include one or more of these elements:

- **IP address**—In dotted decimal notation; for example, 192.168.1.2.
- **Network address**—In dotted decimal and slash notation; for example, 192.168.0.0/24. In this example, only hosts on that network can update the DNS server.
- **Another ACL**—Must be predefined. You cannot delete an ACL that is embedded in another one until you remove the embedded relationship. You should not delete an ACL until all references to that ACL are deleted.
- **Transaction Signature (TSIG) key**—The value must be in the form **key value**, with the keyword **key** followed by the secret value. To accommodate space characters, the entire list must be enclosed in double quotes. For TSIG keys, see [Transaction Security](#), on page 251.

You assign each ACL a unique name. However, the following ACL names have special meanings and you cannot use them for regular ACL names:

- **any**—Anyone can perform a certain action
- **none**—No one can perform a certain action
- **localhost**—Any of the local host addresses can perform a certain action
- **localnets**—Any of the local networks can perform a certain action

Note the following:

- If an ACL is not configured, **any** is assumed.
- If an ACL is configured, at least one clause must allow traffic.
- The negation operator (!) disallows traffic for the object it precedes, but it does not intrinsically allow anything else unless you also explicitly specify it. For example, to disallow traffic for the IP address 192.168.50.0 only, use **!192.168.50.0, any**.

Local Advanced Web UI

From the **Design** menu, choose **ACLs** under the **Security** submenu to open the List/Add Access Control Lists page. Click the **Add ACLs** icon in the ACLs pane and enter an ACL name and match list and click Add ACL. Note that a **key value** pair should not be in quotes. At the regional level, you can additionally pull replica ACLs or push ACLs to local clusters.

CLI Commands

Use **acl name create match-list**, which takes a name and one or more ACL elements. The ACL list is comma-separated, with double quotes surrounding it if there is a space character. The CLI does not provide the pull/push function.

For example, the following commands create three ACLs. The first is a key with a value, the second is for a network, and the third points to the first ACL. Including an exclamation point (!) before a value negates that value, so that you can exclude it in a series of values:

```
nrcmd> acl sec-acl create "key h-a.h-b.example.com."  
  
nrcmd> acl dyn-update-acl create "!192.168.2.13,192.168.2.0/24"  
  
nrcmd> acl main-acl create sec-acl
```

Configuring Zones for ACLs

To configure ACLs for the DNS server or zones, set up a DNS update policy, then define this update policy for the zone (see [Configuring DNS Update Policies, on page 258](#)).

Transaction Security

Transaction Signature (TSIG) RRs enable the DNS server to authenticate each message that it receives, containing a TSIG. Communication between servers is not encrypted but it becomes authenticated, which allows validation of the authenticity of the data and the source of the packet.

When you configure the Cisco Prime Network Registrar DHCP server to use TSIG for DNS updates, the server appends a TSIG RR to the messages. Part of the TSIG record is a message authentication code.

When the DNS server receives a message, it looks for the TSIG record. If it finds one, it first verifies that the key name in it is one of the keys it recognizes. It then verifies that the time stamp in the update is reasonable (to help fight against traffic replay attacks). Finally, the server looks up the key shared secret that was sent in the packet and calculates its own authentication code. If the resulting calculated authentication code matches the one included in the packet, then the contents are considered to be authentic.

Related Topics

[Creating TSIG Keys, on page 251](#)

[Generating Keys, on page 252](#)

[Considerations for Managing Keys, on page 253](#)

[Adding Supporting TSIG Attributes, on page 253](#)

Creating TSIG Keys



Note If you want to enable key authentication for Address-to-User Lookup (ATUL) support, you must also define a key identifier (*id* attribute value). See [Setting DHCP Forwarding, on page 23](#).

Local Advanced Web UI

From the **Design** menu, choose **Keys** under the **Security** submenu to open the List/Add Encryption Keys page.

For a description of the Algorithm, Security Type, Time Skew, Key ID, and Secret values, see [Table 33: Options for the `cnr_keygen` Utility](#). See also [Considerations for Managing Keys, on page 253](#).

To edit a TSIG key, click its name on the List/Add Encryption Keys page to open the Edit Encryption Key page.

At the regional level, you can additionally pull replica keys, or push keys to local clusters.

CLI Commands

Use **key name create secret**. Provide a name for the key (in domain name format; for example, `hosta-hostb-example.com.`) and a minimum of the shared secret as a base-64 encoded string (see [Table 33: Options for the `cnr_keygen` Utility](#) for a description of the optional time skew attribute). An example in the CLI would be:

```
nrcmd> key hosta-hostb-example.com.create secret-string
```

Generating Keys

It is recommended that you use the Cisco Prime Network Registrar **cnr_keygen** utility to generate TSIG keys so that you add them or import them using **import keys**.

Execute the **cnr_keygen** key generator utility from a DOS prompt, or a Linux shell:

- On Windows, the utility is in the *install-path* \bin folder.
- On Linux, the utility is in the *install-path* /usrbin directory.

An example of its usage (on Linux) is:

```
> /opt/nwreg2/local/usrbin/cnr_keygen -n a.b.example.com. -a hmac-md5 -t TSIG -b 16
-s 300

key "a.b.example.com." {
algorithm hmac-md5;
secret "xGVCsFZ0/6e0N97HGF50eg==";
# cnr-time-skew 300;
# cnr-security-type TSIG;
};
```

The only required input is the key name. The options are described in the table below.

Table 33: Options for the `cnr_keygen` Utility

Option	Description
-a hmac-md5	Algorithm. Optional. Only hmac-md5 is currently supported.
-b secret-size	Byte size of the secret. Optional. The preset value is 16 bytes. The valid range is 1 through 64 bytes.
-s time-skew	Time skew for the key, in seconds. This is the maximum difference between the time stamp in packets signed with this key and the local system time. Optional. The preset value is 5 minutes. The range is one second through one hour.
-n name	Key name. Required. The maximum length is 255 bytes.
-t TSIG	Type of security used. Optional. Only TSIG is currently supported.
-h	Help. Optional. Displays the syntax and options of the utility.

Option	Description
-v	Version. Optional. Displays the version of the utility.

The resulting secret is base64-encoded as a random string.

You can also redirect the output to a file if you use the right-arrow (>) or double-right-arrow (>>) indicators at the end of the command line. The > writes or overwrites a given file, while the >> appends to an existing file. For example:

```
> /opt/nwreg2/local/usrbin/cnr_keygen -n example.com > keyfile.txt
> /opt/nwreg2/local/usrbin/cnr_keygen -n example.com >> addtokeyfile.txt
```

You can then import the key file into Cisco Prime Network Registrar using the CLI to generate the keys in the file. The key import can generate as many keys as it finds in the import file. The path to the file should be fully qualified. For example:

```
nrcmd> import keys keydir/keyfile.txt
```

Considerations for Managing Keys

If you generate your own keys, you must enter them as a base64-encoded string (See RFC 4648 for more information on base64 encoding). This means that the only characters allowed are those in the base64 alphabet and the equals sign (=) as pad character. Entering a nonbase64-encoded string results in an error message.

Here are some other suggestions:

- Do not add or modify keys using batch commands.
- Change shared secrets frequently; every two months is recommended. Note that Cisco Prime Network Registrar does not explicitly enforce this.
- The shared secret length should be at least as long as the keyed message digest (HMAC-MD5 is 16 bytes). Note that Cisco Prime Network Registrar does not explicitly enforce this and only checks that the shared secret is a valid base64-encoded string, but it is the policy recommended by RFC 2845.

Adding Supporting TSIG Attributes

To add TSIG support for a DNS update configuration (see [Creating DNS Update Configurations, on page 256](#)), set these attributes:

- *server-key*
- *backup-server-key*

To use GSS-TSIG security algorithm in TSIG, enable the below attribute:

- *use-gss-tsig*

GSS-TSIG

RFC 3645 proposed extending TSIG to allow the Generic Security Service (GSS) method of secure key exchange, eliminating the need for manually distributing keys to all GSS clients. It defines an algorithm to use with TSIG, which is based on the Generic Security Service Application Program Interface (GSS API), as specified in RFC2743.

GSS-TSIG provides the secure DDNS updates and secure Zone Transfers utilizing the Kerberos security mechanism.

Client and Server use GSS API calls to establish a limited lifetime security context for authentication, integrity and confidentiality. Establishing a security context involves the passing of opaque tokens between the client and server until the negotiation is complete. The TKEY resource record [RFC2930] is used as the vehicle to transfer tokens between client and server. Once the security context is established it is used to generate and verify signatures using GSS API calls. These signatures are exchanged by the Client and Server as a part of the TSIG records exchanged in DNS messages sent between the Client and Server, as described in [RFC2845].

Client and Server MUST be locally authenticated with Kerberos server before using this protocol. Generally the initial TGT(ticket to get ticket) ticket is available in cache through system logon or obtained using utility like kinit. DHCP/DNS Client will request Kerberos server for the service ticket using the principal name(DNS/<hostname>). Client provides the service ticket to prove authentication when interacting, securely, with DNS server. The service ticket will be encrypted by the Kerberos server using service key, which can be decrypted only by the application server using the same service key.

For more information, see the [Configuring GSS-TSIG, on page 279](#) for the configuration required on the DHCP Server and DNS Server.



Note By default, Cisco Prime Network Registrar will support HMAC-MD5 based secure TSIG updates. To enable the GSS based secure updates, user has to disable-all HMAC-MD5 configuration in the DNS server by selecting none option in tsig-processing attribute.

DHCP Server and Secondary DNS Server Configuration in Linux

Configure the KDC Server information in /etc/krb5.conf. Use kinit utility to get the initial ticket from KDC.

DHCP Server and Secondary DNS Server Configuration in Windows

The server machine should be under the AD domain and not workgroup.



Note To enable GSS-TSIG in DHCP server/Secondary DNS server, ensure that use-gss-tsig(Boolean) attribute is configured in DNS Update Config/Secondary Zone page respectively.

Troubleshooting DHCP Server and Secondary DNS Server Configuration

- Client-related errors that can occur while getting the initial credentials:
- CLOCK SKEW ERROR - Ensure the Kerberos client and server and synchronized in time if not synchronize with ntp.
- KDC not reachable - Ensure AD hostname is resolvable.
- kinit - Client not found in Kerberos database while getting initial credentials - Verify whether the user exists in AD.
- kinit - Cannot resolve servers for KDC in realm "DOMAIN.com" while getting the initial credentials - Verify whether the REALM exists in AD.

- `kinit` - Preauthentication failed while getting the initial credentials - Verify whether the password entered to get the ticket is same as the password associated to the user in AD.



Note Generally the initial TGT (ticket to get ticket) ticket is available in cache through system logon in Windows.

Creating GSS-TSIG Configuration

DNS/DHCP maintains non-persistent table for key management.



Note You have the option to change the default TKEY management values used by DHCP and DNS server. You must create a GSS- TSIG configuration and provide reference in the DHCP/DNS server page.

Local Basic or Advanced Web UI

From the **Design** menu, choose **GSS-TSIG** under the **Security** submenu to open the List/Add GSS-TSIG Configuration page.

GSS-TSIG attributes

- *tkey-max-exchanges* - Per recommendation from RFC 3645 to prevent endless looping, the DNS server shall impose a maximum number of TKEY exchanges (i.e. number TKEY queries received from a particular client) in the attempt to negotiate a particular key. This attribute shall specify this limit. A TKEY table record maintains the exchange-count. If exchange-count exceeds *tkey-max-exchanges* during key negotiation, the DNS server shall abort the key negotiation.
- *tkey-table-max-size* - This attribute bounds the size of the TKEY table.
- *tkey-table-purge-interval* - The time interval at which purging of expired keys from TKEY table should happen.
- *tkey-session-time* - Specifies the user configurable maximum lifetime of a key. Lifetime of a key is controlled by the Kerberos server expiry time obtained during the initial key negotiation and through this attribute. If set to 0, this attribute is disabled and the key lifetime is controlled only by the Kerberos given expiry time. When this attribute is configured with a value > 0, the minimum of Kerberos expiry time and this value is taken as the maximum lifetime of the key.

To edit a GSS-TSIG configuration, click its name on the List/Add GSS-TSIG Configuration page to open the Edit GSS-TSIG Configuration page.

At the regional level, you can additionally pull or push GSS-TSIG configuration to local clusters.

CLI Commands

Use `gss-tsig name create [attribute=value ...]`. Provide a name for the GSS-TSIG configuration object. For example:

```
nrcmd> gss-tsig gss create tkey-max-exchanges=6 tkey-table-max-size=500
tkey-table-purge-interval=90
```

Creating DNS Update Configurations

A DNS update configuration defines the DHCP server framework for DNS updates to a DNS server or HA DNS server pair. It determines if you want to generate forward or reverse zone DNS updates (or both). It optionally sets TSIG keys for the transaction, attributes to control the style of autogenerated hostnames, and the specific forward or reverse zone to be updated. You must specify a DNS update configuration for each unique server relationship.

For example, if all updates from the DHCP server are directed to a single DNS server, you can create a single DNS update configuration that is set on the server default policy. To assign each group of clients in a client-class to a corresponding forward zone, set the forward zone name for each in a more specific client-class policy.

Local Advanced and Regional Web UI

-
- Step 1** From the **Deploy** menu, choose **DNS Update Configs** under the **DNS Updates** submenu to open the List/Add DNS Update Configurations page.
- Step 2** Click the **Add DNS Update Configs** icon in the **DNS Update Configs** pane to open the **Add DnsUpdateConfig** dialog box.
- Step 3** Enter a name for the update configuration in the *Name* attribute field.
- Step 4** Click **Add DnsUpdateConfig** to add the DNS update configuration.
- Step 5** Select the name of update configuration to open the Edit DNS Update Configuration page.
- Step 6** Click the appropriate *dynamic-dns* setting under the Update Settings block:
- **update-none**—Do not update forward or reverse zones.
 - **update-all**—Update forward and reverse zones (the default value).
 - **update-fwd-only**—Update forward zones only.
 - **update-reverse-only**—Update reverse zones only.
- Step 7** Click the appropriate *dns-client-identity* setting under the Update Settings block:
- **txt**—The server uses TXT RR for DHCPv4 DNS updates and DHCID RR for DHCPv6 DNS updates.
 - **dheid**—The server uses DHCID RR for both DHCPv4 and DHCPv6 DNS updates.
 - **transition-to-dheid**—The server uses DHCID RR for new records in the DNS server and updates existing entries to use the DHCID RR when the next DNS update is done.
 - **regress-to-txt**—The server uses the TXT RR for new entries in the DNS server and upgrades existing entries to use the TXT RR when the next DNS update is done.
- Note** The *dns-client-identity* attribute is also available as part of the DHCP server-wide settings which will be taken into consideration if the attribute of the individual DNS update config was not configured.
- Step 8** Set the other attributes appropriately:
- If necessary, enable *synthesize-name* and set the *synthetic-name-stem* value.
- You can set the stem of the default hostname to use if clients do not supply hostnames, by using *synthetic-name-stem*. For DHCPv4, enable the *synthesize-name* attribute to trigger the DHCP server to synthesize unique names for clients based on the value of the *synthetic-name-stem*. The resulting name is the name stem appended with the

hyphenated IP address. For example, if you specify a *synthetic-name-stem* of **host** for address 192.168.50.1 in the example.com domain, and enable the *synthesize-name* attribute, the resulting hostname is host-192-168-50-1.example.com. The preset value for the synthetic name stem is **dhcp**

The *synthetic-name-stem* must:

- Be a relative name without a trailing dot.
- Include alphanumeric values and hyphens (–) only. Space characters and underscores become hyphens and other characters are removed.
- Include no leading or trailing hyphen characters.
- Have DNS hostnames of no more than 63 characters per label and 255 characters in their entirety. The algorithm uses the configured forward zone name to determine the number of available characters for the hostname, and truncates the end of the last label if necessary.

For DHCPv6, see [Generating Synthetic Names in DHCPv4 and DHCPv6, on page 248](#).

- Set *forward-zone-name* to the forward zone, if updating forward zones. Note that the policy *forward-zone-name* takes precedence over the one set in the DNS update configuration.

For DHCPv6, the server ignores the client and client-class policies when searching for a *forward-zone-name* value in the policy hierarchy. The search for a forward zone name begins with the prefix embedded policy.

- For DHCPv4, set *reverse-zone-name* to the reverse (in.addr.arpa) zone to be updated with PTR and TXT records. If unset and the DHCP server *synthesize-reverse-zone* attribute is enabled, the server synthesizes a reverse zone name based on the address of each lease, scope subnet number, and DNS update configuration (or scope) *dns-host-bytes* attribute value.

The *dns-host-bytes* value controls the split between the host and zone parts of the reverse zone name. The value sets the number of bytes from the lease IP address to use for the hostname; the remaining bytes are used for the in.addr.arpa zone name. A value of 1 means use just one byte for the host part of the domain and the other three from the domain name (reversed). A value of 4 means use all four bytes for the host part of the address, thus using just the in.addr.arpa part of the domain. If unset, the server synthesizes an appropriate value based on the scope subnet size, or if the *reverse-zone-name* is defined, calculates the host bytes from this name.

The *one-a-rr-per-dns-name* controls the DHCPv4 DNS updates to allow either one or multiple A RRs per name. The CNR versions prior to 8.2, the server supported only one A per name (name to address mapping entry) since the server used the mac-address based identifier. The introduction of the DUID support and DHCPID RR in CPNR 8.2, multi-connection clients will have multiple A RRs.

For DHCPv6, see [Determining Reverse Zones for DNS Updates, on page 248](#).

- Set *server-addr/server-ipv6addr* to the IPv4/IPv6 address of the primary DNS server for the forward zone (or reverse zone if updating reverse zones only).

Set *server-key* and *backup-server-key* if you are using a TSIG key to process all DNS updates (see [Transaction Security, on page 251](#)).

Set *use-gss-tsig* to true, if you are using the Generic Security Service (GSS) method of the secure key exchange (see [Configuring GSS-TSIG](#)

- Set *backup-server-addr/backup-server-ipv6addr* to the IPv4/IPv6 address of the backup DNS server, if HA DNS is configured.
- If necessary, enable or disable *update-dns-first* (preset value disabled) or *update-dns-for-bootp* (preset value enabled). The *update-dns-first* setting controls whether DHCP updates DNS before granting a lease. Enabling this attribute is not recommended.

- Step 9** At the regional level, you can also push update configurations to the local clusters, or pull them from the replica database on the List/Add DNS Update Configurations page.
- Step 10** Click **Save**.
- Step 11** To specify this DNS update configuration on a policy, see [Creating and Applying DHCP Policies, on page 166](#).
-

CLI Commands

Use `dhcp-dns-update name create [attribute=value ...]`. For example:

```
dhcp-dns-update example-update-config create
```

Set the `dynamic-dns` attribute to its appropriate value (update-none, update-all, update-fwd-only, or update-reverse-only). For example:

```
nrcmd> dhcp-dns-update example-update-config set dynamic-dns=update-all
```

Related Topics

[DNS Update Process, on page 245](#)

[Special DNS Update Considerations, on page 246](#)

[DNS Updates for DHCPv6, on page 246](#)

Configuring DNS Update Policies

DNS update policies provide a mechanism for managing update authorization at the RR level. Using update policies, you can grant or deny DNS updates based on rules that are based on ACLs as well as RR names and types. ACLs are described in [Assigning ACLs on DNS Caching Servers or Zones , on page 250](#).

Related Topics

[Compatibility with Cisco Network Registrar Releases, on page 258](#)

[Creating and Editing Update Policies, on page 259](#)

[Defining and Applying Rules for Update Policies, on page 259](#)

Compatibility with Cisco Network Registrar Releases

Cisco Prime Network Registrar releases used static RRs that administrators entered, but that DNS updates could not modify. This distinction between static and dynamic RRs no longer exists. RRs can now be marked as protected or unprotected (see the *"Protecting Resource Record Sets"* section in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide*). Administrators creating or modifying RRs can now specify whether RRs should be protected. A DNS update cannot modify a protected RR set, even if an RR of the given type does not yet exist in the set.



Note Previous releases allowed DNS updates only to A, TXT, PTR, CNAME and SRV records. This was changed to allow updates to all but SOA and NS records in unprotected name sets. To remain compatible with a previous release, use an update policy to limit RR updates.

Creating and Editing Update Policies

Creating an update policy initially involves creating a name for it.

Local Advanced Web UI

-
- Step 1** From the **Design** menu, choose **Update Policies** under the **Security** submenu to open the List/Add DNS Update Policies page. The option is available if the server is configured with authoritative service.
 - Step 2** Click the **Add Update Policies** icon in the Update Policies pane to open the **Add DNS Update Policy** dialog box.
 - Step 3** Enter a name for the update policy.
 - Step 4** Click **Add DNS Update Policy**.
 - Step 5** Proceed to [Defining and Applying Rules for Update Policies, on page 259](#).
-

CLI Commands

Use `update-policy name create`. For example:

```
nrcmd> update-policy policy1 create
```

Defining and Applying Rules for Update Policies

DNS update policies are effective only if you define rules for each that grant or deny updates for certain RRs based on an ACL. If no rule is satisfied, the default (last implicit) rule is to deny all updates ("**deny any wildcard * ***").

Related Topics

[Defining Rules for Named Update Policies, on page 259](#)

[Applying Update Policies to Zones, on page 262](#)

Defining Rules for Named Update Policies

Defining rules for named update policies involves a series of Grant and Deny statements.

Local Advanced Web UI

-
- Step 1** Create an update policy, as described in [Creating and Editing Update Policies, on page 259](#), or edit it.
 - Step 2** On the List/Add DNS Update Policies or Edit DNS Update Policy page:

- a) Enter an optional value in the Index field.
- b) Click Grant to grant the rule, or Deny to deny the rule.
- c) Enter an access control list in the ACL List field.
- d) Choose a keyword from the Keyword drop-down list.
- e) Enter a value based on the keyword in the Value field. This can be a RR or subdomain name, or, if the **wildcard** keyword is used, it can contain wildcards (see the table below).

As networks make the transition from the IPv4 to IPv6 addressing, a lot of network devices will use both IPv4 and IPv6 addresses. These devices may be using multiple interfaces on the same host, using different networks, or using different DHCP versions. These devices need to be identified consistently with respect to DHCP server and accordingly the DHCP server will update the DNS server.

In Cisco Prime Network Registrar 8.1 or earlier, DHCPv4 uses TXT RRs and DHCPv6 uses DHCID RRs to make the DNS updates. To avoid conflicts in the client-requested names the dual-stack clients cannot use a single forward FQDN. These conflicts are primarily applied to the client-requested names and not to the generated names, which are generally unique. To avoid these conflicts, different zones were used for the DHCPv4 and DHCPv6 names.

In Cisco Prime Network Registrar 8.2 and later, DHCPv4 uses TXT RR or DHCID RR and DHCPv6 uses DHCID RR for DNS updates. The default value of DHCP server-wide settings attribute `dns-client-identity` is `txt` and the attribute is not configured for individual DNS update config objects. You can configure the DNS updates in one of the following ways:

- **TXT RR for DHCPv4 and DHCID for DHCPv6**—To enable this configuration set the `dns-client-identity` to `txt`. The server will use the TXT RR in DHCPv4 DNS updates and DHCID RR for DHCPv6 DNS updates. This setting is used for backwards compatibility as Cisco Prime Network Registrar 8.1 or earlier only support using TXT RRs for DHCPv4. This setting must be used if Cisco Prime Network Registrar 8.1 or earlier clusters are involved in doing DNS updates to the zone.
- **DHCID RR for both DHCPv4 and DHCPv6**—To enable this configuration set the `dns-client-identity` to `dhcid`. The server will use the DHCID RR for both DHCPv4 and DHCPv6 DNS updates. This setting should be used to support dual stack clients and can only be used if all DHCP servers doing DNS updates to the zones for this configuration support and are configured to use the DHCID RR.
- **Transition to DHCID RR**—To enable this configuration set the `dns-client-identity` to `transition-to-dhcid`. Set the `force-dns-update` attribute to `true`. Reload the server. For the zones that need to be upgraded, set the `dns-client-identity` attribute to `dhcid` and restore the `force-dns-update` attribute to its earlier value, after the longest lease time configured in the server.

Note You must set the `transition-to-dhcid` attribute until all the DHCPv4 resource records are updated to DHCID RR. For more information, see [Transitioning to DHCID RR for DHCPv4, on page 266](#).

- **Regress to TXT RR**—To enable this configuration set the `dns-client-identity` to `regress-to-txt`. Set the `force-dns-update` attribute to `true`. Reload the server. For the zones that need to be upgraded, set the `dns-client-identity` attribute to `txt` and restore the `force-dns-update` attribute to its earlier value, after the longest lease time configured in the server.

Note The CCM server will not allow the failover synchronization if one of the failover servers runs on Cisco Prime Network Registrar 8.1 or earlier and the `dns-client-identity` attribute, in the DHCP server or a DNS Update Configuration, is set to anything other than `txt`.

Table 34: Wildcard Values for Update Policy Rules

Wildcard	Description
*	Matches zero or more characters. For example, the pattern example* matches all strings starting with <i>example</i> , including example- .

Wildcard	Description
?	Matches a single character only. For example, the pattern example?.com matches example1.com and example2.com , but not example.com .
/[/]	Matches any characters in the (escaped) brackets; for example, /[abc/] . Each square bracket must be escaped using a slash (/). The characters can also be in a range; for example, /[0-9/] and /[a-z/] . If a pattern should include a hyphen, make the hyphen the first character; for example, example/[a-z/] .

- f) Enter one or more RR types, separated by commas, in the RR Types field, or use * for “all RRs.” You can use negated values, which are values prefixed by an exclamation point; for example, **!PTR**.
- g) Click **Save**.

- Step 3** At the regional level, you can also push update policies to the local clusters, or pull them from the replica database on the List/Add DNS Update Policies page.
- Step 4** To edit an update policy, click the name of the update policy on the List/Add DNS Update Policies page to open the Edit DNS Update Policy page, make changes to the fields, then click **Save**.

CLI Commands

Create or edit an update policy (see [Creating and Editing Update Policies, on page 259](#), then use **update-policy name rules add rule**, with *rule* being the rule. (See the table above for the rule wildcard values.) For example:

```
nrcmd> update-policy policy1 rules add "grant 192.168.50.101 name host1 A,TXT" 0
```

The rule is enclosed in quotes. To parse the rule syntax for the example:

- **grant**—Action that the server should take, either **grant** or **deny**.
- **192.168.50.101**—The ACL, in this case an IP address. The ACL can be one of the following:
 - Name—ACL created by name, as described in [Assigning ACLs on DNS Caching Servers or Zones, on page 250](#).
 - IP address, as in the example.
 - Network address, including mask; for example, **192.168.50.0/24**.
 - TSIG key—Transaction signature key, in the form **key=key**, (as described in [Transaction Security, on page 251](#)).
 - One of the reserved words:
 - any**—Any ACL
 - none**—No ACL
 - localhost**—Any local host addresses
 - localnets**—Any local network address

You can negate the ACL value by preceding it with an exclamation point (!).

- **name**—Keyword, or type of check to perform on the RR, which can be one of the following:
 - **name**—Name of the RR, requiring a name value.

- **subdomain**—Name of the RR or the subdomain with any of its RRs, requiring a name or subdomain value.
- **wildcard**—Name of the RR, using a wildcard value (see the table above).
- **host1**—Value based on the keyword, in this case the RR named host1. This can also be a subdomain name or, if the **wildcard** keyword is used, can contain wildcards (see the table above).
- **A,TXT**—RR types, each separated by a comma. This can be a list of any of the RR types described in the "Resource Records" section in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide*. You can negate each record type value by preceding it with an exclamation point (!).
- Note that if this or any assigned rule is not satisfied, the default is to deny all RR updates.

Tacked onto the end of the rule, outside the quotes, is an index number, in the example, **0**. The index numbers start at 0. If there are multiple rules for an update policy, the index serves to add the rule in a specific order, such that lower numbered indexes have priority in the list. If a rule does not include an index, it is placed at the end of the list. Thus, a rule always has an index, whether or not it is explicitly defined. You also specify the index number in case you need to remove the rule.

To replace a rule, use **update-policy name delete**, then recreate the update policy. To edit a rule, use **update-policy name rules remove index**, where *index* is the explicitly defined or system-defined index number (remembering that the index numbering starts at 0), then recreate the rule. To remove the second rule in the previous example, enter:

```
nrcmd> update-policy policy1 rules remove 1
```

Applying Update Policies to Zones

After creating an update policy, you can apply it to a zone (forward and reverse) or zone template if you have configured the DNS server with authoritative services.

Local Advanced Web UI

-
- Step 1** From the **Design** menu, choose **Forward Zones** under the **Auth DNS** submenu to open the List/Add Forward Zones page.
- Step 2** Click the name of the of zone to open the Edit Zone page.
- Tip** You can also perform this function for zone templates on the Edit Zone Template page, and primary reverse zones on the Edit Primary Reverse Zone page (see the "Managing Zones" chapter in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide*).
- Step 3** Enter the name or (comma-separated) names of one or more of the existing named update policies in the *update-policy-list* attribute field.
- Note** The server processes the update-acl before it processes the *update-policy-list*.
- Step 4** Click **Save**.
-

CLI Commands

Use **zone name set update-policy-list**, equating the *update-policy-list* attribute with a quoted list of comma-separated update policies, as defined in [Creating and Editing Update Policies, on page 259](#). For example:

```
nrcmd> zone example.com set update-policy-list="policy1,policy2"
```

Creating DNS Update Maps

A DNS update map facilitates configuring DNS updates so that the update properties are synchronized between HA DNS server pairs or DHCP failover server pairs, based on an update configuration, so as to reduce redundant data entry. The update map applies to all the primary zones that the DNS pairs service, or all the scopes that the DHCP pairs service. You must specify a policy for the update map. To use this function, you must be an administrator assigned the server-management subrole of the dns-management or central-dns-management role, and the dhcp-management role (for update configurations).

Local and Regional Web UI

-
- Step 1** From the **Deploy** menu, choose **Update Maps** under the **DNS Updates** submenu to open the List/Add DNS Update Maps page. The option is available if the server is configured with authoritative service.
- Step 2** Click the **Add DNS Update Map** icon in the Update Maps pane to open the Add DNS Update Map dialog box.
- Step 3** Enter a name for the update map in the Name field.
- Step 4** Choose the The DNS server or HA pair associated with this configuration.
- Step 5** Choose the DHCP server or DHCP failover pair associated with this configuration.
- Step 6** Enter the DNS update configuration from the previous section in the *dns-config* field.
- Step 7** Set the kind of policy selection you want for the *dhcp-policy-selector* attribute. The choices are:
- **use-named-policy**—Use the named policy set for the *dhcp-named-policy* attribute (the preset value).
 - **use-client-class-embedded-policy**—Use the embedded policy from the client-class set for the *dhcp-client-class* attribute.
 - **use-scope-embedded-policy**—Use the embedded policy from the scope.
- Step 8** If using update ACLs (see [Configuring Access Control Lists and Transaction Security, on page 249](#)) or DNS update policies (see [Configuring DNS Update Policies, on page 258](#)), set either the *dns-update-acl* or *dns-update-policy-list* attribute. Either value can be one or more addresses separated by commas. The *dns-update-acl* takes precedence over the *dns-update-policy-list*.
- If you omit both values, a simple update ACL is constructed whereby only the specified DHCP servers or failover pair can perform updates, along with any *server-key* value set in the update configuration specified for the *dns-config* attribute.
- Step 9** Click **Add DNS Update Map**.
- Step 10** At the regional level, you can also push update maps to the local clusters, or pull them from the replica database on the List/Add DNS Update Maps page.
-

CLI Commands

Specify the name, cluster of the DHCP and DNS servers (or DHCP failover or HA DNS server pair), and the DNS update configuration when you create the update map, using **dns-update-map name create dhcp-cluster dns-cluster dns-config**. For example:

```
nrcmd> dns-update-map example-update-map create Example-cluster Boston-cluster
example-update-config
```

Set the *dhcp-policy-selector* attribute value to *use-named-policy*, *use-client-class-embedded-policy*, or *use-scope-embedded-policy*. If using the *use-named-policy* value, also set the *dhcp-named-policy* attribute value. For example:

```
nrcmd> dns-update-map example-update-map set dhcp-policy-selector=use-named-policy
```

```
nrcmd> dns-update-map example-update-map set dhcp-named-policy=example-policy
```

Confirming Dynamic Records

The Cisco Prime Network Registrar DHCP server stores all pending DNS update data on disk. If the DHCP server cannot communicate with a DNS server, it periodically tests for re-established communication and submits all pending updates. This test typically occurs every 40 seconds.

Local Advanced Web UI

From the **Design** menu, choose **Forward Zones** under the **Auth DNS** submenu. Click the **Resource Records** tab to open the Edit Zone page.

CLI Commands

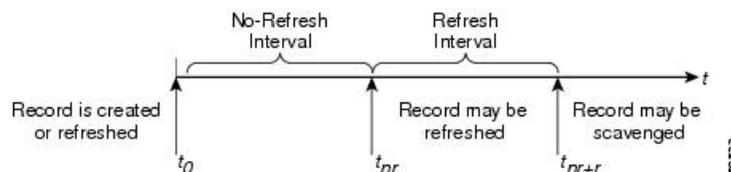
Use `zone name listRR dns`.

Scavenging Dynamic Records

Microsoft Windows DNS clients that get DHCP leases can update (refresh) their Address (A) records directly with the DNS server. Because many of these clients are mobile laptops that are not permanently connected, some A records may become obsolete over time. The Windows DNS server scavenges and purges these primary zone records periodically. Cisco Prime Network Registrar provides a similar feature that you can use to periodically purge stale records.

Scavenging is normally disabled by default, but you should enable it for zones that exclusively contain Windows clients. Zones are configured with *no-refresh* and *refresh* intervals. A record expires once it ages past its initial creation date plus these two intervals. The image below shows the intervals in the scavenging time line.

Figure 13: Address Record Scavenging Time Line Intervals



The Cisco Prime Network Registrar process is:

1. When the client updates the DNS server with a new A record, this record gets a timestamp, or if the client refreshes its A record, this may update the timestamp (“Record is created or refreshed”).
2. During a no-refresh interval (a default value of seven days), if the client keeps sending the same record without an address change, this does not update the record timestamp.
3. Once the record ages past the no-refresh interval, it enters the refresh interval (also a default value of seven days), during which time DNS updates refresh the timestamp and put the record back into the no-refresh interval.
4. A record that ages past the refresh interval is available for scavenging when it reaches the scavenge interval.



Note Only unprotected RRs are scavenged. To keep RRs from being scavenged, set them to protected. However, top-of-zone (@) RRs, even if unprotected, are not scavenged.

The following DNS server attributes affect scavenging:

- *scvg-interval*—Period during which the DNS server checks for stale records in a zone. The value can range from one hour to 365 days. You can also set this for the server (the default value is one week), although the zone setting overrides it.
- *scvg-no-refresh-interval*—Interval during which actions, such as dynamic or prerequisite-only DNS updates, do not update the record timestamp. The value can range from one hour to 365 days. The zone setting overrides the server setting (the default value is one week).
- *scvg-refresh-interval*—Interval during which DNS updates increment the record timestamp. After both the no-refresh and refresh intervals expire, the record is a candidate for scavenging. The value can range from one hour to 365 days. The zone setting overrides the server setting (the default value is one week).
- *scvg-ignore-restart-interval*—Ensures that the server does not reset the scavenging time with every server restart. Within this interval, Cisco Prime Network Registrar ignores the duration between a server down instance and a restart, which is usually fairly short.

The value can range from two hours to one day. With any value longer than that set, Cisco Prime Network Registrar recalculates the scavenging period to allow for record updates that cannot take place while the server is stopped. The zone setting overrides the server setting (the default value is 2 hours).

Enable scavenging only for zones where a Cisco Prime Network Registrar DNS server receives updates exclusively from Windows clients (or those known to do automatic periodic DNS updates). Set the attributes listed above. The Cisco Prime Network Registrar scavenging manager starts at server startup. It reports records purged through scavenging to the changeset database. Cisco Prime Network Registrar also notifies secondary zones by way of zone transfers of any records scavenged from the primary zone. In cases where you create a zone that has scavenging disabled (the records do not have a timestamp) and then subsequently enable it, Cisco Prime Network Registrar uses a proxy timestamp as a default timestamp for each record.

You can monitor scavenging activity using one or more of the log settings `scavenge`, `scavenge-details`, `ddns-refreshes`, and `ddns-refreshes-details`.

Local Advanced Web UI

On the Manage DNS Server page, click the **Commands** button to open the DNS Commands dialog box. Click the **Run** icon next to Scavenge all zones.

To scavenge a particular forward or reverse zone only, go to the Zone Commands for Zone page, which is available by clicking the **Commands** button on the List/Add Forward Zones page or List/Add Reverse Zones

page. Click the **Run** icon next to Scavenge zone on the Zone Commands for Zone page. To find out the next time scavenging is scheduled for the zone, click the **Run** icon next to Get scavenging start time.

CLI Commands

Use **dns scaveng** for all zones that have scavenging enabled. Use the **getScavengingStartTime** action on a zone to find out the next time scavenging is scheduled to start.

Transitioning to DHCID RR for DHCPv4

As networks make the transition from the IPv4 to IPv6 addressing, a lot of network devices will use both IPv4 and IPv6 addresses. These devices may be using multiple interfaces on the same host, using different networks, or using different DHCP versions. These devices need to be identified consistently with respect to DHCP server and accordingly the DHCP server will update the DNS server.

In Cisco Prime Network Registrar 8.1 or earlier, DHCPv4 uses TXT RRs and DHCPv6 uses DHCID RRs to make the DNS updates. To avoid conflicts in the client-requested names the dual-stack clients cannot use a single forward FQDN. These conflicts are primarily applied to the client-requested names and not to the generated names, which are generally unique. To avoid these conflicts, different zones were used for the DHCPv4 and DHCPv6 names.

In Cisco Prime Network Registrar 8.2 and later, DHCPv4 uses TXT RR or DHCID RR and DHCPv6 uses DHCID RR for DNS updates. The default value of DHCP server-wide settings attribute `dns-client-identity` is `txt` and the attribute is not configured for individual DNS update config objects. You can configure the DNS updates in one of the following ways:

- **TXT RR for DHCPv4 and DHCID for DHCPv6**—To enable this configuration set the `dns-client-identity` to `txt`. The server will use the TXT RR in DHCPv4 DNS updates and DHCID RR for DHCPv6 DNS updates. This setting is used for backwards compatibility as Cisco Prime Network Registrar 8.1 or earlier only support using TXT RRs for DHCPv4. This setting must be used if Cisco Prime Network Registrar 8.1 or earlier clusters are involved in doing DNS updates to the zone.
- **DHCID RR for both DHCPv4 and DHCPv6**—To enable this configuration set the `dns-client-identity` to `dheid`. The server will use the DHCID RR for both DHCPv4 and DHCPv6 DNS updates. This setting should be used to support dual stack clients and can only be used if all DHCP servers doing DNS updates to the zones for this configuration support and are configured to use the DHCID RR.
- **Transition to DHCID RR**—To enable this configuration set the `dns-client-identity` to `transition-to-dheid`. Set the `force-dns-update` attribute to `true`. Reload the server. For the zones that need to be upgraded, set the `dns-client-identity` attribute to `dheid` and restore the `force-dns-update` attribute to its earlier value, after the longest lease time configured in the server.



Note You must set the `transition-to-dheid` attribute until all the DHCPv4 resource records are updated to DHCID RR. For more information, see [Transitioning to DHCID RR for DHCPv4, on page 266](#).

- **Regress to TXT RR**—To enable this configuration set the `dns-client-identity` to `regress-to-txt`. Set the `force-dns-update` attribute to `true`. Reload the server. For the zones that need to be upgraded, set the `dns-client-identity` attribute to `txt` and restore the `force-dns-update` attribute to its earlier value, after the longest lease time configured in the server.



Note The CCM server will not allow the failover synchronization if one of the failover servers runs on Cisco Prime Network Registrar 8.1 or earlier and the `dns-client-identity` attribute, in the DHCP server or a DNS Update Configuration, is set to anything other than `txt`.

Local Advanced and Regional Web UI

-
- Step 1** From the **Deploy** menu, choose **DNS Update Configs** under the **DNS Updates** submenu to open the List/Add DNS Update Configurations page.
- Step 2** Select the name of the update configuration to open the **Edit DNS Update Configuration** page.
- Step 3** In the DNS update settings, set *transition-to-dhcid as dns-client-identity* in DNS update settings.
- Step 4** Optionally set *force-dns-update* to true. Using this setting will expedite the process of transitioning from TXT RR to DHCID RR.
- Step 5** Set scavenging settings attributes in forward or reverse zones to the following values:
- Set *scvg-enabled* to true.
- Step 6** Set scavenging settings attributes in DNS server to the following values:
- Set *scvg-interval* to longest lease time.
 - Set *scvg-refresh-interval* to longest lease time.
 - Set *scvg-no-refresh-interval* to 0.
- Step 7** Verify that all TXT RRs are converted to DHCID RRs in the RRs for the zones. You must set the *transition-to-dhcid* attribute until all the DHCPv4 resource records are updated to DHCID RR. If some TXT RRs entries do not transition to DHCID RR, you may need to remove these DNS entries manually by using the Cisco Prime Network Registrar single-record dynamic RR removal feature.
- Step 8** Click **Save**.
-

Configuring DNS Update for Windows Clients

The Windows operating system rely heavily on DNS and, to a lesser extent, DHCP. This reliance requires careful preparation on the part of network administrators prior to wide-scale Windows deployments. Windows clients can add entries for themselves into DNS by directly updating forward zones with their address (A) record. They cannot update reverse zones with their pointer (PTR) records.

Related Topics

[Client DNS Updates, on page 268](#)

[Dual Zone Updates for Windows Clients, on page 270](#)

[DNS Update Settings in Windows Clients, on page 270](#)

[Windows Client Settings in DHCP Servers, on page 270](#)

[SRV Records and DNS Updates, on page 271](#)

[Issues Related to Windows Environments, on page 273](#)

[Frequently Asked Questions About Windows Integration, on page 276](#)

Client DNS Updates

It is not recommended that clients be allowed to update DNS directly.

For a Windows client to send address record updates to the DNS server, two conditions must apply:

- The Windows client must have the **Register this connection's addresses in DNS** box checked on the **DNS** tab of its TCP/IP control panel settings.
- The DHCP policy must enable direct updating (Cisco Prime Network Registrar policies do so by default).

The Windows client notifies the DHCP server of its intention to update the A record to the DNS server by sending the *client-fqdn* DHCP option (81) in a DHCPREQUEST packet. By indicating the fully qualified domain name (FQDN), the option states unambiguously the client location in the domain namespace. Along with the FQDN itself, the client or server can send one of these possible flags in the *client-fqdn* option:

- **0**—Client should register its A record directly with the DNS server, and the DHCP server registers the PTR record (done through the policy *allow-client-a-record-update* attribute being enabled).
- **1**—Client wants the DHCP server to register its A and PTR records with the DNS server.
- **3**—DHCP server registers the A and PTR records with the DNS server regardless of the client request (done through the policy *allow-client-a-record-update* attribute being disabled, which is the default value). Only the DHCP server can set this flag.

The DHCP server returns its own *client-fqdn* response to the client in a DHCPACK based on whether DNS update is enabled. However, if the 0 flag is set (the *allow-client-a-record-update* attribute is enabled for the policy), enabling or disabling DNS update is irrelevant, because the client can still send its updates to DNS servers. See the table below for the actions taken based on how various properties are set.

Table 35: Windows Client DNS Update Options

DHCP Client Action	DNS Update	DHCP Server Action
Checks Register this connection's addresses in DNS and sends <i>client-fqdn</i> ; DHCP server enables <i>allow-client-a-record-update</i>	Enabled or disabled	Responds with <i>client-fqdn</i> that it allows the client to update its A records (sets flag 0), but the DHCP server still updates the PTR records.
Checks Register... and sends <i>client-fqdn</i> ; DHCP disables <i>allow-client-a-record-update</i>	Enabled	Responds with <i>client-fqdn</i> that it does not allow the client to update the DNS server directly (sets flag 3), and updates the A and PTR records.
	Disabled	Does not respond with <i>client-fqdn</i> and does not update the DNS server.
Unchecks Register... and sends <i>client-fqdn</i>	Enabled	Responds with <i>client-fqdn</i> that it is updating the A and PTR records.
	Disabled	Does not respond with <i>client-fqdn</i> and does not update the DNS server.
Does not send <i>client-fqdn</i>	Enabled	Does not respond with <i>client-fqdn</i> , but updates the A and PTR records.

DHCP Client Action	DNS Update	DHCP Server Action
	Disabled	Does not respond with <i>client-fqdn</i> and does not update the DNS server.

A Windows DHCP server can set the *client-fqdn* option to ignore the client request. To enable this behavior in Cisco Prime Network Registrar, create a policy for Windows clients and disable the *allow-client-a-record-update* attribute for this policy.

The following attributes are enabled by default in Cisco Prime Network Registrar:

- **Server use-client-fqdn**—The server uses the *client-fqdn* value on incoming packets and does not examine the *host-name*. The DHCP server ignores all characters after the first dot in the domain name value, because it determines the domain from the defined scope for that client. Disable *use-client-fqdn* only if you do not want the server to determine hostnames from *client-fqdn*, possibly because the client is sending unexpected characters.
- **Server use-client-fqdn-first**—The server examines *client-fqdn* on incoming packets from the client before examining the *host-name* option (12). If *client-fqdn* contains a hostname, the server uses it. If the server does not find the option, it uses the *host-name* value. If *use-client-fqdn-first* is disabled, the server prefers the *host-name* value over *client-fqdn*.
- **Server use-client-fqdn-if-asked**—The server returns the *client-fqdn* value in the outgoing packets if the client requests it. For example, the client might want to know the status of DNS activity, and hence request that the DHCP server should present the *client-fqdn* value.
- **Policy allow-client-a-record-update**—The client can update its A record directly with the DNS server, as long as the client sets the *client-fqdn* flag to 0 (requesting direct updating). Otherwise, the server updates the A record based on other configuration properties.

The hostnames returned to client requests vary depending on these settings (see the table below).

Table 36: Hostnames Returned Based on Client Request Parameters

Client Request	With Server/Policy Settings	Resulting Hostname
Includes <i>host-name</i> (option 12)	<i>use-host-name</i> =true <i>use-client-fqdn</i> =false (or <i>use-client-fqdn-first</i> =false) <i>trim-host-name</i> =true	<i>host-name</i> trimmed at first dot. Example: <i>host-name</i> host1.bob is returned host1.
	(same except:) <i>trim-host-name</i> =false	<i>host-name</i> . Example: <i>host-name</i> host1.bob is returned host1.bob.
Includes <i>client-fqdn</i> (option 81)	<i>use-client-fqdn</i> =true <i>use-host-name</i> =false (or <i>use-client-fqdn-first</i> =true)	<i>client-fqdn</i> trimmed at first dot. Example: <i>client-fqdn</i> host1.bob is returned host1.
Omits <i>host-name</i> (option 12) and <i>client-fqdn</i> (option 81)	Or: <i>use-host-name</i> =false <i>use-client-fqdn</i> =false	Set by client/policy hierarchy.
	(same as the previous except:) <i>hostname</i> is undefined in the client/policy hierarchy, with <i>synthesize-name</i> =true	Synthesized following the synthesizing rule, which is to append the hyphenated IP address of the host after the specified <i>synthetic-name-stem</i> .
	(same as the previous except:) <i>synthesize-name</i> =false	Undefined.

Dual Zone Updates for Windows Clients

Windows DHCP clients might be part of a DHCP deployment where they have A records in two DNS zones. In this case, the DHCP server returns the *client-fqdn* so that the client can request a dual zone update. To enable a dual zone update, enable the policy attribute *allow-dual-zone-dns-update*.

The DHCP client sends the 0 flag in *client-fqdn* and the DHCP server returns the 0 flag so that the client can update the DNS server with the A record in its main zone. However, the DHCP server also directly sends an A record update based on the client secondary zone in the behalf of the client. If both *allow-client-a-record-update* and the *allow-dual-zone-dns-update* are enabled, allowing the dual zone update takes precedence so that the server can update the secondary zone A record.

DNS Update Settings in Windows Clients

The Windows client can set advanced properties to enable sending the *client-fqdn* option.

-
- Step 1** On the Windows client, go to the Control Panel and open the TCP/IP Settings dialog box.
 - Step 2** Click the **Advanced** tab.
 - Step 3** Click the **DNS** tab.
 - Step 4** To have the client send the *client-fqdn* option in its request, leave the **Register this connection's addresses in DNS** box checked. This indicates that the client wants to do the A record update.
-

Windows Client Settings in DHCP Servers

You can apply a relevant policy to a scope that includes the Windows clients, and enable DNS updates for the scope.

-
- Step 1** Create a policy for the scope that includes the Windows clients. For example:
 - a) Create a policywin2k. You have to specify the forward or reverse zone name, main and backup server IP addresses when you create a policy.
 - b) Create a win2k scope with the subnet 192.168.1.0/24 and policywin2k as the policy. Add an address range of 192.168.1.10 through 192.168.1.100.
 - Step 2** Set the zone name, server address (for A records), reverse zone name, and reverse server address (for PTR records), as described in [Creating DNS Update Configurations, on page 256](#).
 - Step 3** If you want the client to update its A records at the DNS server, enable the policy attribute *allow-client-a-record-update* (this is the preset value). There are a few caveats to this:
 - If *allow-client-a-record-update* is enabled and the client sends the *client-fqdn* with the update bit enabled, the *host-name* and *client-fqdn* returned to the client match the client *client-fqdn*. (However, if the *override-client-fqdn* is also enabled on the server, the hostname and FQDN returned to the client are generated by the configured hostname and policy domain name.)
 - If, instead, the client does not send the *client-fqdn* with the update bit enabled, the server does the A record update, and the *host-name* and *client-fqdn* (if requested) returned to the client match the name used for the DNS update.

- If *allow-client-a-record-update* is disabled, the server does the A record updates, and the *host-name* and *client-fqdn* (with the update bit disabled) values returned to the client match the name used for the DNS update.
- If *allow-dual-zone-dns-update* is enabled, the DHCP server always does the A record updates. (See [Dual Zone Updates for Windows Clients, on page 270.](#))
- If *use-dns-update-prereqs* is enabled (the preset value) for the DHCP server or DNS update configuration and *update-dns-first* is disabled (the preset value) for the update configuration, the hostname and *client-fqdn* returned to the client are not guaranteed to match the DNS update, because of delayed name disambiguation. However, the lease data will be updated with the new names.

According to RFC 2136, update prerequisites determine the action the primary master DNS server takes based on whether an RR set or name record should or should not exist. Disable *use-dns-update-prereqs* only under rare circumstances.

Step 4 Reload the DHCP server.

SRV Records and DNS Updates

Windows relies heavily on the DNS protocol for advertising services to the network. The table below describes how Windows handles service location (SRV) DNS RRs and DNS updates.

You can configure the Cisco Prime Network Registrar DNS server so that Windows domain controllers can dynamically register their services in DNS and, thereby, advertise themselves to the network. Because this process occurs through RFC-compliant DNS updates, you do not need to do anything out of the ordinary in Cisco Prime Network Registrar.

Table 37: Windows SRV Records and DNS Updates

Feature	Description
SRV records	<p>Windows domain controllers use the SRV RR to advertise services to the network. This RR is defined in the RFC 2782, "A DNS RR for specifying the location of services (DNS SRV)." The RFC defines the format of the SRV record (DNS type code 33) as:</p> <pre>_ service . _ protocol . name ttl class SRV priority weight port target</pre> <p>There should always be an A record associated with target of the SRV record, so that the client can resolve the service back to a host. In the Windows implementation of SRV records, the records might look like this:</p> <pre>myserver.example.com A 10.100.200.11 _ldap._tcp.example.com SRV 0 0 389 myserver.example.com _kdc._tcp.example.com SRV 0 0 88 myserver.example.com _ldap._tcp.dc._msdcs.example.com SRV 0 0 88 myserver.example.com</pre>

	<p>An underscore always precedes the service and protocol names. In the example, <code>_kdc</code> is the Key Distribution Center. The priority and weight help you choose between target servers providing the same service (the weight differentiating those with equal priorities). With zero priority and weight, the listed order determines the priority. Windows domain controllers automatically place these SRV records in DNS.</p>
How SRV records are used	<p>When a Windows client boots up, it tries to initiate the network login process to authenticate against its Windows domain controller. The client must first discover where the domain controller is, and they do so using the dynamically generated SRV records. Before launching the net-login process, the client queries DNS with a service name; for example, <code>_ldap._tcp.dc._msdcs.example.com</code>. The DNS server SRV record target, for example, is <code>my-domain-controller.example.com</code>. The Windows client then queries DNS with the hostname <code>my-domain-controller.example.com</code>. DNS returns the host address and the client uses this address to find the domain controller. The net-login process fails without these SRV records.</p>
DNS updates	<p>When a Windows server is configured as a domain controller, you statically configure the name of the domain it manages through the Active Directory management console. This Windows domain should have a corresponding DNS zone associated with it. The domain controller should also have a series of DNS resolvers configured in its TCP/IP properties control panel.</p>
	<p>When the Windows domain controller boots up, it performs these steps to register itself in DNS and advertise its services to the network:</p> <ol style="list-style-type: none"> 1. Queries DNS asking for the start of authority (SOA) record for the DNS domain that mostly closely encapsulates its Windows domain. 2. Identifies the primary DNS server for the DNS zone (from the SOA record) that mostly closely encapsulates its Windows domain name. 3. Creates a series of SRV records in this zone using the RFC 2136 DNS Update protocol.
Server boot process log file examples	<p>Under normal operating conditions, the Cisco Prime Network Registrar primary DNS server writes these log entries when a Windows domain controller boots up and creates its SRV records:</p> <pre>data time name/dns/1 Activity Protocol 0 Added type 33 record to name "_ldap._tcp.w2k.example.com", zone "w2k.example.com" data time name/dns/1 Activity Protocol 0 Update of zone "w2k.example.com" from address [10.100.200.2] succeeded.</pre> <p>This log shows only one DNS update for a single SRV record. A Windows domain controller typically registers 17 of these SRV records when it boots up.</p>

Issues Related to Windows Environments

The table below describes the issues concerning interoperability between Windows and Cisco Prime Network Registrar. The information in this table is intended to inform you of possible problems before you encounter them in the field. For some frequently asked questions about Windows interoperability, see [Frequently Asked Questions About Windows Integration, on page 276](#).

Table 38: Issues Concerning Windows and Cisco Prime Network Registrar Interoperability

Issue	Description
Invisible dynamically created RRs	<p>Cisco Prime Network Registrar, when properly configured, accepts DNS updates from both DHCP and Windows servers. You can use the CLI to access the dynamic portion of the DNS zone for viewing and deleting records. Enter this command to view all DNS RRs in a given zone:</p> <pre>nrcmd> zone myzone listRR dynamic myfile</pre>
	<p>This redirects the output to the myfile file (see the following <i>Example: Output Showing Invisible Dynamically Created RRs</i> section). You can delete dynamically generated records by entering this command:</p> <pre>nrcmd> zone myzone removeDynRR myname [type]</pre> <p>You can also use nslookup to verify their existence, and you can use version 5. x (shipped with Windows) to view dynamic SRV records. In this version, use set type=SRV to enable viewing SRV records.</p>
Domain controller registration	<p>A Windows domain controller has to register itself in DNS using DNS updates. The DNS RFCs dictate that only a primary DNS server for a particular zone can accept edits to the zone data. Hence, the Windows domain controller has to discover which DNS server is the primary for the zone that includes its Windows domain name.</p> <p>The domain controller discovers this by querying the first DNS server in its resolver list (configured in the TCP/IP properties control panel). The initial query is for the SOA record of the zone that includes the Windows domain of the domain controller. The SOA record includes the name of the primary server for the zone. If no zone exists for the domain name, the domain controller keeps removing the left-most label of the domain name and sends queries until it finds an SOA record with a primary server included in that domain. Once the domain controller has the name of the primary DNS server for its domain, it sends it DNS updates to create the necessary SRV records.</p> <p>Ensure that the name of the zone primary DNS server is in its SOA record.</p>

<p>Failure of A record DNS updates</p>	<p>When a Windows domain controller tries to advertise itself to the network, it sends several DNS update requests to the DNS server of record for its domain. Most of these update requests are for SRV records. However, the domain controller also requests an update for a single A record of the same name as the Windows domain.</p> <p>If the Cisco Prime Network Registrar DNS server is also authoritative for a zone identical to this Windows domain, it rejects registering its A record, because the DNS A record update conflicts with the static SOA and NS records. This is to prevent possible security infractions, such as a dynamic host registering itself and spoofing Web traffic to a site.</p> <p>For example, the domain controller might control the w2k.example.com Windows zone. If a zone with the same name exists on the Cisco Prime Network Registrar DNS server, these RRs could be part of that zone. (Example follows.)</p> <pre>w2k.example.com. 43200 SOA nameserver.example.com. hostmaster.example.com. (98011312 ;serial 3600 ;refresh 3600 ;retry 3600000 ;expire 43200) ;minim w2k.example.com.86400 NS nameserver.example.com</pre> <p>The domain controller would try to add an additional record; for example:</p> <pre>w2k.example.com. 86400 A 192.168.2.1</pre> <p>Cisco Prime Network Registrar does not allow a DNS update to conflict with any statically configured name in the zone, even if the record type associated with that name is different. In the above example, an attempt to add an A record associated with the name w2k.example.com collides with the SOA and NS records.</p> <p>When the domain controller boots up, a DNS log file entry such as this appears:</p> <pre>0 8/10/2000 16:35:33 name/dns/1 Info Protocol 0 Error - REFUSED - Update of static name "w2k.example.com", from address [10.100.200.2]</pre> <p>This is how Cisco Prime Network Registrar responds to DNS updates of static DNS data. Additionally, you can ignore this DNS update failure. Windows clients do not use this A record. Allocation of domain controllers happens through SRV records. Microsoft added the A record to accommodate legacy NT clients that do not support SRV records.</p> <p>Note that failing to register the controller A record slows down the domain controller bootup process, affecting the overall login of worker clients. As mentioned earlier, the workaround is to define the Windows domain as a subdomain of the authoritative zone, or enable the DNS <code>serversimulate-zone-top-dynupdate</code> attribute. If this is not possible, contact the Cisco Technical Assistance Center for help.</p>
--	---

<p>Windows RC1 DHCP clients</p>	<p>Microsoft released Windows build 2072 (known as RC1) with a broken DHCP client. This client sends a malformed packet that Cisco Prime Network Registrar cannot parse. Cisco Prime Network Registrar drops the packet and cannot serve the client, logging this error:</p> <pre>08/10/2000 14:56:23 name/dhcp/1 Activity Protocol 0 10.0.0.15 Lease offered to Host:'My-Computer' CID: 01:00:a0:24:1a:b0:d8 packet'R15' until True, 10 Aug 2000 14:58:23 -0400. 301 ms.</pre> <pre>08/10/2000 14:56:23 name/dhcp/1 Warning Protocol 0 Unable to find necessary Client information in packet from MAC address:'1,6,00:d0:ba:d3:bd:3b'. Packet dropped!</pre> <p>Cisco Prime Network Registrar includes error checking specifically designed to deal with errors such as this improperly built FQDN option. However, if you encounter this problem, install the Microsoft patch to the RC1 client on the DHCP client. You must obtain this patch from Microsoft.</p>
<p>Windows plug-and-play network interface card (NIC) configuration</p>	<p>If configured to use DHCP, a Windows system tries to obtain a DHCP lease on startup. If no DHCP server is available, Windows may automatically configure the computer interface with a plug-and-play IP address. This address is not one that the network administrator or DHCP server configured or selected.</p> <p>These plug-and-play addresses are in the range 169.254.0.0/16. If you see devices in this address range on a network, it means that Windows autoconfigured the interfaces because it could not obtain a lease from a DHCP server.</p> <p>This can cause significant network and troubleshooting problems. The Windows system no longer informs the user that the DHCP client could not obtain a lease. Everything appears to function normally, but the client cannot route packets off its local subnet. Additionally, you may see the DHCP client trying to operate on the network with an address from the 169.254.0.0/16 network. This may lead you to think that the Cisco Prime Network Registrar DHCP server is broken and handing out the wrong addresses.</p>
	<p>If this problem occurs, perform these steps:</p> <ol style="list-style-type: none"> 1. Ensure that the DHCP client has an active network port and a properly configured NIC. 2. Ensure that the network between the client and the DHCP server(s) are properly configured. Ensure that all router interfaces are configured with the correct IPHelper address. 3. Reboot the DHCP client. 4. If necessary, look at the DHCP log file. If the DHCP client can successfully route packets to the server, this logs a DHCPDISCOVER, even if Cisco Prime Network Registrar does not answer the packet. <p>If the network is correctly configured, and if the DHCP client is not broken, Cisco Prime Network Registrar should receive the packet and log it. If there is no log entry for a packet receipt, there is a problem somewhere else in the network.</p>

Scavenging Windows client address records	Windows clients do not clean up after themselves, potentially causing their dynamic record registration to remain indefinitely. This leaves stale address records on the DNS server. To ensure that these stale records are periodically removed, you must enable scavenging for the zone (see Scavenging Dynamic Records, on page 264).
---	---

Example: Output Showing Invisible Dynamically Created RRs

```
Dynamic Resource Records _ldap._tcp.test-lab._sites 600 IN SRV 0
100 389 CNR-MKT-1.w2k.example.com. _ldap._tcp.test-lab._sites.gc._msdcs 600 IN
SRV 0 100 3268 CNR-MKT-1.w2k.example.com.
_ldap._tcp.test-lab._sites.gc._msdcs 600 IN SRV 0 100 88
CNR-MKT-1.w2k.example.com. _ldap._tcp.test-lab._sites.gc._msdcs 600 IN SRV 0
100 389 CNR-MKT-1.w2k.example.com. _ldap._tcp 600 IN SRV 0 100 389
CNR-MKT-1.w2k.example.com. _kerberos._tcp.test-lab._sites 600 IN SRV 0 100 88
CNR-MKT-1.w2k.example.com. _ldap._tcp.pdc._msdcs 600 IN SRV 0 100 389
CNR-MKT-1.w2k.example.com. _ldap._tcp.gc._msdcs 600 IN SRV 0 100 3268
CNR-MKT-1.w2k.example.com.
_ldap._tcp.1ca176bc-86bf-46f1-8a0f-235ab891bcd2.domains._msdcs 600 IN SRV 0 100
389 CNR-MKT-1.w2k.example.com. e5b0e667-27c8-44f7-bd76-6b8385c74bd7._msdcs 600
IN CNAME CNR-MKT-1.w2k.example.com. _kerberos._tcp.dc._msdcs 600 IN SRV 0 100
88 CNR-MKT-1.w2k.example.com. _ldap._tcp.dc._msdcs 600 IN SRV 0 100 389
CNR-MKT-1.w2k.example.com. _kerberos._tcp 600 IN SRV 0 100 88
CNR-MKT-1.w2k.example.com. _gc._tcp 600 IN SRV 0 100 3268
CNR-MKT-1.w2k.example.com. _kerberos._udp 600 IN SRV 0 100 88
CNR-MKT-1.w2k.example.com. _kpasswd._tcp 600 IN SRV 0 100 464
CNR-MKT-1.w2k.example.com. _kpasswd._udp 600 IN SRV 0 100 464
CNR-MKT-1.w2k.example.com. gc._msdcs 600 IN A 10.100.200.2
_ldap._tcp.test-lab._sites 600 IN SRV 0 100 3268 CNR-MKT-1.w2k.example.com.
```

Frequently Asked Questions About Windows Integration

These questions are frequently asked about integrating Cisco Prime Network Registrar DNS services with Windows:

What happens if both Windows clients and the DHCP server are allowed to update the same zone? Can this create the potential for stale DNS records being left in a zone? If so, what can be done about it?

The recommendation is not to allow Windows clients to update their zones. Instead, the DHCP server should manage all the client dynamic RR records. When configured to perform DNS updates, the DHCP server accurately manages all the RRs associated with the clients that it served leases to. In contrast, Windows client machines blindly send a daily DNS update to the server, and when removed from the network, leave a stale DNS entry behind.

Any zone being updated by DNS update clients should have DNS scavenging enabled to shorten the longevity of stale RRs left by transient Windows clients. If the DHCP server and Windows clients are both updating the same zone, three things are required in Cisco Prime Network Registrar:

1. Enable scavenging for the zone.
2. Configure the DHCP server to refresh its DNS update entries as each client renews its lease. By default, Cisco Prime Network Registrar does not update the DNS record again between its creation and its final deletion. A DNS update record that Cisco Prime Network Registrar creates lives from the start of the lease until the lease expires. You can change this behavior using a DHCP server (or DNS update configuration) attribute, *force-dns-updates*. For example:

```
nrcmd> dhcp enable force-dns-updates
```

```
100 Ok
force-dns-updates=true
```

3. If scavenging is enabled on a particular zone, then the lease time associated with clients that the DHCP server updates that zone on behalf of must be less than the sum of the *no-refresh-interval* and *refresh-interval* scavenging settings. Both of these settings default to seven days. You can set the lease time to 14 days or less if you do not change these default values.

What needs to be done to integrate a Windows domain with a pre-existing DNS domain naming structure if it was decided not to have overlapping DNS and Windows domains? For example, if there is a pre-existing DNS domain called `example.com` and a Windows domain is created that is called `w2k.example.com`, what needs to be done to integrate the Windows domain with the DNS domain?

In the example, a tree in the Windows domain forest would have a root of `w2k.example.com`. There would be a DNS domain named `example.com`. This DNS domain would be represented by a zone named `example.com`. There may be additional DNS subdomains represented in this zone, but no subdomains are ever delegated out of this zone into their own zones. All the subdomains will always reside in the `example.com` zone.

In this case, how are DNS updates from the domain controllers dealt with?

To deal with the SRV record updates from the Windows domain controllers, limit DNS updates to the `example.com` zone to the domain controllers by IP address only. (Later, you will also add the IP address of the DHCP server to the list.) Enable scavenging on the zone. The controllers will update SRV and A records for the `w2k.example.com` subdomain in the `example.com` zone. There is no special configuration required to deal with the A record update from each domain controller, because an A record for `w2k.example.com` does not conflict with the SOA, NS, or any other static record in the `example.com` zone.

The `example.com` zone then might include these records:

```
example.com. 43200 SOA ns.example.com. hostmaster.example.com. (
98011312 ;serial
3600 ;refresh
3600 ;retry
3600000 ;expire
43200 ) ;minimum
example.com.86400 NS ns.example.com
ns.example.com. 86400 A 10.0.0.10
_ldap._tcp.w2k.example.com. IN SRV 0 0 389 dc1.w2k.example.com
w2k.example.com 86400 A 10.0.0.25
...
```

In this case, how are zone updates from individual Windows client machines dealt with?

In this scenario, the clients could potentially try to update the `example.com` zone with updates to the `w2k.example.com` domain. The way to avoid this is to close down the zone to updates except from trusted sources. For Cisco Prime Network Registrar, you can use transaction signatures (TSIG) between the DHCP server and the primary DNS server for the `example.com` zone.

Configure the DHCP server to do DNS updates to the `example.com` zone and the appropriate reverse zone for each client, and use option 81 to prevent the clients from doing DNS updates themselves.

Has security been addressed in this case?

By configuring the forward and reverse zone to accept only updates from trusted IP addresses, you close the zone to updates from any other device on the network. Security by IP is not the most ideal solution, as it would not prevent a malicious attack from a spoofed IP address source. You can secure updates from the DHCP server by configuring TSIG between the DHCP server and the DNS server.

Is scavenging required in this case?

No. Updates are only accepted from the domain controllers and the DHCP server. The DHCP server accurately maintains the life cycle of the records that they add and do not require scavenging. You can manage the domain controller dynamic entries manually by using the Cisco Prime Network Registrar single-record dynamic RR removal feature.

What needs to be done to integrate a Windows domain that shares its namespace with a DNS domain? For example, if there is a pre-existing DNS zone called example.com and a Windows Active Directory domain called example.com needs to be deployed, how can it be done?

In this example, a tree in the Windows domain forest would have a root of example.com. There is a pre-existing domain that is also named example.com that is represented by a zone named example.com.

In this case, how are DNS updates from individual Windows client machines dealt with?

To deal with the SRV record updates, create subzones for:

```
_tcp.example.com.
_sites.example.com.
_msdc.example.com.
_msdc.example.com.
_udp.example.com.
```

Limit DNS updates to those zones to the domain controllers by IP address only. Enable scavenging on these zones.

To deal with the A record update from each domain controller, enable a DNS server attribute, *simulate-zone-top-dynupdate*.

```
nrcmd> dns enable simulate-zone-top-dynupdate
```

It is not required, but if desired, manually add an A record for the domain controllers to the example.com zone.

In this case, how are zone updates from individual Windows client machines dealt with?

In this scenario, the clients could potentially try to update the example.com zone. The way to avoid this is to close down the zone to updates except from trusted sources. For Cisco Prime Network Registrar, you can use transaction signatures (TSIG) between the DHCP server and the primary DNS server for the example.com zone.

Configure the DHCP server to do DNS updates to the example.com zone and the appropriate reverse zone for each client, and use option 81 to prevent the clients from doing DNS updates themselves.

Has security been addressed in this case?

By configuring the forward and reverse zone to accept only updates from trusted IP addresses, you close the zone to updates from other devices on the network. Security by IP is not the most ideal solution, as it would not prevent a malicious attack from a spoofed source. Updates from the DHCP server are more secure when TSIG is configured between the DHCP server and the DNS server.

Has scavenging been addressed in this case?

Yes. The subzones `_tcp.example.com`, `_sites.example.com`, `_msdc.example.com`, `_msdc.example.com`, and `_udp.example.com` zones accept updates only from the domain controllers, and scavenging was turned on for these zones. The `example.com` zone accepts DNS updates only from the DHCP server.

Configuring GSS-TSIG

Cisco Prime Network Registrar DNS Configuration to integrate with AD

To integrate AD with Cisco Prime Network Registrar DNS configuration, follow these steps:

Step 1 Install Cisco Prime Network Registrar DNS on a Workgroup machine.

Step 2 Create a zone (same as the domain of AD).

Install AD on a windows server using dcpromo.exe and integrate with Cisco Prime Network Registrar DNS.

Step 3 Ensure the SRV records are added in Cisco Prime Network Registrar DNS

```
DCHOSTNAME. DOMAIN.COM A AD-IP-ADDRESS
_ldap._tcp.DOMAIN.COM. SRV 0 0 389 DCHOSTNAME.DOMAIN.COM.
_kerberos._tcp.DOMAIN.COM. SRV 0 0 88 DCHOSTNAME.DOMAIN.COM.
_ldap._tcp.dc._msdcs.DOMAIN.COM. SRV 0 0 389 DCHOSTNAME.DOMAIN.COM.
_kerberos._tcp.dc._msdcs.DOMAIN.COM. SRV 0 0 88 DCHOSTNAME.DOMAIN.COM.
```

Note DCHOSTNAME refers to AD host name and DOMAIN.COM is the domain that exists in AD.

Bring Cisco Prime Network Registrar DNS and AD under the same domain in the windows environment:

Step 1 Change the domain, **Computer > Properties > Computer Name** > change the member of domain (same as the domain of AD).

Step 2 Control Panel > Network and Internet > Network and Sharing Center > Local Area Connection > Properties > TCP/IPV4 > Preferred DNS (Cisco Prime Network Registrar DNS running IP).

Step 3 Restart the computer, and login with the User that exists in AD.

Step 4 Login to AD and do the following:

- check the DNS active Hostname is added into, **AD Server Manager > Computers**

```
setspn -s DNS/ <hostname of the DNS server> <Computer Name>
```

Integrating the DNS server to AD-KDC

In Linux, the primary DNS server is integrated to AD-KDC:

Step 1 Ensure the /etc/krb5.conf or DNS server with SRV record is configured to reach the required AD.

```

krb5.conf configuration
[libdefaults]
ticket_lifetime = 24h
default_realm = <AD REALM>
default_tkt_enctypes = rc4-hmac
default_tgs_enctypes = rc4-hmac
dns_lookup_realm = true
dns_lookup_kdc = false
forwardable = true
<AD REALM> = {
    kdc = < AD-HOSTNAME>:88
    admin_server = =< AD-HOSTNAME:749
    default_domain = <AD REALM>
}

```

Note Ensure that the AD-HOSTNAME is resolvable.

Step 2 Create a service account in the Windows Server Active Directory:

1. Use the Active Directory Users and Computers Administrative Tool to create a new user account.

- Assign a user name to the account without any space.
- Assign a password to the account

Note Whenever the password expires/changed, the **keytab** file needs to be generated with a new associated *kvno*.

2. Assign a Service Principal Name (SPN) to the account utilizing the SETSPN.EXE. An SPN is the service-name/hostname/domain depending on the deployment. There can be multiple SPNs assigned to a single account.

For example, specify a <service-name> and a <hostname> where the service-name is DNS and the hostname is the fully qualified domain name of the machine on which the DNS server will be running.

```
setspn -s DNS/<DNS running Computer Name> <Service Name>
```

3. Get the *kvno* details:

```
ldifde -f <Filename> -d "DC=<DOMAIN>,DC=com" -l *,msDS-KeyVersionNumber -r
"(serviceprincipalname=<service-principal name>)" -p subtree OR kvno.exe <service-principal
name>@<REALM>
```

4. Generate the Keytab file using the ktpass.exe command:

```
ktpass -out<filename> -princ <Principal name> -pass <password associated with the user> -crypto
all -ptype KRB5_NT_PRINCIPAL -kvno <Kvno details>
```

Transfer the keytab file to the Linux machine and run Kutil to add the Keytab entry to the existing Keytab file:

```
> ktutil
ktutil: rkt <keytab file name>
ktutil: wkt /etc/krb5.keytab
ktutil: q
```

Step 3 Display the keytab entry using:

```
klist -k -t -e /etc/krb5.keytab
```

Primary DNS Server on Linux Integrated to MIT-KDC

To associate the service-principal name to MIT KDC:

Step 1 Login to the Linux DNS server and use kadmin utility to add the principal name to the MIT-KDC:

```
>kadmin
Authenticating as principal <MIT-KDC USER@REALM> with password.
Password for <MIT-KDC USER@REALM.COM > : <Enter the associated Password>
kadmin: addprinc -randkey DNS/<hostname of the DNS server>
WARNING: no policy specified for DNS/<hostname of the DNS server>@REALM; defaulting to no policy
add_principal: Principal or policy already exists while creating " DNS/<hostname of the DNS
server>@REALM".
kadmin: ktadd -randkey DNS/<hostname of the DNS server>
kadmin: Principal -randkey does not exist.
Entry for principal DNS/<hostname of the DNS server> with kvno x, encryption type AES-256 CTS mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal DNS/<hostname of the DNS server>with kvno x, encryption type AES-128 CTS mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal DNS/<hostname of the DNS server>with kvno x, encryption type Triple DES cbc mode
with HMAC/shal added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal DNS/<hostname of the DNS server>with kvno x, encryption type ArcFour with HMAC/md5
added to keytab WRFILE:/etc/krb5.keytab.
kadmin: quit
```

Step 2 Display the keytab entry using:

```
klist -k -t -e /etc/krb5.keytab
```

Step 3 Login to the MIT-KDC running LINUX server and check the added principal name has the same kvno associated as above using the command:

```
Kvno DNS/<hostname of the DNS server>
```

Troubleshooting GSS-TSIG Configuration

To get the details of GSS/SSPI failure and major/minor status, enable the DEBUG options in the DNS server and set the value of g=3.

- "The key version number for the principal in the key table is incorrect."

The Kvno returned by, `klist -k -t -e /etc/krb5.keytab` in the DNS running machine should be the same kvno in KDC.

Verification of kvno in AD-KDC:

```
ldifde -f c:\spn1_out.txt -d "DC=TIG,DC=com" -l *,msDS-KeyVersionNumber -r
"(serviceprincipalname=DNS/WIN-CPNUV*)" -p subtree
```

Verification of kvno is MIT- KDC:

```
Kvno <principal name>
```

- "Wrong Principal Name"

Ensure that the GSS Client and the server are using the same service-key that is used to encrypt/decrypt the service ticket.

Troubleshooting DNS Update

You can use a standard DNS tool such as **dig** and **nslookup** to query the server for RRs. The tool can be valuable in determining whether dynamically generated RRs are present. For example:

```
$ nslookup

default Server: server2.example.com
Address: 192.168.1.2
> leasehost1.example.com

Server: server2.example.com
Address: 192.168.1.100
> set type=ptr

> 192.168.1.100

Server: server2.example.com
Address: 192.168.1.100
100.40.168.192.in-addr.arpa name = leasehost1.example.com
40.168,192.in-addr.arpa nameserver = server2.example.com
```

You can monitor DNS updates on the DNS server by setting the *log-settings* attribute to *ddns*, or show even more details by setting it to *ddns-details*.



CHAPTER 10

Managing Client-Classes and Clients

Use the Cisco Prime Network Registrar client and client-class concepts to provide differentiated services to users across a common network. You can group clients based on administrative criteria, and then ensure that each group receives its appropriate class of service (COS). Without client-class processing, the DHCP server provides client leases based solely on their network location.

- [Configuring Client-Classes, on page 283](#)
- [Troubleshooting Client-Classes, on page 290](#)
- [Configuring Clients, on page 291](#)
- [Subscriber Limitation Using Option 82, on page 297](#)
- [Configuring Cisco Prime Network Registrar to Use LDAP, on page 301](#)

Configuring Client-Classes

You can differentiate client services in the following ways:

- Register clients using the Cisco Prime Network Registrar database (this section) or the Lightweight Directory Access Protocol (see [Configuring Cisco Prime Network Registrar to Use LDAP, on page 301](#)).
- Register intermediary devices (such as cable modems) so that you can differentiate their upstream clients by class of service.
- Use the contents of client packets without the foreknowledge of client data:
 - Known DHCP options that can be in the packet, such as the *dhcp-user-class-id* DHCP option (77), or the *radius-attribute* suboption of the *relay-agent-info* DHCP option (82, see [Processing Client Data Including External Sources, on page 289](#)).
 - Other data in the packet to extract using an expression in the *client-class-lookup-id* DHCP server attribute (see [Calculating Client-Classes and Creating Keys, on page 298](#)).
- Use a two-stage process of first creating a client-class to assign clients, then set a *client-lookup-id* for certain clients (see [Expression Processing for Subscriber Limitation, on page 299](#)).

Related Topics

[Client-Class Process, on page 284](#)

[Defining Client-Classes, on page 284](#)

[Setting Selection Tags on Scopes and Prefixes, on page 286](#)

[Defining Client-Class Hostname Properties, on page 287](#)

[Editing Clients and Their Embedded Policies, on page 293](#)

[Processing Client Data Including External Sources, on page 289](#)

[Troubleshooting Client-Classes, on page 290](#)

Client-Class Process

Enable or disable client-class processing for the DHCP server and apply a set of properties to groups of clients. With client-class enabled, the server assigns the client to an address from a matching DHCPv4 scope or DHCPv6 prefix. The server acts according to the data in the packet. To configure client-classes:

1. Enable client-class processing for the DHCP server.
2. Define client-classes that include or exclude selection tags (criteria).
3. Apply the selection tags to specific scopes or prefixes (or their templates).
4. Assign clients to these classes.

This process is for clients configured through Cisco Prime Network Registrar. For processing affected by data from external sources, see [Processing Client Data Including External Sources, on page 289](#).

Defining Client-Classes

You enable and define client-classes at the server level.

Local Basic Web UI

-
- Step 1** Enable client-classes. In the Basic or Advanced mode:
- a) From the **Deploy** menu, choose **DHCP Server** under the **DHCP** submenu to open the Manage DHCP Server page.
 - b) Select the server on the DHCP Server pane.
 - c) On the Edit DHCP Server tab, enable the *client-class* attribute.
 - d) Click **Save**.
- Step 2** From the Design menu, choose **Client Classes** under the **DHCP Settings** submenu to open the List/Add DHCP Client Classes page.
- Step 3** Click the **Add Client Classes** icon in the **Client Classes** pane to open the Add DHCP Client Class dialog box.
- Step 4** Enter a name for the client-class.
- Step 5** Set other client-class properties. The hostname and domain name attributes are mainly used for DNS updates if not using a DNS update configuration (see [Creating DNS Update Configurations, on page 256](#)). The hostname properties are described in [Defining Client-Class Hostname Properties, on page 287](#). You can also choose the appropriate policy for the client-class.
- Step 6** Click **Add Client Class**.
- Step 7** Define the selection criteria.

The critical step in creating a client-class is defining its selection criteria so that you can associate the client-class with a DHCPv4 scope or DHCPv6 prefix. Use the *selection-criteria* attribute (see also [Table 39: Selection Tag and Criteria Attributes Used](#)).

You can enter multiple selection tags by separating them with commas. The values have to match the selection tags set for the desired scope or prefix (see [Setting Selection Tags on Scopes and Prefixes, on page 286](#)).

- Step 8** To add an embedded policy to the client-class, see [Editing Clients and Their Embedded Policies, on page 293](#).
 - Step 9** Click **Save**.
 - Step 10** Debug as needed. To debug client-class errors, set the DHCP log settings to **client-criteria-processing**.
 - Step 11** To delete a client-class, select the client and click the **Delete** icon in the Client Classes pane, and confirm the deletion.
-

Local Advanced Web UI

- Step 1** Enable client-classes. In the Basic or Advanced mode:
 - a) From the **Deploy** menu, choose **DHCP Server** under the **DHCP** submenu to open the Manage DHCP Server page.
 - b) Select the server on the DHCP Server pane.
 - c) On the Edit Local DHCP Server tab, enable the *client-class* attribute.
 - d) Click **Save**.
 - Step 2** From the **Design** menu, choose **Client Classes** under the **DHCP Settings** submenu to open the List/Add DHCP Client Classes page.
 - Step 3** Click the **Add Client Classes** icon in the **Client Classes** pane to open the Add DHCP Client Class dialog box.
 - Step 4** Enter a name for the client-class.
 - Step 5** Set other client-class properties. The hostname and domain name attributes are mainly used for DNS updates if not using a DNS update configuration (see [Creating DNS Update Configurations, on page 256](#)). The hostname properties are described in [Defining Client-Class Hostname Properties, on page 287](#). You can also choose the appropriate policy for the client-class.
 - Step 6** Click **Add DHCP Client Class**.
 - Step 7** Define the selection criteria.

The critical step in creating a client-class is defining its selection criteria so that you can associate the client-class with a DHCPv4 scope or DHCPv6 prefix. Use the *selection-criteria* attribute (see also [Table 39: Selection Tag and Criteria Attributes Used , on page 287](#)).

You can enter multiple selection tags by separating them with commas. The values have to match the selection tags set for the desired scope or prefix (see [Setting Selection Tags on Scopes and Prefixes, on page 286](#)).
 - Step 8** To add an embedded policy to the client-class, see [Setting Selection Tags on Scopes and Prefixes, on page 286](#).
 - Step 9** Click **Save**.
 - Step 10** Debug as needed. To debug client-class errors, set the DHCP log settings to **client-criteria-processing**.
 - Step 11** To delete a client-class, select the client and click the **Delete** icon in the Client Classes pane, and confirm the deletion.
-

CLI Commands

Enable client-classes by using **dhcp enable client-class**. To create the client-class, use **client-class name create**. The name should clearly identify its intent. It is not case-sensitive; classPC is the same as Classpc.

Set properties of the clients in the client-class by using **client-class name set attribute=value**. For example, set the desired policy to associate with the client-class by using **client-class name set policy-name=value**.

Associate a scope with the client-class by using **client-class name set selection-criteria**. (See the [Setting Selection Tags on Scopes and Prefixes, on page 286](#)).

Show the properties of a created client-class by using **client-class name [show]**. You can also list the properties for all the client-classes created, or list just their names. To debug the client-class processing, use **dhcp set log-settings=client-criteria-processing**. To delete the client-class, use **client-class name delete**.

Configuring DHCPv6 Client-Classes

You can configure DHCPv6 client-class attributes, which are:

- **v6-client-lookup-id**—Key value to use to look up the DHCPv6 client in the client database (locally or through LDAP), specified as an expression that evaluates to a string (or a blob as a valid string).
- **v6-override-client-id**—Value that replaces any client-identity value in an incoming packet, specified as an expression that evaluates to a blob.

Local Advanced Web UI

From the **Design** menu, choose **Clients** under the **DHCP Settings** submenu to open the List/Add DHCP Clients page. Select an existing client to open the Edit DHCP Client page or click the **Add Clients** icon on Clients pane to add a new client-class on the List/Add DHCP Client page, choose the client-class that includes the DHCPv6 attributes that were set (see [Configuring DHCPv6 Client-Classes, on page 286](#)), then click **Save**.



Tip Disable the *validate-client-name-as-mac* attribute for the DHCP server.

CLI Commands

Use **client list** or **client name show** to show the existing clients. To set the client-class name for the client, use **client name set client-class-name=value**. Also ensure that the *validate-client-name-as-mac* attribute is disabled for the DHCP server.

Setting Selection Tags on Scopes and Prefixes

To assign clients to different address pools, you must define the DHCPv4 scope (or template) or DHCPv6 prefix (or template) with the selection tags that you specified in the selection-criteria for the client-class. All the selection-criteria tags that the client-class has must match the tags the scope or prefix has, even though the scope or prefix might have additional tags. If the client-class omits all selection-criteria, no limitations apply to the scope or prefix selection.

For example:

Scope A has tag1, tag2

Scope B has tag3, tag4

Assuming both scopes are on the same network, a client in a client-class with:

- Tag1, tag2, or both, gets leases from scope A.
- Tag3, tag4, or both, gets leases from scope B.
- One or more tags from both scopes (such as tag1 and tag3) does not get leases from either scope.
- No tags gets leases from either scope.

The table below describes the attributes Cisco Prime Network Registrar uses to refer to selection tags or selection criteria for network objects.

Table 39: Selection Tag and Criteria Attributes Used

Object	Attribute
Client	selection-criteria
Client-class	selection-criteria
Scope	selection-tag-list
Scope template	selection-tag-list
Prefix	selection-tags
Prefix template	selection-tags
Address block	selection-tags
Subnets	selection-tags

Local Basic or Advanced Web UI

Create or edit a scope or prefix or its template; on the Add or Edit page for the scope or prefix (or its template), find the Selection Tags attribute and enter a list of comma-separated selection tags created in the *selection-criteria* attribute for the client-class that you want to associate with this scope or prefix (or its template). Then save the changes and reload the DHCP server, if necessary.

CLI Commands

Use **scope name set selection-tag-list**. For a scope template, use **scope-template name set selection-tag-list**. For a prefix, use **prefix name set selection-tags**. For a prefix template, use **prefix-template name set selection-tags**.

Defining Client-Class Hostname Properties

You can specify the hostname that each client should adopt, using the Hostname (*host-name*) attribute of the client-class. This can be an absolute, valid DNS value to override the one included in the DHCP client request, or can be any of these:

- **@host-name-option**—The server uses whatever hostname option the client sent.
- **@no-host-name-option**—The server ignores the hostname that the client sends. If DNS name generation is in effect, the server uses a generated name, if set up as such for dynamic DNS updating.
- **@use-macaddress**—The server synthesizes a hostname from the client MAC address, hyphenates the octets, then adds an x at the front. For example, if a client MAC address is 1,6:00:d0:ba:d3:bd:3b, the synthesized hostname would be x1-6-00-d0-ba-d3-bd-3b.

If you omit a value, the hostname is unspecified. You can also synthesize hostnames by using a DNS update configuration (see [Creating DNS Update Configurations, on page 256](#)).

Related Topics

- [Editing Clients and Their Embedded Policies, on page 293](#)
- [Processing Client Data Including External Sources, on page 289](#)
- [Troubleshooting Client-Classes, on page 290](#)
- [Subscriber Limitation Using Option 82, on page 297](#)
- [Configuring Cisco Prime Network Registrar to Use LDAP, on page 301](#)

Editing Client-Classes and Their Embedded Policies

Editing a client-class involves the same attributes as creating a client-class. You can also add and modify an embedded policy for the client-class so that you can set its policy options. The embedded policy has no properties or DHCP options associated with it until you add them. (See also [Creating and Editing Embedded Policies, on page 170](#)). The client-class embedded policy setting is the third priority the DHCP server uses in its policy selection, after that set for the client itself (see [DHCPv4 Policy Hierarchy, on page 164](#)).

Local Advanced Web UI

-
- Step 1** Create the client-class.
 - Step 2** Select the client-class on the List/Add DHCP Client Classes page to open the Edit DHCP Client Class page.
 - Step 3** Make changes to attribute settings as required.
 - Step 4** To add a new embedded policy for the client-class, click **Create New Embedded Policy**. If there is an existing embedded policy that you want to edit, click **Edit Existing Embedded Policy**. Both actions open the Edit DHCP Embedded Policy for Client-Class page. (If you want to unset the existing embedded policy, click **Unset** on the Edit DHCP Client-Class page; this resets the button to **Create New Embedded Policy**.)
 - a) Modify the fields, options, and attributes on this page. For example, under the DHCPv4 Options, set the client lease time by choosing **dhcp-lease-time [51]** from the drop-down list, enter a lease interval value in the Value field, then click **Add Option**. If necessary, unset attribute values.
 - Step 5** Click **Save**.
-

CLI Commands

To check if there are any embedded policy values already set for a client-class, use **client-class-policy *client-class-name* show**. To set the attributes for the embedded policy, use **client-class-policy *client-class-name* set *attribute* =*value***.

To set the DHCP options, use one of these commands:

```
nrcmd> client-class-policy client-class-name setOption {opt-name | id} value [-blob]
nrcmd> client-class-policy client-class-name setV6Option {opt-name | id}[.instance] value
[-blob]
nrcmd> client-class-policy client-class-name setVendorOption {opt-name | id} opt-set-name
value [-blob]
nrcmd> client-class-policy client-class-name setV6VendorOption {opt-name | id} opt-set-name
value [-blob]
```

To set the lease time, use **client-class-policy *client-class-name* setLeaseTime *value***.

Processing Client Data Including External Sources

Information about network hosts running DHCP clients and their users can arrive at the DHCP server from several external sources. The server can use this data as part of client-class processing, and capture it in its lease database to make it available to the Cisco Prime Network Registrar management system.

Recently introduced external factors that can influence client definitions are:

- A *subscriber-id* suboption of the *relay-agent-info* DHCP option (82), whereby a network administrator defines a network subscriber or client and sends this data to the DHCP server.
- RADIUS authentication server data, as part of 802.1x protocol deployments where the RADIUS data can be helpful in DHCP decision making. In this case, a device can send the data as part of *radius-attribute* suboption attributes in the *relay-agent-info* DHCP option (82).

Both these external options use DHCP option 82, as described in [Subscriber Limitation Using Option 82, on page 297](#). The RADIUS source can end the following attributes:

- Client user or account name (the user attribute)
- Administratively defined class string (the class attribute)
- Vendor-specific data (the vendor-specific attribute)
- Session timeout value (the session-timeout attribute)
- IP address pool to use for the client (the framed-pool attribute)
- IPv6 address pool to use for the client (the framed-ipv6-pool attribute)

Cisco Prime Network Registrar provides extension support for the *subscriber-id* suboption and the user, class, and framed-pool attributes of the RADIUS suboption, and expression support for all of the suboptions (see [Using Expressions, on page 315](#)). Additionally, the DHCP server now includes attribute settings to configure how the server handles the RADIUS class and framed-pool attributes. Cisco Prime Network Registrar can use the server attributes to map the RADIUS attribute value as a selection tag or client-class name, or append the value to the selection tag that it finds in its client database. For example:

```
nrcmd> dhcp set map-radius-class=append-to-tags
```

For client-classes and selection tags determined from external resources such as RADIUS, the processing order is slightly more complex than that described in [Client-Class Process, on page 284](#). See the following subsections. Remember that to use the client-class feature, you must enable the DHCP server *client-class* attribute.

Related Topics

[Processing Order to Determine Client-Classes, on page 289](#)

[Processing Order to Determine Selection Tags, on page 290](#)

Processing Order to Determine Client-Classes

The order in which the DHCP server uses possible sources to determine client-class names is as follows:

1. It uses the client-class name in the extension environment dictionary.
2. If it finds a real client-entry in the database, it uses its *client-class-name*. (To prevent this unnecessary database read, enable the *skip-client-lookup* DHCP server attribute; see [Skipping Client Entries for Client-Classing, on page 295](#).)
3. If you map the RADIUS framed-pool value to a client-class (by using `dhcp set map-radius-pool-name=map-as-class`), it uses the framed-pool value.

4. If you map the RADIUS class value to a client-class (by using **dhcp set map-radius-class=map-as-class**), it uses the class value.
5. If you map the *dhcp-user-class-id* DHCP option (77) to a client-class (by using **dhcp set map-user-class-id=map-as-class**), it uses the option value. (Note that you can alternatively use a lookup ID expression instead of this mapping; see [Client-Class Lookup Expression Processing, on page 298](#).)
6. If it finds no mapping or user-class ID, it uses the default-client-class-name from the environment dictionary.
7. If it finds no default-client-class-name or client-entry, it uses the client-class-name from the client named **default** (if found).

Processing Order to Determine Selection Tags

The order in which the server uses the possible sources to determine selection tags (it uses the first nonnull source) is as follows:

1. Selection tags in the extension environment dictionary.
2. If it finds a real client-entry in the database, it uses the client-entry *selection-tags*. (To prevent this unnecessary database read, enable the *skip-client-lookup* DHCP server attribute; see [Skipping Client Entries for Client-Classing, on page 295](#).)
3. Selection tags in the client-class.
4. If you map an available RADIUS framed-pool value to a tag (by using **dhcp set map-radius-pool-name=map-as-tag**), it uses that tag.
5. If you map an available RADIUS class value to a tag (by using **dhcp set map-radius-class=map-as-tag**), it uses that tag.
6. If you map an available *dhcp-user-class-id* DHCP option (77) to a tag (by using **dhcp set map-user-class-id=map-as-tag**), it uses that tag.

Next, the server could append one of the following to the list of selection tags (if any):

1. If a RADIUS framed-pool value is available and you set the *map-radius-pool* DHCP attribute to append to the tags (by using **dhcp set map-radius-pool=append-to-tags**), the server appends it.
2. If a RADIUS class value is available and you set the *map-radius-class* DHCP attribute to append to the selection tags (by using **dhcp set map-radius-class=append-to-tags**), the server appends it.
3. If a *dhcp-user-class-id* is available and you set the *map-user-class-id* DHCP attribute to append to the selection tags (by using **dhcp set map-user-class-id=append-to-tags**), the server appends it.

Troubleshooting Client Classes

To troubleshoot a client-class, enable client-class logging using the *log-settings* attribute on the Edit DHCP Server page of the web UI, or **dhcp set log-settings=setting** in the CLI, then reload the DHCP server (if in staged dhcp edit mode). The recommended settings are:

- **client-detail**—Logs a single line at the end of every client-class client lookup operation. This line shows all the data found for the client as well as the data that was found in the client-class.
- **client-criteria-processing**—Logs a message whenever the server examines a scope or prefix to find an available lease or to determine if a lease is still acceptable for a client that already has one.
- **ldap-query-detail**—Logs messages whenever the DHCP server initiates a lease state entry creation to an LDAP server, receives a response from an LDAP server, or retrieves a result or error message from an LDAP server.
- If the problem could be related to your LDAP server, also enable the LDAP *can-query* setting.

These logs will help answer these questions:

- Is the server reading the client entry from the expected database?

The server can read the client entry from LDAP or CNRDB (the Cisco Prime Network Registrar internal database). The *client-detail* log shows you from where the server is reading the client entry.

- Is client-class enabled?

If enabled but you are getting unexpected results, verify from which database is your Cisco Prime Network Registrar server reading clients. Is it reading from LDAP or CNRDB? The *ldap-query-detail* log tells you if it is reading from LDAP. If not, enable the DHCP *use-ldap-client-data* property.



Note Using LDAP requires configuring the LDAP server for queries. Enable the LDAP *can-query* attribute. You also must configure the DHCP server to use LDAP for queries.

- Is the server providing clients the right data, but you see the wrong results from that data (for example, clients are not receiving the expected IP addresses)?

Verify the explicit relationships on your network. The *client-criteria-processing* log shows from which scopes or prefixes the server is getting addresses. If it does not get addresses from the expected sources, explicit relationships might be incorrectly defined. A scope that you thought was a secondary scope might not be defined that way.

- In Expert mode, did you set the include and exclude criteria for selection tags properly?

If you define a series of selection tags to include, the tags of a scope or prefix must match those of the client. In Expert mode, you can also use a *selection-criteria-excluded* attribute on the client-class to exclude selection tags. If you define a series to exclude, a scope or prefix must have none of these tags defined so that the client can get configuration parameters from it. Avoid complex inclusion and exclusion scenarios as you begin working with selection tags.

Configuring Clients

DHCP client properties include the participating client-class and associated policy for a client, the action to perform, and the inclusion and exclusion criteria for selection tags. A client inherits the properties from its client-class, which you may choose to override or supplement by specifying different properties for the client.

Local Basic or Advanced Web UI

- Step 1** From the **Design** menu, choose **Clients** under the **DHCP Settings** submenu to open the List/Add DHCP Clients page.
- Step 2** Click the **Add Clients** icon in the Clients pane to open the Add DHCP Client dialog box, enter the client identity, typically a MAC address, but it can also be a DUID or lookup key. (Note that you can set up the DHCP server to validate the client name as a MAC address by enabling the server attribute *validate-client-name-as-mac*.)

You can also create a client named **default** that does not have a specific client configuration. For example, you can have a client always use its MAC address for its hostname.

Step 3 Select a client-class name, if desired, from the drop-down list of predefined client-classes.

Step 4 Click **Add DHCP Client**. This opens the Edit DHCP Client page.

The critical step in creating a client is defining its selection criteria so that you can associate the client with a scope or prefix (unless the selection criteria were already set up for the client-class associated with the client).

Use the *selection-criteria* attribute under the Attribute list (see also [Table 39: Selection Tag and Criteria Attributes Used](#), on page 287). You can enter multiple selection tags by separating them with commas. The values have to match the selection tags set for the desired scope or prefix (see [Setting Selection Tags on Scopes and Prefixes](#), on page 286).

Note If you chose a client-class for the client, this page does not appear, and the client name is listed on the List/Add Client page.

Step 5 Set other attributes as desired. For example:

- Set the *host-name* attribute to *@no-host-name-option* to provide provisional addresses to unknown clients. See [Allocating Provisional Addresses](#), on page 294.
- Set the domain name of the zone to use when performing dynamic DNS updates.
- Set the policy and action for the client. With the *exclude* action, the server ignores all communication from this client (no packets are shown); with the *one-shot* action, the server does not renew or re-offer a lease to this client.
- Choose the number of time units (seconds, minutes, hours, days, weeks), or UNIX style date (such as Mar 24 12:00:00 2002) to indicate when the authentication expires, or use **forever**.

Step 6 Click **Save** at the bottom of the page.

Step 7 Debug as needed. To debug client errors, set the DHCP log settings to **client-criteria-processing**.

Step 8 To delete a client, click the **Delete** icon in the Clients pane, and confirm the deletion.

CLI Commands

To create a client, use **client name create**. To associate a client-class with the client, use **client name set client-class-name=value**. To set selection criteria for scopes or prefixes, use **client name set selection-criteria**. To set other attributes, use **client name set attribute=value**.

To display client properties, use **client name [show]**. To display properties for all the clients, use **client list**, or **client listnames** to list just the names. To debug clients, use **dhcp set log-settings=client-detail**. To delete a client, use **client name delete**.

Related Topics

[Editing Clients and Their Embedded Policies](#), on page 293

[Setting Windows Client Properties](#), on page 294

[Allocating Provisional Addresses](#), on page 294

[Skipping Client Entries for Client-Classing](#), on page 295

[Limiting Client Authentication](#), on page 295

[Setting Client Caching Parameters](#), on page 296

Editing Clients and Their Embedded Policies

Editing a client involves the same attributes as creating a client. You can also add and modify an embedded policy for the client so that you can set its policy options. The embedded policy has no properties or DHCP options associated with it until you add them. (See also [Creating and Editing Embedded Policies, on page 170](#).) The client embedded policy setting is the first priority the DHCP server uses in its policy selection (see [DHCPv4 Policy Hierarchy, on page 164](#)).

Local Basic or Advanced Web UI

-
- Step 1** Create the client.
- Step 2** Select the client from the Clients pane on the List/Add DHCP Clients page to open the Edit DHCP Client page.
- Step 3** Make changes to attribute settings as required.
- Step 4** To add a new embedded policy for the client-class, click **Create New Embedded Policy**. If there is an existing embedded policy that you want to edit, click **Edit Existing Embedded Policy**. Both actions open the Edit DHCP Embedded Policy for Client page. (This page is almost identical to the Edit DHCP Embedded Policy for Client-Class page.)
- Modify the fields, options, and attributes on the Edit DHCP Embedded Policy for Client page. For example, under the DHCPv4 Options, set the client lease time by choosing **dhcp-lease-time [51]** from the drop-down list, enter a lease interval value in the Value field, then click **Add Option**. If necessary, unset attribute values.
- If you want to unset the existing embedded policy, click **Unset** on the Edit DHCP Client page; this resets the button to **Create New Embedded Policy**.
- Step 5** Click **Save**.
-

CLI Commands

To see if there are any embedded policy values already set for a client, use **client-policy *client-name* show**. To create an embedded policy, use **client-policy *client-name* set *attribute* =*value***.

To set the DHCP options, use one of these commands:

```
nrcmd> client-policy client-name setOption <opt-name | id> value [-blob]
nrcmd> client-policy client-name setV6Option <opt-name | id>[.instance] value [-blob]
nrcmd> client-policy client-name setVendorOption <opt-name | id> opt-set-name value [-blob]
nrcmd> client-policy client-name setV6VendorOption <opt-name | id> opt-set-name value [-blob]
```

To set the lease time, use **client-policy *client-name* setLeaseTime *value***.

Configuring DHCPv6 Clients

You can configure DHCPv6 clients.

Local Advanced Web UI

From the **Design** menu, choose **Clients** under the **DHCP Settings** submenu to open the List/Add DHCP Clients page. Select an existing client to open the Edit DHCP Client page or click the **Add Clients** icon on Clients pane to add a new client-class on the List/Add DHCP Client page, choose the client-class that includes the DHCPv6 attributes that were set (see [Configuring DHCPv6 Client-Classes, on page 286](#)), then click **Save**.



Tip Disable the *validate-client-name-as-mac* attribute for the DHCP server.

CLI Commands

Use **client list** or **client name show** to show the existing clients. To set the client-class name for the client, use **client name set client-class-name=value**. Also ensure that the *validate-client-name-as-mac* attribute is disabled for the DHCP server.

Setting Windows Client Properties

Windows clients support class-based provisioning. You can set certain properties that relate to client-class processing. These are:

- Look up the client entry to determine the default client for client-class processing.
- Map the user class ID to the client-class or selection tag.
- Set whether to append the class ID to the selection tag name.

Settings in Windows Clients

On the Windows client host, use **ipconfig /setclassid** to set the class ID. If you plan to map this client ID to a client-class or selection tag, it must have the same name. Then confirm by using **ipconfig /showclassid**. For example:

```
DOS> ipconfig /setclassid adapter engineering
```

```
DOS> ipconfig /showclassid adapter
```

Settings in DHCP Servers

You must set Windows client properties in the DHCP server.

Use DHCP server attributes in the local cluster web UI or **dhcp set** command attributes in the CLI to set the Windows client properties for the server. If you set the *skip-client-lookup* attribute to true (the default is false), the DHCP server skips the client entry for client-class processing. (See [Skipping Client Entries for Client-Classing](#), on page 295.) Use one of the *map-user-class-id* attribute settings:

- **0**—Ignore the user class ID (the default)
- **1**—Map the user class ID to the selection tag
- **2**—Map the user class ID to the client-class
- **3**—Append the user class ID to the list of selection tags

Allocating Provisional Addresses

You can provide provisional addresses to clients.

Provisional Addresses for Unknown Clients

The DHCP server can allocate provisional addresses to unknown clients for a short time on a one-shot basis. The server gives an address to the unknown client only as long as its lease period (which should be set short),

and the client cannot renew the lease. Once the lease expires, the client cannot obtain a new lease until after the grace period expires (this locks the client out of network access). The idea is to give the client a short time to register and prevent it from using the network if it does not register in that time.

-
- Step 1** Create an **unknown** policy, for example (the name is arbitrary).
- Step 2** Use the *Grace period* field on the Edit DHCP Policy page, or the **policy unknown create grace-period=extended-time** setting in the CLI.
- Step 3** Use the **default** client to set the *Policy name* value to **unknown**, and the *action* attribute value to **one-shot** on the Edit DHCP Client page, or use **client default create policy-name=unknown action=one-shot** in the CLI, to give provisional addresses to unknown clients.
- Note** Provisioning unknown clients is not supported in DHCPv6.
-

Using One-Shot Action

Use the one-shot action to allocate provisional addresses. This is useful when you want a client to have an address for only a short time. Configure the default client (or the client-class that the default client specifies) by setting the *action* attribute to **one-shot**.

The server then gives a lease to an unknown client, but does not allow it to renew the lease. When the lease expires, the server does not respond to that client during the lease grace period, and only responds when a different client uses the lease. The grace period, therefore, is the minimum period during which the client cannot obtain a lease.

You can allow the client a relatively short lease time, such as one day, and specify a long grace period, such as two weeks. This way you can offer an incentive to the client to register with some authority and become a known client, while not re-allocating the lease to another client. After the lease expires, the client cannot get another address for the lease for the two-week grace period.

You can configure the lease and grace period differently for each scope or prefix, so that provisional leases can have different lease and grace periods than nonprovisional ones. Provisional addresses are less restrictive if you use multiple DHCP servers, because each server operates its one-shot capabilities independently. With the approach described and two DHCP servers, an unregistered client can get two days of provisional address use every two weeks.

Skipping Client Entries for Client-Classing

You may not want to honor client entries for client-classing to prevent unnecessary database reads. To accomplish this, enable the *skip-client-lookup* DHCP server attribute (**dhcp enable skip-client-lookup** in the CLI).

Limiting Client Authentication

By default, client entries get unlimited authentication. Using the *authenticate-until* attribute, you can limit authenticating a client entry by specifying an expiration time.

When a client entry is no longer authenticated, the DHCP server uses the *unauthenticated-client-class-name* attribute value for the name of the client-class entry to use in answering this DHCP request. If this attribute is unset, or if there is no client-class entry in it, the DHCP server ignores the request.

The valid client authentication values are:

- **+num unit**—Time in the future, where *num* is a decimal number and *unit* is *s*, *m*, *h*, *d*, or *w* for seconds, minutes, hours, days or weeks, respectively. For example, “+3w” is three weeks in the future.
- **date**—Month, day, 24-hour, and 2-or-4-digit-year. For example, “Jun 30 20:00:00 2002.” Enter the local process time. If the server runs in another time zone, disregard the time zone and use local time instead.
- **forever**—Does not expire the authentication for this client.

An example follows of using the *authenticate-until* attribute to distinguish between clients that are authenticated and those that are not authenticated. After the authentication expires and the client requests another address, the DHCP server assigns the client an address from the unauthenticated scope range:

-
- Step 1** Create an authenticated and an unauthenticated client-class. Set the selection criteria for each as appropriate.
 - Step 2** Create the client and include the *authenticate-until* expiration time. Set the *client-class-name* and *unauthenticated-client-class-name* attributes as appropriate.
 - Step 3** Create the authenticated and unauthenticated scopes, define their address ranges, and tie them to their respective selection tags.
 - Step 4** Enable client-class processing for the server.
 - Step 5** If necessary, reload the DHCP server.
-

Setting Client Caching Parameters

The initial request from a client for an address from a DHCP server often goes through a DHCPDISCOVER-DHCPOFFER-DHCPREQUEST-DHCPACK cycle. This process requires that the server must consult the database twice per request for client data. If the client caching parameters are set, the DHCP server caches client data in memory so that it only needs to consult the database once. Client caching can provide a noticeable performance improvement in systems that store client information in LDAP. Client caching is enabled by default unless you unset the applicable attributes.

You can adjust the maximum cache count and time-to-live (TTL) parameters based on the expected rate of client requests. If you expect an onslaught of requests, you might want to increase the cache count, up to a limit based on your available memory. If you expect a longer request cycle, you might want to increase the TTL. The aim is to have the server consult the client cache once during the request cycle.

To set the limit on the number of entries that the server keeps in the client cache, use the *client-cache-count* attribute on the Edit DHCP Server page, or **dhcp set client-cache-count** in the CLI. By default, the maximum number to cache is 1000 clients. To disable the cache, set the attribute to 0.

The client cache is usually valid for only ten seconds, called the cache TTL. After the TTL expires, the server reads the client information from the database, if necessary. You can adjust the TTL using the *client-cache-ttl* attribute on the Edit DHCP Server page, or **dhcp set client-cache-ttl** in the CLI.

When the client cache count reaches the specified maximum, the server cannot cache any more clients until a client-entry TTL expires, after which it reads from the database and begins caching again.

The DHCP server, by default, caches client data while processing DISCOVER message only. If you want to cache client data during REQUEST (Renew or Rebind) message, you need to set *cache-client-for-requests* attribute to true. This attribute can be configured on the Edit DHCP Server page, or using **dhcp set cache-client-for-requests** in the CLI. This attribute should be set to true only if the duration between the two REQUEST (Renew or Rebind) messages are lesser than the cache TTL.

Subscriber Limitation Using Option 82

In many situations, service providers want to limit the number of IP addresses the DHCP server should give out to devices on customer premises. They want these devices to have “real” addresses that the DHCP server provides, but limit their number. One way is to use the client-class to register (or provision) each customer device so that the server issues IP addresses only to devices that are registered in the client-entry database. The major drawback to this approach is that it requires registering every customer device, which involves knowing its MAC address. Service providers often do not want to know about each device, but simply that there are not too many of them per customer.

Another approach is to limit customer devices on a per-subscriber basis on values in the *relay-agent-info* DHCP option (option 82, as described in RFC 3046) that the DHCP relay agent sends in a DHCPDISCOVER message. This option includes data about the port on a switch over which the customer device is attached. In a cable modem scenario, one of the option 82 suboptions usually contains the MAC address of the cable modem when the DHCP request comes from a device attached beyond the cable modem. In general, many devices that generate option 82 data place some values in its suboptions such that the value varies per subscriber on the same upstream device. In some cases, this value is unique across all possible subscribers (such as the MAC address of the cable modem). In others, it can be a port on a switch and thus unique across the other subscribers attached to that switch. However, it might not be unique across all subscribers on the switch.

Using this approach, the network administrator can configure limitations on subscriber use of the DHCP-allocated addresses without seriously impacting other DHCP server capabilities. In many environments, network administrators might want to use option 82 limitation for some class of devices and not others. A key aspect of this support is to allow network administrators to separate the devices for which they want to use option 82 limitation from those for which they do not.

Related Topics

- [General Approach to Subscriber Limitation, on page 297](#)
- [Typical Limitation Scenario, on page 298](#)
- [Calculating Client-Classes and Creating Keys, on page 298](#)
- [Client-Class Lookup Expression Processing, on page 298](#)
- [Limitation Processing, on page 299](#)
- [Expression Processing for Subscriber Limitation, on page 299](#)
- [Configuring Option 82 Limitation, on page 299](#)
- [Lease Renewal Processing for Option 82 Limitation, on page 300](#)
- [Administering Option 82 Limitation, on page 300](#)
- [Troubleshooting Option 82 Limitation, on page 301](#)
- [Expression Examples, on page 301](#)

General Approach to Subscriber Limitation

The current approach to client processing is to look up every client in the client-entry database. One of the goals of option 82 limitation is to remove the need explicitly to register (provision) every customer device in

the client-entry database (either in the CNRDB or LDAP). However, there is still a requirement that the specific number to which a subscriber is limited should be configurable and override the default number given to all unregistered subscribers.



Note Limitation processing is not currently available for DHCPv6 clients.

At a high level, you can configure subscriber limitation by creating an *expression* that the server evaluates for each incoming packet and returns the name of the client-class where you want the client to go (see [Using Expressions, on page 315](#) for details on the use of expressions.). Each client-class allows specification of a limitation identifier (ID), a key the server determines from the incoming packet and uses in later processing to actually limit the number of devices. The server considers all devices with the same limitation ID (the *limitation-id* property) to come from the same subscriber.

Typical Limitation Scenario

For example, an incoming packet might be evaluated such that:

1. If the *remote-id* suboption of option 82 matches the client hardware address (*chaddr*), the subscriber is a cable modem and should be in the **cm-client-class**.
2. If the first six bytes in the *dhcp-class-identifier* option value match the string **docsis**, then the subscriber is a DOCSIS modem and should be in the **docsis-cm-client-class**.
3. If the *user-class* option value matches the string **alternative-class**, then the subscriber should be in the **alternative-cm-client-class**.

Calculating Client-Classes and Creating Keys

You set the expression that determines the client-class for the *client-class-lookup-id* attribute of the DHCP server, or **dhcp set client-class-lookup-id=expression** in the CLI. Include simple expressions in the attribute definition or more complex ones in a file referenced in the attribute definition (see [Using Expressions, on page 315](#)).

Clients and client-classes also allow specifying a *limitation-id* value for the client or client-class. The server uses this identifier (ID) value to set the address limit on the number of devices with the identical ID on the same network or LAN segment. If a requesting client oversteps the limit of available addresses for that ID, the server assigns it to an *over-limit-client-class-name* (if set); otherwise, it drops the packet. The *limitation-id*, in effect, defines a subscriber.

Client-Class Lookup Expression Processing

The initial client-class lookup is to allow you to decide whether the client should participate in some sort of limitation. Configure an expression server-wide with the *client-class-lookup-id* attribute. The server executes this expression on every incoming packet with the goal of determining the client-class of the packet.

The expression should return a string that is the client-class name for the packet, or the distinguishing string `<none>` indicating that no client-class value was considered for the client request. Returning the `<none>` string is equivalent to not configuring a *client-class-lookup-id* value and that no client-class processing should occur. If the expression returns null or there is an error evaluating the *client-class-lookup-id*, the server drops the packet (with an accompanying log message).

Limitation Processing

The DHCP server limits the number of IP addresses allocated to DHCP clients with the same *limitation-id* value in the same network or LAN segment. In cases where the server finds that allocating another address to the client would go over the limit, it places the client packet in the *overflow-client-class* (if any is specified). This allows special handling for clients that are over the configured limit. Handling these clients in some self-provisioning way is one of the benefits of using limitation on the DHCP server instead of in the hardware (should it even be supported).

If there is no over-limit client-class, the server drops a packet where allocating an address for that packet would exceed the allowed *limitation-count* for that *limitation-id*. Note that the server enforces the limitation only in a single network or LAN segment. This is hardly a restriction, because network managers tend to see a single subscriber connecting only over one LAN segment at a time.

Configure the *limitation-count* with an identical *limitation-id* in a DHCP policy. The limitation code searches up the policy hierarchy for the *limitation-count* just as it does for any other policy item. This means that you can configure the *limitation-count* in a client-class embedded or named policy, a scope embedded or named policy, or the system *system_default_policy*.

When you configure a *limitation-id* on a client-class, you thereby signal to pursue limitation processing for the client-class. When you do not configure a *limitation-id*, you thereby signal not to pursue it. When executing the expression to determine the *limitation-id*, if the expression returns null, this signals that limitation processing should occur and to use the *limitation-id* saved in the lease state database.

Expression Processing for Subscriber Limitation

Expressions exist in several places in the limitation processing. Each expression evaluates to null or a string (typically to determine a client-class name when looking up a client-class), or to a series of bytes (a blob) when creating a *limitation-id*. You can use expressions in these places:

- Looking up a client-class
- Creating the key to limit clients of the same subscriber (the *limitation-id*)
- Creating the key to look up in the client-entry database (the *client-lookup-id*)

Configuring Option 82 Limitation

-
- Step 1** If you do not register clients explicitly, do not enable client-class as a DHCP server property when using option 82 data.
- Step 2** Determine if you want to limit some clients and not others. If you want to limit some clients:
- a) Find some method to distinguish these clients from the others, based on some values contained in the DHCP requests from each class of clients.
 - b) Determine the names of the client-classes into which you want to put the clients that are not limited, and the selection tag and scope or scopes you want to use for these unlimited clients.
- Step 3** Decide if you want to put clients that are over-limit into a different client-class or just drop their packets. If you want to put them in an over-limit client-class, determine the client-class name and the selection tag and scope or scopes into which you want to put the over-limit clients.
- Step 4** Determine the client-class into which you want to put clients that you intend to limit and the selection tags and scope or scopes you want to use for these clients.
- Step 5** Create all these selection tags, client-classes, and scopes.

- Step 6** Configure the *limitation-count* in a policy, probably the named policy associated with the client-class for the clients to limit.
- Step 7** Write the expression to separate the incoming clients into those to be limited and those not to be limited. Configure it on the DHCP server by setting the *client-class-lookup-id* attribute.
- Step 8** Write the expression to determine the limitation ID for the devices to limit, and configure it on the client-class for clients to limit by setting the *limitation-id*.

Lease Renewal Processing for Option 82 Limitation

Only packets that the DHCP client broadcasts arrive at the server with option 82 data attached. The BOOTP or DHCP relay agent adds the option 82 data in the first upstream router from the client device. A DHCPRENEW packet is unicast to the server and arrives without option 82 data. This can pose a problem when trying to configure the server for subscriber limitation.

There are generally two approaches to take when dealing with renewals:

- Place all packets that do not have option 82 data in a client-class with no associated selection tags. This is equivalent to a wildcard selection and means that any packet with no option 82 data is accepted.
- Place a DHCPRENEW in the same client-class as you would place a packet that has option 82 data, and have its *limitation-id* evaluate to null. This is a signal that when checking for limitation, the DHCP server should use a previously stored *limitation-id* instead of one from the packet.

Both approaches work. The second one can be more secure, but in practice, it is not much better than the first. This is because you have to use an IP address for the DHCP server to respond to a DHCPRENEW, and most clients would not do this unless the server loses some of its state. In this case, you would want it to give the client the address. In the case of a malicious client, it would still have to use the address to get the server to give the address to the client, thereby limiting the exposure for this case.

Administering Option 82 Limitation

Whenever a client is involved in limitation because of its inclusion in a client-class with a *limitation-id*, the limitation ID used appears in the DHCP log file whenever client data logging occurs as "... LID: *nnn* :*nnn* :*nnn*" The data is logged only for clients with active leases that are currently occupying one of the *limitation-count* counts.

You can determine all the clients using a *limitation-id* in a subnet. On the Manage DHCP Server page, click the **Run** icon in the Commands column to open the DHCP Server Commands page. Enter at least the IP address of the currently active lease in the IP Address field, then click the **Run** icon. You can also enter the *limitation-id* itself in the form *nn:nn:nn* or as a string ("*nnnn*"), in which case the IP address becomes the network in which to search. In the CLI, use **dhcp limitationList**:

```
nrcmd> dhcp limitationList ipaddr [limitation-id] show
```

If you specify both the *ipaddr* and *limitation-id*, the *ipaddr* value, the server uses it just like a *giaddr* to determine the subnet. You can use any IP address that could appear in any scope (primary or secondary) for the network to specify a subnet. If you specify only the *ipaddr*, it must be an address that the DHCP server serves, and the command returns all the clients and corresponding leases they use.

If a client is denied service due to a *limitation-count* overflow, a message such as this appears in the DHCP server log file:

```
Warning Server 0 05646 Could not add Client MAC: '1,6,01:02:03:04:0c:03' with
limitation-id: 01:02:03 using Lease: 10.0.0.23, already 3 Clients with that id.
No over-limit client class specified! Dropping packet!
```

You can determine which clients are extended beyond the *limitation-count*, thus causing a denial of service for any new client, by using **dhcp limitationList**. The *ipaddr* value in the command should be the “using Lease:” value, and the limitation-id should be the “limitation-id:” value, in the log file. Using the log file example, the command would be:

```
nrcmd> dhcp limitationList 10.0.0.23 01:02:03 show
```

Troubleshooting Option 82 Limitation

There are several ways that you can debug limitation support. First, you might want to turn on packet tracing by setting the DHCP server debug value to **VX=1** (or using **dhcp setDebug VX=1**). (The **VX=0** debug value disables packet tracing.) Then, you probably want to enable client-class debugging by adding *client-criteria-processing* and *client-detail* to your log settings.

There is also a server-wide expression trace level, *expression-trace-level*, that you can set to various levels. Setting it to 6 gives you a details trace of every expression evaluation. This can take a bit of space in the log, and slows down the server considerably as well, but is invaluable in the process of getting familiar with expression evaluation. See [Debugging Expressions, on page 349](#).

When things seem to be going strangely, or when submitting log files to report a problem, it is important to enable some additional tracing by setting the DHCP server debug value to **QR57=9** (or using **dhcp setDebug QR57=9**). (The **QR57=0** debug value disables this tracing). Note that the Q and R are both uppercase. The Q is client-class debugging and the R is response debugging (required to get the flow of control clear in the log). The 5 is expression processing and the 7 is *client-class-lookup* processing. This generates a page or so of output for each packet, which will help you understand what is going on inside the server.

Expression Examples

See [Expression Examples, on page 345](#).

Configuring Cisco Prime Network Registrar to Use LDAP

The Lightweight Directory Access Protocol (LDAP) provides directory services to integrate Cisco Prime Network Registrar client and lease information. By building on your existing standard schema for objects stored in LDAP directories, you can handle information about DHCP client entries. Thus, instead of maintaining client information in the DHCP server database, you can ask the Cisco Prime Network Registrar DHCP server to issue queries to one or more LDAP servers for data in response to DHCP client requests, or write lease data to an LDAP server.

Cisco Prime Network Registrar on Windows uses the Oracle Directory Server Enterprise Edition LDAP Software Development Kit (SDK) version 5.0. Linux uses their OpenLDAP client distribution. Windows use Win32 LDAP from Microsoft.

Related Topics

[About LDAP Directory Servers, on page 302](#)

[Adding and Editing LDAP Remote Servers, on page 302](#)

[Configuring DHCP Client Queries in LDAP, on page 303](#)

[Configuring DHCP LDAP Update and Create Services, on page 306](#)

[Troubleshooting LDAP, on page 312](#)

About LDAP Directory Servers

LDAP directory servers provide a way to name, manage, and access collections of attribute/value pairs. You can enter information into your LDAP server in any number of ways, because Cisco Prime Network Registrar does not depend on specific LDAP object classes or schema:

- You can store DHCP client information in unused attributes. For example, you could use the *givenname* attribute to hold the DHCP *client-class name* value.
- You can add new attributes to an object class without altering your LDAP schema if you disable LDAP schema checking. For example, you could add the *client-class-name* attribute to the organizational person object class.
- You can create a new object class and define the appropriate attributes. For example, you can create the DHCP client object class and define the client attributes that you want to use.

When you configure the DHCP server to read from LDAP, a query dictionary tells the server which LDAP attributes to query for. The server converts the resulting data into DHCP client data attributes.



Tip You can configure Cisco Prime Network Registrar to generate SNMP traps when an LDAP server stops responding or resumes responding to requests from the DHCP server.

Adding and Editing LDAP Remote Servers

You must add a remote LDAP server so that you can begin using the LDAP services.

Local Advanced Web UI

From the **Deploy** menu, choose **LDAP** under the **DHCP** submenu to open the List/Add LDAP Remote Servers page. Click the **Add LDAP** icon in the **LDAP** pane to open the Add DHCP LDAP Server dialog box. To edit the remote server, select the LDAP in the LDAP pane to open the Edit LDAP Remote Server page.

On this page, you must specify at least the name and fully qualified domain name or IP address (IPv4 or IPv6) of the LDAP server. The username and password are required for successful operation.



Note The Query Settings and the Create Settings will be used in the local for DHCP lease while the same settings will be used in the regional for BYOD client creation.

CLI Commands

Use **ldap name create domain-name**. For example:

```
nrcmd> ldap ldap-1 create ldap.example.com
```

You can also use **ldap server** IP address (IPv4 or IPv6). For example:

```
nrcmd> ldap ldap-1 create 192.0.2.1
nrcmd>ldap ldap-1 create 2001:DB8:1::1
```

Configuring DHCP Client Queries in LDAP

You can configure and unprovision DHCP client queries, and configure embedded policies, in an LDAP client entry.

Configuring DHCP-Server-to-LDAP Client Queries

To enable the DHCP server to query your LDAP server for client data, perform the following steps. Like local client entries, LDAP client entries are keyed by the client MAC address.



Note When connecting to an LDAP server, use the *distinguished name (dn)* of the user. It uniquely identifies an object in the LDAP schema, and is like a unique key in a database or a fully qualified path name for a file. For example, a dn for a person might be dn: cn=Beth Jones, ou=Marketing, o=Example Corporation. In this company, there may be many people named Beth and many people named Jones, but no one else named Beth Jones works in Marketing at Example Corporation.

Step 1 Supply a hostname for the LDAP server. On the Add LDAP Remote Server page, enter a value in the name field. In the local CLI, use this command:

```
nrcmd> ldap ldap-1 create ldap.example.com
```

Later, if you need to delete the server, use **ldap server delete**.

Step 2 Configure the connection credentials. Use the distinguished name (*dn*) for the user. Enter a value in the username field. In the CLI, use this command, for example:

```
nrcmd> ldap ldap=1 set username="cn=joe,o=Example Corp,c=US" password=access
```

Step 3 Set the search path (and, if necessary, the search scope). The path is a point in the directory from which to start searches. If the search scope is:

- SUBTREE, the server searches all the children of the search path.
- ONELEVEL, the server searches only the immediate children of the base object.
- BASE, the server searches only the base object itself.

This example sets the base of the search to be the organization Example Corp and the country US, with a subtree search scope. Enter a value in the search-path field. In the CLI, use this command, for example:

```
nrcmd> ldap ldap-1 set search-path="o=Example Corp,c=US" search-scope=SUBTREE
```

Step 4 Set the search filter to be the attribute for which DHCP will substitute the clients' MAC addresses. In this example, the attribute is the common name (*cn*). Enter a value in the search-filter field. In the CLI, use this command, for example:

```
nrcmd> ldap ldap-1 set search-filter=(cn=%s)
```

Step 5 Configure a query dictionary that contains all the LDAP-to-DHCP mappings. Use **ldap servername setEntry** to set these mappings.

1. Retrieve the DHCP surname from the *sn* LDAP attribute:

```
nrcmd> ldap ldap-1 setEntry query-dictionary sn=host-name
```

2. Retrieve the client-class name from the first *givenname* LDAP attribute:

```
nrcmd> ldap ldap-1 setEntry query-dictionary givenname=client-class-name
```

3. Retrieve the domain name from the *localityname* LDAP attribute:

```
nrcmd> ldap ldap-1 setEntry query-dictionary localityname=domain-name
```

4. If you need to unset any of the entries, use **ldap server unsetEntry attribute key**. You can also check any of the settings using **ldap server getEntry attribute key**.

Step 6 Enable queries for the LDAP server. This example enables queries for *myserver*. Set the *can-query* attribute to enabled. In the CLI, use this command:

```
nrcmd> ldap ldap-1 enable can-query
```

Step 7 Enable client-class processing for the DHCP server. On the Edit DHCP Server page, set the *client-class* attribute to enabled. In the CLI, use this command:

```
nrcmd> dhcp enable client-class
```

Step 8 Enable the DHCP server to use LDAP for client entry queries. On the Manage DHCP Server page, set the *client-class* attribute to enabled. In the CLI, use this command:

```
nrcmd> dhcp enable use-ldap-client-data
```

Step 9 If you have more than one LDAP server configured, you can also set them to operate in round-robin or failover mode:

- **round-robin**—The LDAP servers' preference values are ignored and all servers that are configured to handle client queries and accept lease state updates are treated equally.
- **failover**—The DHCP server uses the active LDAP server with the highest preference (lowest preference number). If the preferred server loses its connection or fails, the DHCP server uses the next LDAP server of lower preference (increasing preference number). If the preference values are the same (or not set), the DHCP reverts to round-robin mode with these servers.

Set the LDAP server mode by setting the *ldap-mode* on the Edit DHCP Server page. LDAP failover mode actually performs preferential load balancing. The DHCP server assesses the LDAP connection and error states and how fast the LDAP server responds. In an optimal state, the DHCP server uses the LDAP server with the highest assigned preference (lowest preference number). In a less-than-optimal state, the DHCP server uses the next LDAP server of lower preference (increasing preference number). If the preference values are the same (or unset), the DHCP server reverts to round-robin mode.

In the CLI, use **dhcp set ldap-mode** to set the mode, and **ldap server set preference** to set the server preferences; for example:

```
nrcmd> dhcp set ldap-mode=failover
nrcmd> ldap ldap-1 set preference=1
nrcmd> ldap ldap-2 set preference=2
```

Note also that, depending on how many threads you have open, as set by using the *connections* attribute (see [Recommended Values for LDAP, on page 313](#)) between the DHCP and LDAP servers, the DHCP server opens only as many threads as it can before the *query-timeout* expires. The LDAP server might be processing these threads, but it is not servicing the request, because the failover server has now taken over.

Step 10 Enable the DHCP server to use LDAP for client entry queries. On the Manage DHCP Server page, set the *client-class* attribute to enabled. In the CLI, use this command:

```
nrcmd> dhcp enable use-ldap-client-data
```

Step 11 Show or list the LDAP configuration. Go to the List/Add LDAP Remote Servers page. In the CLI, use:

```
nrcmd> ldap ldap-1
nrcmd> ldap list
nrcmd> ldap listnames
```

Step 12 Reload the DHCP server.

Unprovisioning Client Entries

You can unprovision LDAP client entries so that the client information remains in LDAP, but the DHCP server treats the client as if that information does not exist. The DHCP server then supplies the client with the default behavior. Configure the search filter set in Step 4 in [Configuring DHCP-Server-to-LDAP Client Queries, on page 303](#) of the preceding section so that the LDAP server does not return a client entry containing a specified attribute with a value.

If you want to unprovision the LDAP entry *givenname*, configure the search filter accordingly. For example:

```
nrcmd> ldap ldap-1 set search-filter=(&(cn=%s)!(givenname=unprovision))
```

Whenever the *givenname* attribute in the LDAP client entry is set to the “unprovision” string, the LDAP server does not return the client entry to the DHCP server. In other words, the DHCP server treats the client as if it has no LDAP client entry. This procedure has no measurable performance impact on either the DHCP or the LDAP server.

Configuring Embedded Policies in LDAP

Step 1 Configure an LDAP server for the DHCP server, naming it *myserver*, for example.

Step 2 Map the LDAP attribute that you want the DHCP server to interpret as the embedded policy to the internal embedded-policy property. This example maps the *businessCategory* LDAP attribute:

```
nrcmd> ldap myserversetEntry query-dictionary businessCategory=embedded-policy
```

Step 3 Add a string to the LDAP attribute that the DHCP server can interpret as an embedded policy. The most practical way to determine what this string should look like is to create a dummy client in the Cisco Prime Network Registrar database and extract data from the client embedded policy setup. Note that this dummy client will never be used, because you are using LDAP, and you can subsequently delete it. Have the embedded policy include the option data types that you need.

1. For example, create an embedded client policy for dummy client 1,6,00:d0:ba:d3:bd:3b. Add some reply options and a multivalue option (routers) with an IP address data type:

```
nrcmd> client 1,6,00:d0:ba:d3:bd:3b create
nrcmd> client-policy 1,6,00:d0:ba:d3:bd:3b set v4-reply-options=routers
nrcmd> client-policy 1,6,00:d0:ba:d3:bd:3b setOption routers 1.2.3.4,5.6.7.8
nrcmd> save
```

2. Get the client embedded policy data so that you can display the values:

```
nrcmd> client 1,6,00:d0:ba:d3:bd:3b get embedded-policy
100 Ok
embedded-policy="( (ClassName Policy) (name client-policy:00:d0:ba:d3:bd:3b) (option-list [ ( (ClassName
```

Configuring Embedded Policies in LDAP (with multiple option definitions)

```
Option) (number 3) (option-definition-set-name dhcp-config) (value
01:02:03:04:05:06:07:08))) (v4-reply-options [routers ])"
```

3. Copy what is between the quotes in the client output in the previous substep and paste it in for the definition of the businessCategory LDAP attribute:

```
businessCategory:((ClassName Policy) (name client-policy:00:d0:ba:d3:bd:3b) (option-list [((ClassName
Option) (number 3) (option-definition-set-name dhcp-config) (value
01:02:03:04:05:06:07:08))) (v4-reply-options [routers ])
```

4. Use the syntax as a model for each new embedded policy entry in LDAP. To see how other option data types appear in the LDAP string, add these options to the client or create further dummy clients with them. Once you extract the data, you can delete the dummy client:

```
nrcmd> client 1,6,00:d0:ba:d3:bd:3b delete
nrcmd> save
```

Configuring Embedded Policies in LDAP (with multiple option definitions)

Here is another example with multiple option definitions:

- Step 1** Create a dummy client 1,6,00:d0:ba:d3:bd:3b and an embedded policy attached to that client, with the following options and values:

```
3 routers 10.1.1.1,10.2.1.1
66 tftp-server tftp-server.com
67 bootfile device-boot-file.txt
```

- Step 2** Save the changes to the embedded policy, save the client, then extract the following output string into an LDAP client configuration:

```
nrcmd> client 1,6,00:d0:ba:d3:bd:3b get embedded-policy
100 Ok
embedded-policy="((ClassName Policy) (name client-policy:00:d0:ba:d3:bd:3b) (option-list [((ClassName
Option) (number 3) (option-definition-set-name dhcp-config) (value 0a:01:01:01:0a:02:01:01)) ((ClassName
Option) (number 66) (option-definition-set-name dhcp-config) (value
74:66:74:70:2d:73:65:72:76:65:72:2e:63:6f:6d)) ((ClassName Option) (number 67) (option-definition-set-name
dhcp-config) (value 64:65:76:69:63:65:2d:62:6f:6f:74:2d:66:69:6c:65:2e:74:78:74)) ])"
```

Configuring DHCP LDAP Update and Create Services

You can configure the Cisco Prime Network Registrar DHCP server to write lease and client data to an LDAP server. The DHCP server can use the client data when responding to DHCP client requests, through the use of the *query* configuration. You can configure the DHCP LDAP service to copy lease state data to attributes on client objects in the LDAP server. The DHCP server converts the lease state data to string form, and uses an update dictionary to map the DHCP data values to the LDAP attributes.

Each time the lease state changes, the DHCP server writes the change to the LDAP server that you configured to store the data. The lease data that the DHCP server writes to LDAP is “write-only” in that it is a copy of the authoritative data in the lease state database.

Related Topics

- [Lease State Attributes, on page 307](#)
- [Configuring DHCP to Write Lease States to LDAP, on page 308](#)
- [Using LDAP Updates, on page 309](#)
- [Configuring LDAP State Updates, on page 309](#)
- [Configuring LDAP Entry Creation, on page 311](#)

Lease State Attributes

You can store any of these attributes about the lease state information in your LDAP server:

- **address**—IP address of this lease.
- **client-dns-name**—Name the DHCP server attempted to enter into the DNS server for this client.
- **client-domain-name**—Domain into which to put the client name.
- **client-flags**—A variety of flags relating to the client.
- **client-host-name**—DNS name that the client requested the DHCP server to place in the DNS server.
- **client-id**—Client-id specified by the client, or one synthesized by the DHCP server for this client.
- **client-mac-addr**—MAC address that the client presented to the DHCP server.



Note Although the MAC addresses in LDAP have to be formatted exactly the way they are formatted by Cisco Prime Network Registrar when it creates local client-entries, they are separate instances and thus unique to lease data.

- **expiration**—The time at which the lease expires.
- **flags**—Flags for the lease (reserved or deactivated).
- **lease-renewal-time**—The earliest time in which the client is expected to issue a lease renewal. You can have Cisco Prime Network Registrar save this as part of the lease state by using **dhcp enable save-lease-renewal-time** (it is not saved by default).
- **start-time-of-state**—The time at which the state last changed to its current value.
- **state**—The lease state can be:
 - Available (1)
 - Deferred (2)
 - Leased (3)
 - Expired (4)
 - Unavailable (5)
 - Released (6)
 - Other_available (7)
 - Disconnected (8)
 - Deleted (9)
- **vendor-class-identifier**—The name of the vendor, used by clients and servers to exchange vendor-specific information.

Not every lease has all these attributes. The *client-mac-addr* and *client-id* lease state attribute are not present if a client releases its lease or is forced available through Cisco Prime Network Registrar. In addition, the *lease-renewal-time* attribute may not be present if the *save-lease-renewal-time* property is disabled through DHCP. Similarly, the *vendor-class-identifier* property may not be present if the *save-vendor-class-id* property is disabled through DHCP, using the CLI.

Configuring DHCP to Write Lease States to LDAP

To have DHCP write lease state updates to LDAP:

-
- Step 1** Choose the LDAP lease state update scheme.
 - Step 2** Add entries to the directory or modify existing entries to store the lease state information. You may need to extend entries through the addition of attributes or custom object classes.
 - Step 3** Configure Cisco Prime Network Registrar to perform the updates.

Given the flexibility of directories, there are many different ways in which you could choose to store a copy of lease state attributes in a directory. For example, you could choose to store the lease state data as part of an existing entry, or you could store the lease state data independently.

Storing Lease State Data as Part of Existing Entries

You can store lease state data as part of an existing entry. It is even possible to store the client entry, lease state, and employee data in the same entry. As part of the setup for this method, you must decide how you want to store the lease data attributes. You can store data attributes using these methods:

- Map attributes from the entry
- Add attributes to the entry
- Extend the entry by creating a new object class

The advantage is that lease data is stored directly with other client information. The disadvantage is that there are scenarios, albeit unlikely, related to client-class and reservations that could result in stale data being in the directory for a short period of time when the server moves a client off a lease.



Note If the lease whose state is being updated does not have a client, it will not have an associated MAC address. This situation occurs when a client gets a lease, and then is moved off that lease by client-class processing. It can also occur when a client has a pre-existing lease and a reservation for a different lease in the same LAN segment. When the reserved lease is available, the server moves the client off its existing lease and onto the reservation. Both of these transfers result in an LDAP update for the old lease without a client MAC address. This is generally not a problem, because the update for the new lease (which has an associated MAC address) should come through.

Also, this method requires two LDAP interactions to write the lease information. When updating lease state information, the DHCP LDAP service contacts the directory twice because when updating an entry it is not enough just to know how to find the entry. You must specifically know the *dn* of the entry.

The DHCP LDAP service first finds the appropriate entry in the directory by using one of the lease state attributes that you chose (preferably the MAC address) as the search criteria. This is necessary because none

of the lease state attributes is part of the *dn* of the entry. When the DHCP LDAP service locates the entry, the *dn* is returned. The DHCP LDAP service then updates that same entry with the appropriate information. For an example how to use this method, see [Configuring LDAP State Updates, on page 309](#).

Storing Lease State Data Independently

You can store lease state data by IP address in its own entries. This method results in a copy of the server lease database in a directory, and is the most straightforward way to configure the database. As part of the setup for this method, create new entries for each IP address that the server can serve. The advantage to this method is that there are no scenarios in which the lease state data in the directory will be stale. The disadvantage is that lease data is not stored directly with other associated client information.

To update the lease state information, the DHCP LDAP service contacts the directory service once. When performing the update, the service uses the IP address to construct the *dn*.

Using LDAP Updates

There are two ways you can use the LDAP update feature:

- Keep track of clients that use LDAP client entry information and to associate some of the attributes of that LDAP host with lease state attributes.
- Create and update objects that can be located by their IP address. When Cisco Prime Network Registrar creates these objects, it can make a level of LDAP objects that matches (or is) the DHCP server lease state.

When using Cisco Prime Network Registrar, you should be aware that:

- The DHCP server only reads from a single object and writes to a single object. You can use separate objects to hold the client entry data read and the lease state data written, but Cisco Prime Network Registrar cannot read some attributes from one object and some from another.
- The performance of LDAP queries, like all database access, depends on indexed attributes. If you did not index the attributes that you configure to use in query filters, you will experience poor performance.
- LDAP attributes must either come preconfigured in the LDAP schema at server installation or be created by some other means outside Cisco Prime Network Registrar.

Configuring LDAP State Updates

There are two options available for performing a lease state update to an LDAP server:

- *update-search-path*—The DHCP server first queries to locate the *dn* for an update.
- *dn-format*—The server is provided with the *dn* for an update. In other words, the DHCP performs a direct update without having to query before an update.

Option 1: Using the update-search-path Option

The following example illustrates the first option, *update-search-path*. It shows what to do when the distinguished name (*dn*) of an LDAP object cannot be constructed from data that is available in the lease state. The DHCP server creates an LDAP query based on the *update-search-xxx* information, locates the LDAP object, and uses its *dn* to issue an LDAP update.

The example shown in the table below assumes that you are using the standard LDAP organizational person object class attributes to hold lease update data.

Table 40: LDAP-to-DHCP Mapping Example

Attribute	DHCP Lease Entry Mapping
<i>uid</i>	address (IP address)
<i>carlicense</i>	state (lease state)

-
- Step 1** Tell DHCP about the LDAP server by supplying the server hostname in the LDAP configuration.
- Step 2** Configure the credentials to use when connecting to the LDAP server. This CLI example sets the administrator to joe and his password to access. Use the distinguished name (*dn*) for the user:
- ```
nrcmd> ldap myserver set username="cn=joe,o=Example Corporation,c=US" password=access
```
- Step 3** Configure the *update-search-path* attribute, which is the starting point in the directory for the objects that the DHCP server will update. You can also set the update search scope. This CLI example sets the search path to begin at the organizational unit (ou) IT, the organization Example Corporation, and country US. The update search scope is set to SUBTREE:
- ```
nrcmd> ldap myserver set update-search-path="ou=IT,o=Example Corp,c=US"
update-search-scope=SUBTREE
```
- Step 4** Set the ID of the attribute you want to use to search for the LDAP object that will be updated. This CLI example sets the search attribute to be the client MAC address:
- ```
nrcmd> ldap myserver set update-search-attribute=client-mac-addr
```
- Step 5** Configure a filter expression into which the *update-search-attribute* attribute should be formatted. This expression must contain a "%s," which indicates where the search attribute data should be substituted. Here is a CLI example:
- ```
nrcmd> ldap myserver set update-search-filter=(cn=%s)
```
- Step 6** Configure the *update-dictionary* attribute, which allows you to identify the LDAP attributes that you want set with the values of the corresponding lease state attributes. This example specifies that the LDAP UID should be updated to contain the IP address, and that the *carlicense* attribute should be updated to contain the DHCP lease state information. Using the CLI:
- ```
nrcmd> ldap myserver setEntry update-dictionary uid=address carlicense=state
```
- Step 7** Enable updates for the new LDAP server. Here is a CLI example:
- ```
nrcmd> ldap myserver enable can-update
```
- Step 8** Reload the DHCP server.
-

Option 2: Using the dn-format Option

This example illustrates using the second option, *dn-format*:

- Step 1** Tell DHCP about the LDAP server by supplying the server hostname in the LDAP configuration.
- Step 2** Configure the credentials to use when connecting to the LDAP server. This CLI example sets the administrator to joe and his password to access. Use the *dn* for the user:

```
nrcmd> ldap myserver_option2 set username="cn=joe,o=Example Corporation,c=US"
password=access
```

Step 3 Use the *dn-format* string to specify where in the LDAP server database hierarchy you want to begin searching for the update. Here is a CLI example:

```
nrcmd> ldap myserver_option2 set dn-format="cn=\"%s\",ou=IT,o=Example Corp,c=US"
```

Step 4 Set the *dn-attribute* attribute to which you want the *dn-format* string to refer. This CLI example sets the *dn-attribute* to be the client MAC address:

```
nrcmd> ldap myserver_option2 set dn-attribute=client-mac-addr
```

Step 5 Specify the entries to be updated. Using the CLI:

```
nrcmd> ldap myserver_option2 setEntry update-dictionary uid=address carlicense=state
```

Step 6 Enable the *can-update* attribute. Here is a CLI example:

```
nrcmd> ldap myserver_option2 enable can-update
```

Step 7 Reload the DHCP server.

Configuring LDAP Entry Creation

This section explains how to create LDAP entries. LDAP entry creation provides the ability to locate entries and update them with current lease information. Entries are created only if a state update operation fails because it cannot locate an entry.

After performing the steps in the previous example, follow these steps in the CLI:

Step 1 Set the *dn-attribute* property for the LDAP server for the lease object attribute, such as the *client-mac-addr* field, and set the *dn-format* string. Here is a CLI example:

```
nrcmd> ldap myserver set dn-attribute=client-mac-addr
dn-format="cn=\"%s\",ou=IT,o=Example Corp,c=US"
```

This step is required only if you configure the lease state updates using the *update-search-path* option. (See [Option 1: Using the update-search-path Option, on page 309](#)). Skip this step if you configure lease state updates using the *dn-format* string. (See [Option 2: Using the dn-format Option, on page 310](#).)

Step 2 Specify the *dn* of the entry to be created when combined with the existing *dn-attribute* property. Here is a CLI example:

```
nrcmd> ldap myserver set dn-create-format="cn=\"%s\",ou=IT,o=Example Corp,c=US"
```

The Cisco Prime Network Registrar *client-mac-addr* field uses the form **1,6:xx:xx:xx:xx:xx:xx**. Since the comma character is a special separator in LDAP, you must use the `\` characters to quote the *dn*.

Step 3 Using the *create-dictionary* property, establish mappings between LDAP attributes and lease state attributes by entering a series of name=value pairs. The LDAP attributes indicate the entry attributes set to the value of their corresponding lease state attributes. In the CLI:

```
nrcmd> ldap myserver setEntry create-dictionary sn=client-host-name
nrcmd> ldap myserver setEntry create-dictionary givenname=client-class-name
nrcmd> ldap myserver setEntry create-dictionary localityname=client-domain-name
```

Step 4 Using the *create-object-classes* property, specify the object classes to be used when creating the entry. Here is a CLI example:

```
nrcmd> ldap myserver set create-object-classes=
"top,person,organizationalPerson,inetorgperson"
```

Step 5 Enable entry creation for the LDAP server myserver. Here is a CLI example:

```
nrcmd> ldap myserver enable can-create
```

Note Enable the *can-update* attribute before you enable the *can-create* attribute. For an example, see [Configuring LDAP State Updates, on page 309](#).

Step 6 Reload the DHCP server.

Step 7 To see if creation, queries, and updates were successful, view the LDAP log settings.

Troubleshooting LDAP

The following sections include some advice on fine-tuning and detecting failures of the LDAP server.

Related Topics

[LDAP Connection Optimization, on page 312](#)

[Recommended Values for LDAP, on page 313](#)

LDAP Connection Optimization

You can optimize LDAP connections by using separately tunable read and write objects. This CLI example tunes write (create and update) operations, which require longer server processing:

```
nrcmd> ldap LDAP-Write create csrc-ldap password=changeme port=389 preference=1

nrcmd> ldap LDAP-Write setEntry query-dictionary csrcclientclasas=client-class-name

nrcmd> ldap LDAP-Write set
search-filter=((&(macaddress=%s) (|(csrcclassname=Computer) (csrcclassname=Modem)))

nrcmd> ldap LDAP-Write set search-path=csrcprogramname=csrc,o=NetscapeRoot

nrcmd> ldap LDAP-Write set
username=uid=admin,ou=Administrators,ou=TopologyManagement,o=NetscapeRoot

nrcmd> ldap LDAP-Write disable can-query

nrcmd> ldap LDAP-Write enable can-create

nrcmd> ldap LDAP-Write enable can-update

nrcmd> ldap LDAP-Write enable limit-requests

nrcmd> ldap LDAP-Write set connections=2 max-requests=8 timeout=10s
```

This CLI example tunes read (query) operations, which require shorter server processing:

```
nrcmd> ldap LDAP-Read create csrc-ldap password=changeme port=389 preference=1
```

```

nrcmd> ldap LDAP-Read setEntry query-dictionary csrcclientclasas=client-class-name

nrcmd> ldap LDAP-Read set
search-filter=((&(macaddress=%s)(|(csrcclassname=Computer)(csrcclassname=Modem)))

nrcmd> ldap LDAP-Read set search-path=csrcprogramname=csrc,o=NetscapeRoot

nrcmd> ldap LDAP-Read set
username=uid=admin,ou=Administrators,ou=TopologyManagement,o=NetscapeRoot

nrcmd> ldap LDAP-Read enable can-query

nrcmd> ldap LDAP-Read disable can-create

nrcmd> ldap LDAP-Read disable can-update

nrcmd> ldap LDAP-Read enable limit-requests

nrcmd> ldap LDAP-Read set connections=3 max-requests=12 timeout=4s

```

Recommended Values for LDAP

The table below shows recommended values for some key LDAP attributes.

Table 41: Recommended Values for LDAP Attributes

Attribute and Value	Description
<i>connections</i> =5 to 25	Number of connections that the server should make to an LDAP server. This is primarily a performance tuning parameter. The default is one connection. In some cases, more than one connection can improve overall throughput. The amount depends on the load on the LDAP server. With many applications using LDAP, five connections would be appropriate; with just Cisco Prime Network Registrar using LDAP, 25 would be appropriate.
<i>threadwaittime</i> =2	Interval (in milliseconds) at which each LDAP client connection polls for results, if it has outstanding queries or updates.
<i>query-timeout</i> =3	Cisco Prime Network Registrar DHCP servers fail over at the <i>query-timeout</i> interval if <i>failover</i> and <i>can-query</i> are set. The default setting is 3 seconds and is recommended (in that it is less than the default 4-second <i>drop-old-packets</i> value for the DHCP server, after which time the connection is considered inactive and the LDAP server as “unhealthy”).
<i>timeout</i> =10	Number of seconds an LDAP request remains in a connection queue before being declared stale and timing out. Any response the DHCP client receives after the client timeout period is stale. The default is 10 seconds, which is recommended. Cisco Prime Network Registrar DHCP servers fail over at the <i>timeout</i> interval if <i>failover</i> and <i>can-update</i> or <i>can-create</i> are enabled.

Recommended Values for LDAP



CHAPTER 11

Using Expressions

Cisco Prime Network Registrar provides enhanced client-class support. You can now place a request into a client-class based on the contents of the request, without having to register the client in the client database. Also, you can now place requests in a client-class based on the number of the active leases of a subscriber, allowing limitations on the level of service offered to various subscribers. This is possible through the special DHCP options processing using expressions.

You can set the limitation on subscriber addresses based on values in the DHCP *relay-agent-info* option (option 82, as described in RFC 3046). These values do not need to reveal any sensitive addresses. You can create values that relate an individual to a subscriber by creating an expression that evaluates the incoming DHCPDISCOVER request packets against option 82 suboptions (*remote-id* or *circuit-id*) or other DHCP options. The expression is a series of *if* statements that return different values depending on what is evaluated in the packet. This, in effect, calculates the client-class in which the subscriber belongs, and limits address assignment to the scope of that client-class.



Note Expressions are not the same as DHCP extensions. Expressions are commonly used to create client identities or look up clients. Extensions (see [Using Extension Points, on page 351](#)) are used to modify request or response packets. The expressions described here are also not the same as regex.

- [Using Expressions, on page 315](#)
- [Entering Expressions, on page 316](#)
- [Creating Expressions, on page 317](#)
- [Expression Functions, on page 322](#)
- [Expression Examples, on page 345](#)
- [Debugging Expressions, on page 349](#)

Using Expressions

Expression processing is used in several places:

- **Calculating a client-class**—*client-class-lookup-id*. This expression determines the client-class based on the contents of the incoming packet.
- **Creating the key to look up in the client-entry database**—*client-lookup-id*. This accesses the client-entry database with the key resulting from the expression evaluation.

- **Creating the ID to use to limit clients of the same subscriber**—*limitation-id*. This is the ID to use to check if any other clients are associated with this subscriber. This is supported only for DHCPv4 (not DHCPv6).

This kind of processing results in this scenario:

1. The DHCP server tries to get a client-class based on a *client-class-lookup-id* expression. If it cannot calculate the client-class, it uses the usual MAC address method to look up the client.
2. If the server can calculate the client-class, it determines if it needs to do a client-entry lookup, based on evaluating a *client-lookup-id* expression that returns a *client-lookup-id*. If it has such an ID, it uses it to look up the client. If it does not have such an ID, it uses the calculated client-class value to assign addresses.
3. If the server uses the *client-lookup-id* and finds a client-entry, it uses the data for the client. If it cannot find a client-entry, it uses the calculated or default client-class data.

For DHCPv4, you can also set the upper limit on assigned addresses to clients on a network or LAN segment having an identical *limitation-id* value on the policy level. Set this upper limit as a positive integer using the *limitation-count* attribute for the policy. Similar processing is possible for DHCPv6 using the *v6-client-class-lookup-id* and *v6-client-lookup-id* expressions.

The values to set for limiting IP addresses to subscribers are:

- For a policy, set the *limitation-count* attribute to a positive integer.
- For a client-class, set the *limitation-id* and *client-lookup-id* attributes to an expression, and set the *over-limit-client-class-name* attribute to a client-class.
- For a client, set the *over-limit-client-class-name* attribute to a client-class.

The expressions to use are described in [Creating Expressions, on page 317](#).

Entering Expressions

You can include simple expressions as such in the attribute definition, or include more complex ones in an expression file and reference the file in the attribute definition. Either way, the maximum allowable characters is 16 KB.

Most expressions that are configured with the CLI are stored in a text file which is then associated with the desired configuration attribute. The default path of this file is the current working directory. You can configure a simple expression directly in the CLI without storing it in a text file. Simple expressions must adhere to these rules when you enter them in the CLI:

- They must be limited to a single command line.
- The entire expression must be enclosed in double quotes (" ").
- Embedded double quotes must be escaped with a backslash (\).

Here is an example of a simple expression to set the *client-class-lookup-id*:

```
\ "limit\"
```

If you want to use a slightly more extensive example to set the client-class *limitation-id*:

```
(request option 82 "circuit-id")
```

Entering this expression directly in the CLI is not possible because of limitations in the CLI's command parsing. You must enter more complex expressions by placing them in a text file and then reference that file

in the attribute definition prefixed by the "@" symbol (@). For example, if that expression is placed in the `cclookup.txt` file, the CLI command is:

```
nrcmd> dhcp set client-class-lookup-id=@cclookup.txt
```

The syntax of the expression in the file does not have the extra requirements (as to spacing and escaping of characters) of the simple expression. It can also include comment lines, prefixed by the pound sign (#), double-slash (//), or a semicolon (;), and terminated at the end of line. For example,

```
// Expression to set client-class based on remote-id
(if (equal (request option "relay-agent-info" "remote-id") (request chaddr))
    "no-limit"
    "limit")

// Expression to calculate client-class based on remote-id
(try
  (if (equal (request option "relay-agent-info" "remote-id") (request chaddr))
      "cm-client-class"
      "cpe-client-class")
  "<none>")
```

The IPv6 version of the previous example (using option numbers) is:

```
// Expression to calculate client-class based on DOCSIS 3.0 cm-mac-address
(try
  (if (equal (request option 17 enterprise-id 4491 36)
            (or (request relay option 17 enterprise-id 4491 1026) "none"))
      "v6-cm-client-class"
      "v6-cpe-client-class")
  "<none>")
```

You can also write the previous expression by substituting option names in place of numbers:

```
// Expression to calculate client-class based on DOCSIS 3.0 cm-mac-address
(try
  (if
    (equal
      (request option
        "vendor-opts" enterprise-id "dhcp6-cablelabs-config" "device-id")
      (or
        (request relay option
          "vendor-opts" enterprise-id "dhcp6-cablelabs-config" "cm-mac-address")
        "none"))
    "v6-cm-client-class"
    "v6-cpe-client-class")
  "<none>")
```

The `or` function in the examples ensures that if the packet was not relayed or if the relay agent did not add the option, then the server assumes the client to be a CPE and not a cable modem (CM).

Creating Expressions

Using DHCP expressions, you can retrieve, process, and make decisions based on data in incoming DHCP packets. You can use them for determining the client-class of an incoming packet, and create the equivalence key for option 82 limitation support. They provide a way to get information out of a packet and individual

options, a variety of conditional functions to allow decisions based on information in the packet, and data synthesis capabilities where you can create a client-class name or key.

The expression to include in an expression file that would describe the example in [Typical Limitation Scenario, on page 298](#) would be:

```
// Begins the try function
(try
  (or
    (if (equal
        (request option "relay-agent-info" "remote-id")
        (request chaddr)
        "cm-client-class")
      (if (equal
          (substring (request option "dhcp-class-identifier") 0 6)
          "docsis")
          "docsis-cm-client-class")
        (if (equal
            (request option "user-class")
            "alternative-class")
            "alternative-cm-client-class"))
        "<none>")

    // Ends the try function
```

The expression uses the **or** function and evaluates three **if** functions. In a simpler form, you can calculate a client-class and include this expression in the `cclookup.txt` file.

```
// Expression to calculate client-class based on remote-id
(try
  (if (equal (request option "relay-agent-info" "remote-id") (request chaddr))
      "cm-client-class"
      "cpe-client-class")
  "<none>")
```

Refer to this file to use the expression to set the client-class lookup ID for the server:

```
nrcmd> dhcp set client-class-lookup-id=@cclookup.txt
```

You can generate a limitation key by trying to get the *remote-id* suboption from option 82, and if unable, to use a standard MAC blob key. Include an expression in a file and set the limitation ID to it in the `cclimit.txt` file:

```
// Expression to use remote-id or standard MAC
(try (request option "relay-agent-info" "remote-id") 00:d0:ba:d3:bd:3b)
```

Expression Syntax

Expressions consist solely of functions and literals. Its syntax is similar to that of Lisp. It follows many of the same rules and uses Lisp function names where possible. The basic syntax is:

```
(function argument-0 ... argument-n)
```

A more useful example is:

```
(try
  (if (equal (request option "relay-agent-info" "remote-id") (request chaddr))
      "cm-client-class"
      "cpe-client-class")
```

```
"<none>")
```

This example compares the *remote-id* suboption of the *relay-agent-info* option (option 82) with the MAC address in the packet, and if they are the same, returns “cm-client-class,” and if they are different, returns “cpe-client-class.” (If the expression cannot evaluate the data, the **try** function returns a “<none>” value—see [Expressions Can Fail, on page 320](#).) The intent is to determine if the device is a cable modem (where, presumably, the *remote-id* equals the MAC address) and, if so, put it into a separate client-class than the customer premise equipment or PC. Note that both functions and literals are expressions. The previous example shows a function as an expression. For literals, see [Literals in Expressions, on page 319](#).

Expression Datatypes

The datatypes that expressions support are:

- **Blob**—Counted series of bytes, with a recommended maximum length of 1 KB.
- **String**—Counted series of NVT ASCII characters, not terminated by a zero byte, with a recommended maximum length of 1 KB.
- **Signed integer**—32-bit signed integer.
- **Unsigned integer**—32-bit unsigned integer.

Note that there is no IP address datatype; an IPv4 address is a 4-byte blob, while an IPv6 address is a 16 byte blob. All numbers are in network byte order. See [Datatype Conversions, on page 320](#).

Literals in Expressions

A variety of literals are included in the expression capability:

- **Signed integers**—Normal numbers that must fit in 32 bits.
- **Unsigned integers**—Normal unsigned numbers that must fit in 32 bits.
- **Blobs**—Hex bytes separated by colons. For example, 01:02:03:04:05:06 is a 6-byte blob with the bytes 1 through 6 in it. This is distinct from “01:02:03:04:05:06” (a 17-byte string). The string is related to the blob by being the text representation of the blob. For example, the expression (**to-blob "01:02:03"**) returns the blob 01:02:03. Note that you cannot create a literal representation of a one-byte blob, as 01 will turn into an integer. To get a one-byte blob containing a 1, you can use (**byte 1**) as that will return a blob of 01. Alternatively, you can use the expression (**substring (to-blob 1) 3 1**). The 3 indicates the offset to extract the fourth byte of the 4-byte integer (00:00:00:01), with the 1 being the number of bytes extracted, with a result of “01.”
- **String**—Characters enclosed in double quotes. For example, “example.com” is a string, as is “01:02:03:04:05:06.” To place a quote in a literal string, escape it with a backslash (\), for example:

```
"this has one \"quote"
```

Integer literals (signed and unsigned) are assumed to be in base 10. If they start with a 0, they are considered octal; if they start with 0x, they are considered hexadecimal. Some examples of literals:

- “hello world” is a string literal (and a perfectly valid expression).
- 1 is an unsigned integer literal (also a perfectly valid expression). It contains 4 bytes, the first three of which are zero, and the last of which contains a 1 in the least significant bit.
- 01:02:03 is a blob literal containing three bytes, 01, 02, and 03.
- -10 is a signed integer literal containing four bytes with the twos-complement representation of decimal -10.

Expressions Return Typed Values

With few exceptions, the point of an expression is to return a value. The expression configured to determine a client-class is configured in the DHCP server property *client-class-lookup-id*. When this expression is evaluated, the DHCP server expects it to return a string containing the name of a client-class, or the string "**<none>**".

Every function returns a value. The datatype of the value may depend on the datatype of the argument or arguments. Some expressions only accept arguments of a certain datatype; for example:

```
(+ argument0 argument1)
```

In most cases, a function that requires a certain datatype for a particular argument tries to convert the argument that it gets to the proper datatype. For example, `(+ "1" 2)` returns 3, because it successfully converts the string literal "1" into a numeric 1. However, `(+ "one" 2)` causes an error, because "one" does not convert successfully into a number. In general, the expression evaluator tries to do the right thing as much as possible when making datatype conversion decisions.

Expressions Can Fail

While some of the functions that make up an expression operate correctly on any datatype or value, many do not. In the previous section, the `+` function would not convert the string literal "one" into a valid number, so the evaluation of that function failed. When a function fails to evaluate, its calling function also fails, and so on, until the entire expression fails. A failed expression evaluation has different consequences depending on the expression involved. In some cases, it can cause the packet to be dropped, while in others it only generates a warning message.

You can prevent the evaluation from failing by using the `(try expression failure-expression)` function. The **try** function evaluates the expression and, if successful, the value of the function is the value of the *expression*. If the evaluation fails (for whatever reason), the value of the function is the value of the *failure-expression*. The only situation where a **try** function itself fails is if the *failure-expression* evaluation fails. Thus, you should be careful what expression you define as a *failure-expression*. A string literal is a safe bet. Thus, protecting the evaluation of the *client-class-lookup-id* with a **try** function is a good idea. The previously cited example shows how this can work:

```
(try
  (if (equal (request option "relay-agent-info" "remote-id")
            (request chaddr)
            "cm-client-class"
            "cpe-client-class")
      "<none>")
  )
```

If evaluating the `if` function fails in this case, the value of the *client-class-lookup-id* expression is "**<none>**". It could have been a client-class name instead, of course.

Datatype Conversions

When a function needs an argument of a particular datatype, it tries to convert a value into that datatype. Sometimes this can fail, often causing the entire function to fail. Datatype conversion is also performed by the **to-string**, **to-blob**, **to-sint**, and **to-uint** functions. Whenever a function needs an argument in a specific datatype, it calls the internal version of these externally available functions.

There are also **as-string**, **as-blob**, **as-sint**, and **as-uint** conversion functions, where the data in a value are simply relabeled as the desired datatype, although some checking does go on. The conversion matrix for both function sets appears in the table below.

Note the distinction between **to-string** and **as-string**. For example, let us say that you have data in blob format. You could have this data because of the result of a function evaluation (**request get option**) which retrieves data from a request packet, or as the result of processing blob data with substring. If this data, despite being of blob type, actually represent ASCII string data, you might want to use it as a string. You have two choices of conversions—**as-string** and **to-string**. Which one to choose? If the data consists of ASCII bytes and you want to simply recognize that and essentially reset the type of the data as string, you want to use the **as-string** function. This means that, you are going to use the bytes of the blob "as" a string. The blob 00:01 cannot be converted into a string and it will throw an error if you try. The blob 68:65:6c:6c:6f will successfully convert to a string with **as-string** and yield "hello". On the other hand, if you have a series of bytes that may or may not be ASCII data and you want to represent the data in the blob in a string format, you should use **to-string**. For example, **to-string** will turn a two byte blob consisting of first a 0 then a 1 into the string "00:01".

Table 42: Datatype Conversion Matrix

Function	String	Blob	Signed Integer	Unsigned Integer
as-blob	Cannot fail; relabels ASCII characters as blob bytes.	—	Cannot fail; produces a 4-byte blob from the 4 bytes of the integer.	Cannot fail; produces a 4-byte blob from the 4 bytes of the integer.
as-sint	Not usually useful; converts a 1-, 2-, 3-, or 4-byte string to a blob and then packs it up into a signed integer.	Not usually useful; converts only 1-, 2-, 3-, or 4-byte blobs.	—	Cannot fail; converts to a signed integer, negative if a larger unsigned integer would fit into a positive signed integer.
as-string	—	Relabels as string bytes, if printable characters	Converts to a 4-byte blob, then processes it as a blob (which fails except for a few special integers)	Converts to a 4-byte blob, then processes as a blob (which fails except for a few special integers)
as-uint	Not usually useful; converts a 1-, 2-, 3-, or 4-byte string to a blob and then a signed integer.	Not usually useful; converts only 1-, 2-, 3-, or 4-byte blobs.	Cannot fail; converts to an unsigned integer, and a negative signed integer becomes a large unsigned integer.	—
to-blob	Must be in the form "01:02:03"	—	Cannot fail; produces a 4-byte blob from the 4 bytes of the integer.	Cannot fail; produces a 4-byte blob from the 4 bytes of the integer.
to-sint	Must be in the form n or $-n$.	1-, 2-, 3-, or 4-byte blobs only.	—	Converts only if it is not too big to fit into a signed integer.
to-string	—	Cannot fail	Cannot fail	Cannot fail
to-uint	Must be in the form n .	1-, 2-, 3-, or 4-byte blobs only.	Nonnegative only.	—

Expression Functions

The sections below list the expression functions. Expressions must be enclosed in parentheses.

+, -, *, /, %

Syntax:

(+ arg1 ... argn)

(- arg1 ... argn)

(arg1 ... argn)*

(/ arg1 ... argn)

(% arg1 arg2)

Description:

Arithmetic operations on a signed integer or an expression is convertible to a signed integer. Any argument that cannot convert to a signed integer (and is not null) returns an error. Any argument that evaluates to null is ignored (except that the first argument for `-` and `/` must not evaluate to null). These functions always return signed integers (note that overflow and underflow are currently not caught):

- `+` sums the arguments; if no arguments, the result is 0.
- `-` negates the value of a single argument or, if multiple arguments, successively subtracts the values of the remaining ones from the first one; for example, `(- 3 4 5)` becomes `-6`.
- `*` takes the product of the argument values; if no arguments, the result is 1.
- `/` successively divides the first argument by all of the others; for example, `(/ 100 4 5)` becomes 5. If any argument other than the first equals 0, an error is returned.
- `%` is the modulo arithmetic operator to determine the remainder of the result of the first argument divided by the second one; for example, `(% 12 7)` becomes 5 ($12 / 7 = 1 * 7 + 5$).

Examples:

`(+ 1 2 3 4)` returns 10

`(- 10 5 2)` returns 3

`(* 3 4 5)` returns 60

`(/ 20 2 5)` returns 2

`(/ 20 0)` returns an error

`(% 12 7)` returns 5 ($12 / 7 = 1 * 7 + 5$)

and

Syntax:

(and arg1 ... argn)

Description:

Returns a value that is the datatype of *argn* or null. It evaluates its arguments in order from left to right (the arguments can evaluate to a datatype). If any argument evaluates to null, it stops evaluating the arguments and returns null. Otherwise, it returns the value of the last argument, *argn*.

Examples:

(and "hello" "world") returns "world"

(and (request option 82 1) (request option 82 2)) returns option-82 sub-option 2 if both option-82 sub-option 1 and sub-option 2 are present in the request, otherwise it returns null.

as-blob

Syntax:

(as-blob *expr*)

Description:

Treats *expr* as if it were a blob. If *expr* evaluates to a string, the bytes that make up the string become the bytes of the blob that is returned. If *expr* evaluates to a blob, that blob is returned unmodified. If *expr* evaluates to either kind of integer, a 4-byte blob containing the bytes of the integer is returned.

Examples:

(as-blob "hello world") returns the blob 68:65:6c:6c:6f:20:77:6f:72:6c:64

as-sint

Syntax:

(as-sint *expr*)

Description:

Treats *expr* as if it were a signed integer. If *expr* evaluates to a string or blob of 4 bytes or less, the function returns a signed integer constructed out of those bytes (if longer than 4 bytes, it returns an error). If *expr* evaluates to a signed integer, it returns the value unchanged; if an unsigned integer, it returns a signed integer with the same bit value.

Examples:

(as-sint ff:ff:ff:ff) returns -1

(as-sint 2147483648) returns an error

as-string

Syntax:

(as-string *expr*)

Description:

Treats *expr* as if it were a string. If *expr* evaluates to a string, it returns that string. If *expr* evaluates to a blob, it returns a string constructed from the bytes in the blob, unless they are nonprintable ASCII values, which returns an error. If *expr* evaluates to an integer, it considers its value to be the ASCII value for a single character and returns a string consisting of that one character, unless it is nonprintable, which returns an error.

Examples:

(as-string 97) returns "a"

(as-string 68:65:6c:6c:6f:20:77:6f:72:6c:64) returns "hello world"

(as-string 0) returns an error.

as-uint

Syntax:

(as-uint *expr*)

Description:

Treats *expr* as if it were an integer. If *expr* evaluates to a string or blob of 4 bytes or less, it returns an unsigned integer constructed from those bytes; if longer than 4 bytes, it returns an error. If the result is an unsigned integer, it returns the argument unchanged; if a signed integer, it returns an unsigned integer with the same bit value.

Examples:

(as-uint -2147483648) returns the unsigned integer 2147483648

(as-uint -1) returns the unsigned integer 4294967295

(as-uint ff:ff:ff:ff) returns the unsigned integer 4294967295

ash

Syntax:

(ash *expr shift*)

(lshift *expr shift*)

Description:

Returns an integer or blob with the bits shifted by the *shift* amount. The *expr* can evaluate to an integer, blob or string. If *expr* evaluates to a string, this function tries to convert it to a signed integer, and if that fails, to a blob. If both fail, it returns an error. The *shift* must evaluate to something that is convertible to a signed integer. If *shift* is positive, the shift is to the left; if negative, the shift is to the right. If *expr* results in a signed integer, the right shift is with sign extension. If *expr* results in an unsigned integer or blob, a right shift shifts zero bits in on the most significant bits.

Examples:

(ash 00:01:00 1) returns the blob 00:02:00

(lshift 00:01:00 -1) returns the blob 00:00:80

(ash 1 1) returns the unsigned integer 2

bit

Syntax:

(bit-and *arg1 arg2*)

(bit-andc1 *arg1 arg2*)

(bit-andc2 *arg1 arg2*)

(bit-eqv *arg1 arg2*)

(bit-or *arg1 arg2*)

(bit-orc1 *arg1 arg2*)

(bit-orc2 *arg1 arg2*)

(bit-xor *arg1 arg2*)

Description:

Return the result of a bit-wise boolean operation on the two arguments. The data type of the result is a signed integer if both arguments result in either kind of integer, otherwise the result is a blob. The *arg1* and *arg2* arguments must evaluate to two integers, two blobs of equal length, or one integer and one blob of length 4. If either argument evaluates to a string, the function tries to convert the string to a signed integer, and if that fails, to a blob. After this conversion, the results must match the criteria mentioned above. If these conditions are not met, it returns an error.

Operations with **c1** and **c2** indicate that the first and second arguments, respectively, are complemented before the operation.

Examples:

(bit-and 00:20 00:ff) returns 00:20

(bit-or 00:20 00:ff) returns 00:ff

(bit-xor 00:20 00:ff) returns 00:df

(bit-andc1 00:20 00:ff) returns 00:df

bit-not

Syntax:

(bit-not *expr*)

Description:

Returns a value that is the bit-by-bit complement of *expr*. The expression must evaluate to an integer of either type, or a blob. If it evaluates to a string, the function tries to convert it to a signed integer; if that fails, to a blob, and if that fails, returns an error. The datatype of the result is the same as the result of evaluating *expr* and any subsequent conversions.

Examples:

(bit-not ff:ff) returns 00:00

(bit-not 1) returns 4294967295

(bit-not "hello world") returns an error

byte

Syntax:

(byte *arg1*)

Description:

Eases creation of one-byte blobs. It returns this blob depending on the data type:

- **sint, uint**—Returns the low order byte of the integer.
- **blob**—Returns the last byte in the blob.
- **string**—Returns the last byte in the string.

Examples:

(byte 150) returns a blob of 96

(byte 0x96) returns a blob of 96

comment

Syntax:

(comment *comment expr1 ... exprn*)

Description:

Does not evaluate its first argument and returns null if there is only one argument. Evaluates arguments *expr1* through *exprn*, and returns the value of *exprn* if there is more than one argument.

Examples:

(comment "this is a comment that won't get lost" (request option 82 1))

concat

Syntax:

(concat *arg1 ... argn*)

Description:

Concatenates the values of the arguments into a string or blob (ignoring null arguments). The first argument (*arg1*) must evaluate to a string or a blob; if it evaluates to an integer, the function converts it to a blob. The

datatype of *arg1* (after any conversion) determines the datatype of the result. The function converts all subsequent arguments to the datatype of the result, and if this conversion fails, returns an error.

Examples:

(concat "hello" "world") returns "helloworld"

(concat -1 "world") returns an error

(concat -1 00:01:02) returns the blob ff:ff:ff:ff:00:01:02

datatype

Syntax:

(datatype *expr*)

Description:

Returns the datatype of the result of the expression (*expr*). If the expression cannot evaluate *expr*, it returns an error, otherwise it returns the datatype as a string, which can be:

- "unset" (internal, considered as null)
 - "null"
 - "uint"
 - "sint"
 - "string"
 - "blob"
-

dotimes

Syntax:

(dotimes (var *count-expr* [*result-expr*]) *exp1* ... *expn*)

Description:

Creates an environment with a single local integer variable, *var*, which is initially set to zero, and evaluates *exp1* through *expn*. It then increments *var* by one, and if it is less than *count-expr*, evaluates *exp1* through *expn* again. When *var* is equal to or greater than *count-expr*, the function evaluates *result-expr* and returns it as the result of the entire **dotimes**. If there is no *result-expr*, the function returns null.

The *var* defines a local variable, and must be an alphabetic name. The *count-expr* must evaluate to an integer or be convertible to one. The *exp1* through *expn* are expressions that can evaluate to any data type. The *result-expr* is optional, and if it appears, it can evaluate to any data type. When the function evaluates *count-expr*, *var* is not bound and cannot appear in *count-expr*. Alternatively, *var* is bound for the evaluation of *result-expr* and has the value of *count-expr*. If *result-expr* is omitted, the function returns null.



Note Be careful changing the value of *var* in *exp1* through *expn*, because you can easily create an infinite loop (see the example).

Examples:

(let (x y) (setq x 01:02:03) (dotimes (i (length x)) (setq y (concat (substring x i 1) y)))) returns 03:02:01
 (dotimes (i 10) (setq i 1)) loops forever!

environmentdictionary

Syntax:

(environmentdictionary {get | put *val* | delete} *attr*)

Description:

Gets, puts, or deletes a DHCP extension environment dictionary attribute value. The *val* is the value of the attribute and *attr* is the attribute name. Both are converted to a string regardless of their initial datatype. The initial environment dictionary cannot be changed, but it can be shadowed (you can redefine something that is in the initial dictionary, but if you remove it, then the original initial value is still there). Note that the **get** keyword is not optional for a “get.”

Examples:

nrcmd> dhcp set initial-environment-dictionary=first=one, second=2

(environmentdictionary get "first") returns "one"

(environmentdictionary get "second") returns "2" (note string 2)

(environmentdictionary put "two" "second") returns "second"

(environmentdictionary delete "first") returns null

equal, equali

Syntax:

(equal *expr1 expr2 expr3*)

(equali *expr1 expr2 expr3*)

Description:

The **equal** function evaluates the equivalency of the result of evaluating *expr1* and *expr2*. If they are equal, it returns:

1. The value of *expr3*, if specified, else
2. The value (and datatype, after possible string conversion) of *expr2*, as long as *expr2* is not null, else
3. The string “*T*” (since returning null would incorrectly indicate a failed comparison).

If *expr1* and *expr2* are not equal, the function returns null.

The arguments can be any datatype. If different, the function converts them to strings (which cannot fail) before comparing them. Note that any string conversion is performed using the equivalent of (**to-string** ...). Thus, the blob 61:62 is not equal to the “ab” string. Note also that a one-byte blob 01 is not equal to a literal integer 1 (both are converted to strings, and the “01” and “1” strings are not equal).

The **equali** function is identical to the **equal** function, except that if the comparison is for strings (either because string arguments were used or because the arguments were converted to strings), a case insensitive comparison is used.

Examples:

(equal (request option "dhcp-class-identifier") "docsis") returns the string "docsis" if the value of the option dhcp-class-identifier is a string identical to "docsis"

(equali "abc" "ABC") returns "ABC"

(equal "abc" "def") returns null

(equal "ab" (as-string 61:62)) "this is true") returns "this is true"

(equal "ab" 61:62 "this is not true") returns null

(equal 01:02:03 01:02:03) returns 01:02:03

(equal (as-blob "ab") 61:62) returns 61:62

(equal 1 (to-blob 1)) returns null

(equal (null) (request option 20)) returns "*T*" if there is no option 20 in the packet

error

Syntax:

(error)

Description:

Returns a “no recovery” error that causes the entire expression evaluation to fail unless there is a **try** function above the **error** function evaluation.

if

Syntax:

(if cond [then else])

Description:

Evaluates the condition expression *cond* in an *if-then-else* sense. If *cond* evaluates to a value that is nonnull, it returns the result of evaluating the *then* argument; otherwise it returns the result of evaluating the *else* argument. Both *then* and *else* are optional arguments. If you omit the *then* and *else* arguments, the function simply returns the results of evaluating the *cond* argument. If you omit the *else* argument and *cond* evaluates to null, the function returns null. There are no restrictions on the data types of any of the three arguments.

Examples:

```
(if (equali
    (substring (request option "dhcp-class-identifier") 0 6)
    "docsis"
    (request option 82 1))
```

returns sub-option 1 of option 82 if the first six characters of the dhcp-class-identifier are "docsis" in any case; otherwise returns null.

ip-string

Syntax:

(**ip-string** *blob*)

Description:

Returns the string representation of the four-byte IP address *blob* in the form "*a.b.c.d*". The single argument *blob* must evaluate to a blob or be convertible into one. If the blob exceeds four bytes, the function uses only the first four to create the IP address string. If the blob has fewer bytes, the function considers the right-most bytes as zero when it creates the IP address string.

Examples:

(**ip-string 01:02:03:04**) returns "1.2.3.4"

(**ip-string -1**) returns "255.255.255.255"

(**ip-string (as-blob "hello world")**) returns "104.101.108.108"

ip6-string

Syntax:

(**ip6-string** *blob*)

Description:

Returns the string representation of a 16-byte IPv6 address *blob* in the form "*a:b:c:d:e:f:g:h*". The single argument *blob* must evaluate to a blob or be convertible into one. If the blob exceeds 16 bytes, the function uses only the first 16 to create the IPv6 address string. If the blob has fewer bytes, the function considers the right-most bytes as zero when it creates the IPv6 string.



Note Since there is more than one acceptable way to represent an IPv6 address as a string, comparing the string format of IPv6 addresses is likely to yield inconsistent results. It is best to compare IPv6 addresses as blob values, where no ambiguity exists in the representation of the addresses. See **to-ip6**, if you already have a string formatted IPv6 address.

Examples:

(**ip6-string (as-blob "hello world")**) returns "6865:6c6c:6f20:776f:726c:6400::"

is-string

Syntax:

(is-string *expr*)

Description:

Returns the value of *expr*, if the result of evaluating *expr* is a string or can be used as a string, otherwise it returns null. That is, if **as-string** does not return an error, then **is-string** returns the value of *expr*.

Examples:

(is-string 01:02:03:04) returns null

(is-string "hello world") returns "hello world"

(is-string 68:65:6c:6c:6f:20:77:6f:72:6c:64) returns the blob 68:65:6c:6c:6f:20:77:6f:72:6c:64

length

Syntax:

(length *expr*)

Description:

Returns an integer whose value is the length, in bytes, of the value of *expr*. The argument *expr* can evaluate to any datatype. Integers always have length 4. The length of a string does not include any zero byte that may terminate the string.

Examples:

(length 1) returns 4

(length 01:02:03) returns 3

(length "hello world") returns 11

let

Syntax:

(let (*var1 ... varn*) *expr1 ... exprn*)

Description:

Creates an environment with local variables *var1* through *varn*, which are initialized to a null value (you can give them other values by using the **setq** function). Once the local variables are initialized to null, the function evaluates expressions *expr1* through *exprn* in order. It then returns the value of its last expression, *exprn*. The benefit of this function is that you can use it to calculate a value once, assign it to a local variable, then reuse that value in other expressions without having to recalculate it. Variables are case-sensitive.

Examples:

```
(let (x)
  (setq x (substring (request option "dhcp-class-identifier") 0 6))
  (if (equali x "docsis") "client-class-1")
  (if (equali x "something else") "client-class-2"))
```

log

Syntax:

(**log** *severity expr*)

Description:

Logs the result of converting *expr* to a string. The *severity* and *expr* must be a string and are converted to one if they do not evaluate to one. The *severity* can also be null; if a string, it must have one of these values:

- "debug"
- "activity" (the default if severity is null)
- "info"
- "warning"
- "error"

**Note**

Logging consumes considerable server resources, so limit the number of **log** function evaluations you put in an expression. Even if "error" severity is logged, the log function does not return an error. This only tags the log message with an error indication. See the **error** function to return an error as part of a function evaluation.

mask-blob

Syntax:

(**mask-blob** *mask-size length*)

Description:

Returns a blob that contains the mask of length *mask-size* starting from the high-order bit of the blob, with a blob length of *length*. The *mask-size* is an expression that evaluates to an integer or must be convertible to one. Likewise the *length*, which cannot be smaller than the *mask-size*, but has no fixed limit except that it must be zero or positive. If *mask-size* is less than zero, it denotes a mask length calculated from the right end of the blob.

Examples:

(**mask-blob 1 4**) yields 80:00:00:00

(**mask-blob 4 2**) yields f0:00

(**mask-blob 31 4**) yields ff:ff:ff:fe

(**mask-blob -1 4**) yields 00:00:00:01

mask-int

Syntax:

(mask-int *mask-size*)

Description:

Returns an integer that contains a mask of *mask-size*, starting from the high-order bit of the integer. The *mask-size* is an expression that evaluates to an integer or must be convertible to one. If *mask-size* is less than zero, it denotes a mask length calculated from the right end of the integer.

Examples:

(mask-int 1) yields 0x80000000

(mask-int 4) yields 0xf0000000

(mask-int 31) yields 0xffffffe

(mask-int -1) yields 0x00000001

not

Syntax:

(not *expr*)

Description:

expr is an expression that can evaluate to a string, a blob, or an integer. If the result of that evaluation is non-null, then null is returned. If the result of that evaluation is null, then a nonnull value is returned. The nonnull value returned when the value of *expr* is null is not guaranteed to remain the same over two calls.

Examples:

(not "hello world") returns null

null

Syntax:

(null [*expr1* ... *exprn*])

Description:

Returns null and does not evaluate any of its arguments.

or, pick-first-value

Syntax:

(or *arg1*... *argn*)

(pick-first-value *arg1*... *argn*)

Description:

Evaluates the arguments sequentially. When the evaluation of an argument returns a nonnull value, that value is returned. No other arguments are evaluated after one argument returns a nonnull value. Otherwise, returns the value of the last argument, *argn*. The datatypes need not be the same.

Examples:

```
(or
  (request option 82 1)
  (request option 82 2)
  01:02:03:04)
```

returns the value of sub-option 1 in option 82, and if that does not exist, returns the value of sub-option 2, and if that does not exist, returns 01:02:03:04.

progn, return-last

Syntax:

```
(progn arg ... argn)
```

```
(return-last arg ... argn)
```

Description:

Evaluates arguments sequentially and returns the value of the last argument, *argn*.

Examples:

```
(progn
  (log (null) "I was here")
  (request option 82 1))

(return-last
  (log (null) "I was here")
  (request option 82 1))
```

regex

Syntax:

```
(regex expr1 expr2 var1... varn)
```

```
(regex expr1 expr2)
```

Description:

Searches for sub-strings, matching with regular-expression pattern (*expr1*), in specified target-string (*expr2*) and sets them to specified variables *var1*, *var2*, or *varn*. That means, first sub-string, matching with regular-expression pattern (*expr1*), in specified target-string (*expr2*), will be set to *var1*, second sub-string will be set to *var2*, and so on. You must precede it with the **let** function when specifying variables. This function can also be used without variables, in this case, it returns first sub-string matching with regular-expression pattern (*expr1*), in specified target-string (*expr2*).

As regular-expression pattern matching works only with strings, both patterns (*expr1*) and target-string (*expr2*) must be strings. If they are not, you should use **as-string** function as used in example below.

Examples:

```
(let (x y z)
  (regex "LID[0-9]{6};")
  (as-string (request option 82 2))
  x
  y
  z)
(log (null) "I was here")
```

(regex "[H][a-z]+" "Hello World") returns "Hello".

request

Syntax:

(request [get | get-blob] [relay [number]] packetfield)

Description:

Valid values for the DHCPv4 *packetfield* are:

- op** (blob 1)
- htype** (blob 1)
- hlen** (blob 1)
- hops** (blob 1)
- xid** (uint)
- secs** (uint)
- flags** (uint)
- ciaddr** (blob 4)
- yiaddr** (blob 4)
- siaddr** (blob 4)
- giaddr** (blob 4)
- chaddr** (blob *hlen*)
- sname** (string)
- file** (string)

The **request** *packetfield* function returns the value of the named field from the request packet. DHCP request packets contain named fields as well as options in an option area. This form of the request function is used to retrieve specific named fields from the request packet. The **relay** keyword is described in the **request option** function.

The *packetfield* values defined in RFC 2131 are listed above. There are several *packetfield* values that can be requested which do not appear in exactly these ways in the raw DHCP packet. These take data that appears in the packet and combine it in commonly used ways. In these explanations, the packet contents assumed are:

hlen = 1 *htype* = 6 *chaddr* = 01:02:03:04:05:06

macaddress-string (string)—Returns the MAC address in *hlen*, *htype*, *chaddr* format (for example, “1,6,01:02:03:04:05:06”)

macaddress-blob (blob)—Returns the MAC address in *hlen:htype:chaddr* format (for example, 01:06:01:02:03:04:05:06)

macaddress-clientid (blob)—Returns a client-id created from the MAC address in the Microsoft *htype:chaddr* client-id format (for example, 01:01:02:03:04:05:06)

Valid values for the DHCPv6 *packetfield* are:

msg-type (uint)

msg-type-name (string)

xid (uint)

relay-count (uint)

hop-count (uint)

link-address (blob 16)

peer-address (blob 16)

The **msg-type** packet field for DHCPv6 describes the current relay or client message type, and has the values:

1=SOLICIT, 2=ADVERTISE, 3=REQUEST, 4=CONFIRM, 5=RENEW, 6=REBIND, 8=RELEASE, 9=DECLINE, 11=INFORMATION-REQUEST, 12=RELAY-FORWARD

The **msg-type-name** packet field returns a string of the message type name. The string value is always uppercase; for example, SOLICIT.

The **xid** is the 24-bit client transaction ID, and the **relay-count** is the number of relay messages in the request.

If a DHCPv6 packet field is requested from a DHCPv4 packet, an error is returned. The inverse is also true.

Examples:

(request get ciaddr) returns the ciaddr if it exists, otherwise returns null

(request ciaddr) is the same as **(request get ciaddr)**

(request giaddr) returns the giaddr if it is non-zero, otherwise returns null.

request dump

Syntax:

(request dump)

Description:

Dumps the current request packet to the log file, after the function evaluates the expression. Note that not all expression evaluations support the **dump** keyword, and when unsupported, it is ignored.

request option

Syntax:

```
(request [get | get-blob] [relay [n ]] option opt [{enterprise-id n} | {vendor string}] [instance n] [[subopt | {option opt}] [{enterprise-id n} | {vendor string}] [instance n] [[sub-subopt | {option opt}] [{enterprise-id n} | {vendor string}] [instance n]] [instance-count | {index n} | count])
```

Description:

Returns the value of the option from the packet. The keywords are:

- **get**—Optional and assumed if omitted.
- **get-blob**—Returns the data as a blob, providing direct access to the option bytes.
- **relay**—Applies to IPv6 packets only, otherwise returns an error. Requests a relay option instead of a client option. The *n* indicates the *n*th closest relay agent to the client; if omitted, 0 (the relay agent nearest to the client) is assumed.
- **option**—Options are specified with the *opt* argument, which must evaluate to an integer or a string. If it does not evaluate to one of these, the function does not convert it and returns an error. Valid string values for the *opt* specifier are the same as those used for extensions.
- **enterprise-id**—After an option or suboption, and for DHCPv4 and DHCPv6, returns only the data bytes after the given enterprise-id in the packet, instead of the entire data of an option.
- **vendor**—After an option or suboption, requests that the vendor custom option definition be used for decoding the data in the option. Does not apply to DHCPv6 options. Note that if no definition exists for the specified vendor string, no error is issued and the standard definition of an option is used (or, if none, it is assumed to be a blob).
- **instance**—Selects the (*n* + 1)th instance of the preceding option or suboption. Instances start at 0. (You cannot use the instance and instance-count together in a single request function.)
- **instance-count**—Returns the number of instances of the preceding option or suboption, and is usually used to loop through all instances of it.
- **index**—Selects the (*n* + 1)th value in an option that contains multiple values. For example, **index 0** returns the first value and **index 1** returns the second value.
- **count**—Returns the number of relevant data items in the preceding option, and is usually used with the **index** keyword to loop through all data values for an option or suboption.

The only string-valued suboption names defined for the *subopt* (suboption) specifier are for the relay-agent-info option (82) and are listed in the **DHCPv4 and BOOTP Options** table of the [Decoded DHCP Packet Data Items, on page 449](#) section.

The **request option** function returns a value with a datatype depending on the option requested. This shows how the datatypes in the table correspond to the datatypes returned by the **request** function:

Option Data Type	Returned Data Type
blob	blob
IP address	4-byte blob
string	string
8-bit unsigned integer	uint

Option Data Type	Returned Data Type
16-bit unsigned integer	uint
32-bit unsigned integer	uint
integer	sint
byte-valued boolean	sint=1 if true, null if false

Examples:

(request option 82) returns the relay-agent-info option as a blob

(request option 82 1) returns just the circuit-id (1) suboption

(request option 82 "circuit-id") is the equivalent **(request option 82 1)**

(request option "domain-name-servers") returns the first IP address from the domain-name-servers option

(request option 6 index 0) is the equivalent **(request option 6 count)** returns the number of IP addresses

(request get-blob option "dhcp-class-identifier") returns the value as a blob, not a string

(request option "IA-NA" instance 2 option "IAADDR" instance 3) returns the third instance of the IA-NA option, and the fourth instance of the IAADDR option encapsulated in the IA-NA option

(request get-blob option "vendor-opts" enterprise-id 1234) returns a blob of the option data for enterprise-id 1234

(request option "vendor-opts" enterprise-id 1234 3) returns suboption 3 from the requested vendor option data

requestdictionary

Syntax:

(requestdictionary {get | put *val* | delete} *attr*)

Description:

Gets, puts, or deletes a DHCP extension request dictionary attribute value. *val* is the value of the attribute and *attr* is the attribute name. Both are converted to a string regardless of their initial datatype. Note that the **get** keyword is not optional for a “get.”

response

Syntax:

(response [get | get-blob] [relay [*number*]] *packetfield*)

Description:

Returns the value of the named *packetfield* from the response packet. The description and valid values are identical to those for the **request packetfield** function.

response dump

Syntax:

(response dump)

Description:

Dumps the current response packet to the log file after the function evaluates the expression. Note that not all expression evaluations support the **dump** keyword, and when unsupported, it is ignored.

response option

Syntax:

(response [get | get-blob] [relay [n]] option opt [{enterprise-id n} | {vendor string}] [instance n] [[subopt | {option opt}] [{enterprise-id n} | {vendor string}] [instance n] [[sub-subopt | {option opt}] [{enterprise-id n} | {vendor string}] [instance n]] [instance-count | {index n} | count])

Description:

Returns the value of the option from the packet. The keywords are identical to those for the **request** function.

responsedictionary

Syntax:

(responsedictionary {get | put val | delete} attr)

Description:

Gets, puts, or deletes a DHCP extension response dictionary attribute value. The *val* is the value of the attribute and *attr* is the attribute name. Both are converted to a string regardless of their initial datatype. Note that the **get** keyword is not optional for a “get.”

search

Syntax:

(search arg1 arg2 fromend)

Description:

Searches *arg1* for a subsequence in *arg2* that exactly matches. If found, it returns the index of the element in *arg2* where the subsequence begins (unless you set the *fromend* argument to “true” or some other arbitrary nonnull value); otherwise it returns null. (If *arg1* is null, it returns 0; if *arg2* is null, it returns null.) The function does an implicit **as-blob** conversion on both arguments. Thus, it compares the actual byte sequences of strings and blobs, and sints and uints become 4-byte blobs for the purpose of comparison.

A nonnull *fromend* argument returns the index of the leftmost element of the rightmost matching subsequence.

Examples:

(search "test" "this is a test") returns 10

(search "test" "this test test test" "true") returns 15

setq

Syntax:

(setq *var expr*)

Description:

Only valid within the **let** function. *var* must be one of the *var1* through *varn* local variables defined in the enclosing **let** function.

Examples:

See the **let** function for examples

starts-with

Syntax:

(starts-with *expr prefix-expr*)

Description:

Returns the value of *expr* if the *prefix-expr* value matches the beginning of *expr*, otherwise null. If *prefix-expr* is longer than *expr*, it returns null. The function returns an error if *prefix-expr* cannot be converted to the same datatype as *expr* (string or blob), or if *expr* evaluates to an integer.

Examples:

(starts-with "abcdefghijklmnop" "abc") returns "abcdefghijklmnop"

(starts-with "abcdefgji" "bcd") returns null

(starts-with 01:02:03:04:05:06 01:02:03) returns 01:02:03:04:05:06

(starts-with "abcd" (as-string 61:62)) returns "abcd"

(starts-with "abcd" 61:62) returns null

(starts-with "abcd" (to-string 61:62)) returns null

substring

Syntax:

(substring *expr offset len*)

Description:

Returns *len* bytes of expression *expr*, starting at *offset*. The *expr* can be a string or blob; if an integer, converts to a blob. The result is a string or a blob, or null if any argument evaluates to null. If:

- *offset* is greater than the length *len*, the result is null.
- *offset* plus *len* is data beyond the end of *expr*, the function returns the rest of the data in *expr*.
- *offset* is less than zero, the offset is from the end of the data (the last character is index -1 , because $-0=0$, which references the first character).
- This references data beyond the beginning of data, the offset is considered to be zero.

Examples:

(substring "abcdefg" 1 6) returns bcdefg

(substring 01:02:03:04:05:06 3 2) returns 04:05

synthesize-host-name

Syntax:

(synthesize-host-name *method namestem*)

Description:

Generates a hostname based on the configured method (if none is specified), or the specified *method* and *namestem*.

The valid methods for the *method* argument depend on whether a DHCPv4 or DHCPv6 request is being processed. For DHCPv4, the valid methods are: **default** or one of the v4-synthetic-name-generator enumeration values of: **address**, **client-id**, or **hashed-client-id**. For DHCPv6, the valid methods are: **default** or one of the v6-synthetic-name-generator enumeration values of: **duid**, **hashed-duid**, **cablelabs-device-id**, or **cablelabs-cm-mac-addr**. For more information on these enumeration methods, see [Generating Synthetic Names in DHCPv4 and DHCPv6](#), on page 248.

The *namestem* argument specifies the *synthetic-name-stem* value of the DNS update configuration (see [Creating DNS Update Configurations](#), on page 256).

Examples:

(synthesize-host-name) returns "dhcp-rhfxxi5pkjp6o"

(synthesize-host-name "duid" "test") returns "test-00030001010203040506"

(synthesize-host-name "client-id" "test") returns "test-00030001010203040506"

to-blob

Syntax:

(to-blob *expr*)

Description:

Converts an expression to a blob. If:

- *expr* evaluates to a string it must be in “*nn:nn:nn*” format. This function returns a blob that is the result of converting the string to a blob. If the function cannot convert the string to a blob, it returns an error.
- *expr* evaluates to a blob, it returns that blob.
- *expr* evaluates to an integer, it returns a four-byte blob representing the bytes of the integer in network order. (See [Datatype Conversions](#), on page 320.)

Examples:

(to-blob 1) returns 00:00:00:01

(to-blob "01:02") returns 01:02

(to-blob 02:03) returns 02:03

to-ip, to-ip6

Syntax:

(to-ip *expr*)

(to-ip6 *expr*)

Description:

Converts an expression as string, blob, or integer to an IP address. If:

- A string, it must be in dotted decimal IP address format for IPv4 or colon-formatted format for IPv6. Returns the blob IP address determined by parsing the string into an IP address.
- The result is a blob, it returns the first 4 bytes for (to-ip ...) and the first 16 bytes for (to-ip6 ...). If the blob is less than the 4 bytes for to-ip or 16 bytes for to-ip6, it pads the argument blob with zero bytes in the high order bytes.
- The result is an integer, it converts the integer (of either type) into a blob. Because the integers and blobs are in network order, no order change is required.

to-lower

Syntax:

(to-lower *expr*)

Description:

Takes a string and produces a lowercase string from it. When using the *client-lookup-id* attribute to calculate a client-specifier to look up a client-entry in the CNRDB local store (as opposed to LDAP), the resulting string must be lowercase. Use this function to easily make the result of the *client-lookup-id* a lowercase string. You may or may not want to use this function when accessing LDAP using the *client-lookup-id*.

to-sint

Syntax:

(to-sint *expr*)

Description:

Converts an expression to a signed integer.

If *expr* evaluates to a string, it must be in a format that can be converted into a signed integer, else the function returns an error. If:

- *expr* evaluates to a blob of one to four bytes, the function returns it as a signed integer.
- *expr* evaluates to a blob of more than 4 bytes in length, it returns an error.
- *expr* evaluates to an unsigned integer, it returns a signed integer with the same value, unless the value of the unsigned integer was greater than the largest positive signed integer, in which case it returns an error.
- *expr* evaluates to a signed integer, it returns that value.

Examples:

(to-sint "1") returns 1

(to-sint -1) returns -1

(to-sint 00:02) returns 2

(to-sint "00:02") returns an error

(to-sint "4294967295") returns 2147483647

to-string

Syntax:

(to-string *expr*)

Description:

Converts an expression to a string. If *expr* evaluates to a string, it returns it; if a blob or integer, it returns its printable representation. It never returns an error if *expr* itself evaluates without error, because every value has a printable representation.

Examples:

(to-string "hello world") returns "hello world"

(to-string -1) returns "-1"

(to-string 02:04:06) returns "02:04:06"

to-uint

Syntax:

(to-uint *expr*)

Description:

Converts an expression to an unsigned integer. If

expr evaluates to a string, it must be in a format that can be converted into an unsigned integer, else the function returns an error. If:

- *expr* evaluates to a blob of one to four bytes, it returns it as an unsigned integer.
- *expr* evaluates to a blob of more than 4 bytes in length, it returns an error.
- *expr* evaluates to a signed integer, it returns an unsigned integer with the same value, unless the value of the signed integer less than zero, in which case it returns an error.
- *expr* evaluates to an unsigned integer, the function returns that value.

Examples:

(to-uint "1") returns 1

(to-uint 00:02) returns 2

(to-uint "4294967295") returns 4294967295

(to-uint "00:02") returns an error

(to-uint -1) returns an error

translate

Syntax:

(translate *expr search replace*)

Description:

Takes as an argument an expression that evaluates to a sequence of bytes (either a string or a blob), and replaces various characters or bytes that appear in *search* with corresponding values (in the same position) in *replace*. If:

- *expr* is a string or blob, the value is left as it is, otherwise it is forced to be a string. If, after processing, *expr* is a string, *search* and *replace* must be strings.
- *expr* is a blob, both *search* and *replace* must also be blobs.
- *replace* is shorter than *search*, the bytes or characters in *search* that do not have corresponding bytes or characters in *replace* are dropped from the output.
- *replace* does not appear, all the bytes or characters in *search* are removed from *expr*.

Examples:

(translate "Hello apple and eve" "abcdef" "123456") returns "H5llo 1ppl5 1n4 5v5"

(translate "a&b\$c%d" "%\$&") returns "abcd"

try

Syntax:

(try *expr failure-expr*)

Description:

Evaluates *expr* and returns the result of that evaluation if there were no errors encountered during the evaluation. If an error occurs while evaluating *expr* then:

- If there is a *failure-expr* and it evaluates without error, it returns the result of that evaluation as the result of the **try** function.
- If there is a *failure-expr* and the function encounters an error while evaluating *failure-expr*, it returns that error.
- If there is no *failure-expr*, the **try** returns null.

Examples:

(try (try (*expr*) (*complex-failure-expr*)) "string-constant") ensures that the outer try never returns an error (because evaluating "string-constant" cannot fail)

(try (error) 01:02:03) always returns 01:02:03

(try 1 01:02:03) always returns 1

(try (request option 82) "failure") never returns "failure" because (request option 82) turns null if there is no option-82 in the packet and does not return an error

(try (request option "junk") "failure") returns "failure" because "junk" is not a valid option-name.

validate-host-name

Syntax:

(validate-host-name *hostname*)

Description:

Takes the *hostname* string and returns a validated hostname, which can be the same as the input *hostname* or modified as follows:

- Space and underscore characters mapped to a hyphen.
- Invalid hostname characters removed. Valid characters are A-Z, a-z, 0-9, and hyphen.
- Null labels removed (".." changed to ".").
- Each label in the hostname truncated to 63 characters.

Examples:

(validate-host-name "a b c d e f") returns "a-b-c-d-e-f"

(validate-host-name "_a_b_c_d_e_f_") returns "a-b-c-d-e-f"

(validate-host-name "abcdef") returns "abcdef"

(validate-host-name "a&b*c#d@!e()f") returns "abcdef"

Expression Examples

These examples provide the maximum support for option 82 processing. They set up clients to limit, those not to limit, and those that exceed configuration limits and should be assigned to an over-limit client-class. There are separate scopes and selection tags for each of the three classes of clients. These examples assume

the following CPNR configuration environment (which will certainly differ from any real environment and is used just for illustration).

- **Client-classes**—limit, no-limit, and over-limit.
- **Scopes**—10.0.1.0 (primary), 10.0.2.0 and 10.0.3.0 (secondaries), named for their subnets.
- **Selection tags**—limit-tag, no-limit-tag, and over-limit-tag. The scopes are named for the address pools that they represent. The selection tags are allocated to the scopes with 10.0.1.0 getting limit-tag, 10.0.2.0 getting no-limit-tag, and 10.0.3.0 getting over-limit-tag.

Related Topics

[Limitation Example 1: DOCSIS Cable Modem, on page 346](#)

[Limitation Example 2: Extended DOCSIS Cable Modem, on page 347](#)

[Limitation Example 3: DSL over Asynchronous Transfer Mode, on page 348](#)

Limitation Example 1: DOCSIS Cable Modem

The test is to determine whether the device is considered a DOCSIS cable modem, and limit the number of customer devices behind every cable modem. The limitation ID for the limit client-class is the cable modem MAC address, included in the *remote-id* suboption of the *relay-agent-info* option.

The expression for the *client-class-lookup-id* attribute on the server is:

```
// Expression to set client-class to no-limit or limit based on remote-id
(if (equal (request option "relay-agent-info" "remote-id")
          (request chaddr))
    "no-limit"
    "limit")
```

The above expression indicates that if the contents of the *remote-id* suboption (2) of the *relay-agent-info* option is the same as the *chaddr* of the packet, then the client-class is *no-limit*, otherwise *limit*.

The *limitation-id* expression for the *limit* client-class is:

```
(request option "relay-agent-info" "remote-id")
```

Use this expression in the following steps:

-
- Step 1** Define the client-classes.
 - Step 2** Define the scopes, their ranges and tags, and if they are primary or secondary. Note the host range for each scope, which is less likely to be misread than if they all have the same host number.
 - Step 3** Define the limitation count. It can go in the *default* policy; if the request does not show a limitation ID, the count is not checked.
 - Step 4** Add an expression in an expression file, *cclookup1.txt*, for the purpose:

```
// Expression to set limitation count based on remote-id
(if (equal (request option "relay-agent-info" "remote-id")
          (request chaddr))
    "no-limit"
    "limit")
```

Step 5 Refer to the expression file when setting the *client-class lookup-id* attribute on the server level.

Step 6 Add another expression for the limitation ID for the client in a `cclimit1.txt` file:

```
// Expression to set limitation ID based on remote-id
(request option "relay-agent-info" "remote-id")
```

Step 7 Refer to this expression file when setting the *limitation-id* attribute for the client-class.

Step 8 Reload the server.

The result of doing this for a previously unused configuration would be to put the first two DHCP clients with a common *remote-id* option 82 suboption value in the *limit* client-class. The third client with the same value would go in the *over-limit* client-class. There are no limits to the number of devices a subscriber can have in the *no-limit* client-class, because it has no configured limitation ID. Any device with a MAC address equal to the value of the *remote-id* suboption is ignored for the purposes of limitation, and goes in the *no-limit* client class, for which there is no limitation ID configured.

Limitation Example 2: Extended DOCSIS Cable Modem

This example is an extension to the example described in [Limitation Example 1: DOCSIS Cable Modem, on page 346](#). In the latter example, all of the cable modems allowed only two client devices beyond them, since a limitation count of two was defined for the *default* policy. In this example, specific cable-modems are configured to allow a different number of devices to be granted IP addresses from the scopes that use the *limit-tag* selection tag.

In this case, you need to explicitly configure any cable modem with more than two addresses behind it in the client-class database. This requires enabling client-class processing server-wide, so that you can look up the client entry for a cable modem in the Cisco Prime Network Registrar or LDAP database. Not finding the cable modem limits the number of devices to two; finding it uses the limitation count from the policy configured for the cable modem.

This example requires just one additional policy, *five*, which allows five devices.

Step 1 Enable client-class processing server-wide.

Step 2 Create the *five* policy with a limitation count of five devices.

Step 3 As in the previous example, use an expression to set a limitation ID for the *limit* client-class. Put the limitation ID in a `cclimit2.txt` file, and the lookup ID in a `cclookup2.txt` file:

```
cclimit2.txt file:
// Expression to set limitation ID
(request option "relay-agent-info" "remote-id")

cclookup2.txt file:
// Expression to set client-class lookup ID
(concat "1,6," (to-string (request option "relay-agent-info" "remote-id")))
```

Step 4 Refer to these files when setting the appropriate attributes.

Step 5 Define some cable modem clients and apply the *five* policy to them.

Step 6 Reload the server.

Limitation Example 3: DSL over Asynchronous Transfer Mode

This example shows how to use expressions to configure Digital Subscriber Line (DSL) access for a subscriber to a service provider using asynchronous transfer mode (ATM) routed bridge encapsulation (RBE). Service providers are increasingly using ATM RBE to configure a DSL subscriber. The DHCP Option 82 support for routed bridge encapsulation feature as of Cisco IOS Release 12.2(2)T enables those service providers to use DHCP to assign IP addresses and option 82 to implement security and IP address assignment policies.

In this scenario, DSL subscribers are identified as individual ATM subinterfaces on a Cisco 7401ASR router. Each customer has their own subinterface in the router and each subinterface has its own virtual channel identifier (VCI) and virtual path identifier (VPI) to identify the next destination of an ATM cell as it passes through ATM switches. The 7401ASR router routes up to a Cisco 7206 gateway router.

Step 1 Set up the DHCP server and interfaces for the router using IOS. This is a typical IOS configuration:

```
Router#ip dhcp-server 170.16.1.2
Router#interface Loopback0
Loopback0(config)#ip address 11.1.1.129 255.255.255.192
Loopback0(config)#exit
Router#interface ATM4/0
ATM4/0(config)#no ip address
ATM4/0(config)#exit
Router#interface ATM4/0.1 point-to-point
ATM4/0.1(config)#ip unnumbered Loopback0
ATM4/0.1(config)#ip helper-address 170.16.1.2
ATM4/0.1(config)#atm route-bridged ip
ATM4/0.1(config)#pvc 88/800
ATM4/0.1(config)#encapsulation aal5snap
ATM4/0.1(config)#exit
Router#interface Ethernet5/1
Ethernet5/1(config)#ip address 170.16.1.1 255.255.0.0
Ethernet5/1(config)#exit
Router#router eigrp 100
eigrp(config)#network 11.0.0.0
eigrp(config)#network 170.16.0.0
eigrp(config)#exit
```

Step 2 In IOS, enable the system to insert the DHCP option 82 data in forwarded BOOTREQUEST messages to a Cisco IOS DHCP server:

```
Router#ip dhcp relay information option
```

Step 3 In IOS, specify the IP address of the loopback interface on the DHCP relay agent that is sent to the DHCP server using the option 82 *remote-id* suboption (2):

```
Router#rbe nasip Loopback0
```

Step 4 In Cisco Prime Network Registrar, enable client-class processing server-wide.

Step 5 Create the *one* policy with a limitation count of one device.

Step 6 Put the packets in the right client-class. All the packets should be in the *limit* client-class. Create a lookup file containing just the value *limit*, then set the client-class lookup ID. In the ccllookup3.txt file:

```
// Sets client-class to limit
"limit"
```

- Step 7** Use an expression to ensure that those packets that are limited have the right limitation ID. Put the expression in a file and refer to that file to set the limitation ID. The *substring* function gets the VPI/VCI by extracting bytes 10 through 12 of the option 82 suboption 2 (*remote-id*) data field. In the *cclimit3.txt* file:

```
// Sets limitation ID
(substring (request option 82 2) 9 3)
```

- Step 8** Reload the server.
-

Debugging Expressions

If you are having trouble with expressions, examine the DHCP log file at server startup. All expressions are printed in such a way as to clarify the nesting of functions, and can help in confirming your intentions. Pay special attention to the **equal** function and any datatype conversions of arguments. If the arguments are not the same datatype, they are converted to strings using code similar to the **to-string** function.

You can set various debug levels for expressions by using the *expression-trace-level* attribute for the DHCP server. All executed expressions are traced to the degree set by the attribute. The highest trace level is 10. If you set the level to at least 2, any failing expression is retried again at level 10.

The trace levels for *expression-trace-level* are (use the number value):

- **0**—No tracing
- **1**—Failures, including those protected by (**try ...**)
- **2**—Total failure retries (with trace level = 6 for retry)
- **3**—Function calls and returns
- **4**—Function arguments evaluated
- **5**—Print function arguments
- **6**—Datatype conversions (everything)

To trace expressions you have trouble configuring, there is also an *expression-configuration-trace-level* attribute that you can set to any level from 1 through 10. If you set the level to at least a 2, any expression that does not configure is retried again with the level set to 6. Gaps in the numbering are to accommodate future level additions. The trace levels for *expression-configuration-trace-level* are (use the number value):

- **0**—No additional tracing
- **1**—No additional tracing
- **2**—Failure retry (the default)
- **3**—Function definitions
- **4**—Function arguments
- **5**—Variable lookups and literal details
- **6**—Everything



CHAPTER 12

Using Extension Points

You can write extensions to affect how Cisco Prime Network Registrar handles and responds to DHCP requests, and to change the behavior of a DHCP server that you cannot normally do using the user interfaces. This chapter describes the extension points to which you can attach extensions for DHCPv4 and DHCPv6.

- [Using Extensions, on page 351](#)
- [Language-Independent API, on page 354](#)
- [Tcl Extensions, on page 357](#)
- [C/C++ Extensions, on page 359](#)
- [DHCP Request Processing Using Extensions, on page 362](#)
- [Extension Dictionaries, on page 373](#)
- [Request and Response Dictionaries, on page 376](#)
- [Extension Point Descriptions, on page 378](#)

Using Extensions

You can alter and customize the operation of the Cisco Prime Network Registrar DHCP server by using extensions, functions that you can write in Tcl or C/C++.

Follow this process to create an extension for use in the DHCP server:

1. Determine the task to perform. What DHCP packet process do I want to modify?
2. Determine the approach to use. How do I want to modify the packet process?
3. Determine the extension point to which to attach the extension.
4. Choose the language (Tcl or C/C++).
5. Write (and possibly compile and link) the extension.
6. Add the extension to the DHCP server configuration.
7. Attach the extension to the extension point.
8. Reload the DHCP server so that it recognizes the extension.
9. Test and debug the results.



Note While upgrading Cisco Prime Network Registrar, it is recommended to recompile all the DHCP C/C++ extensions (dex extensions).

Related Topics

[Creating, Editing, and Attaching Extensions, on page 352](#)

[Determining Tasks, on page 353](#)

[Deciding on Approaches, on page 353](#)

[Choosing Extension Languages, on page 354](#)

Creating, Editing, and Attaching Extensions

You can create, edit, and attach extensions.

You can associate multiple extensions per extension point. Each extension executes in the order specified by the sequence number used when the attachment was created. In the web UI, the sequence is the order in which the extensions appear per extension point on the List DHCP Extension Points page. In the CLI, you use the *sequence-number* value with the **dhcp attachExtension** command.

For more on multiple extensions per extension point, see [Multiple Extension Considerations, on page 357](#).

Local Advanced Web UI

Creating and Attaching Extensions

-
- Step 1** From the **Deploy** menu, choose **Extensions** under the **DHCP** submenu to open the List/Add DHCP Extensions page.
 - Step 2** Click the **Add Extensions** icon to open the Add DHCP Server Extension dialog box.
 - Step 3** After you create an extension, you can attach it to one or more of the extension points on this page. To show the extension points where you can attach extensions, on the List/Add DHCP Extensions page, click **DHCP Extension Points** tab.
 - Step 4** If you attach more than one extension for each extension point, you can change the sequence in which they are processed by clicking the arrow keys to rearrange the entries. To remove the extension, click the **Delete** icon.
-

CLI Command

Use the **extension** command, which requires this syntax:

```
nrcmd> extension name create language extension-file entry-point
```

The *entry-point* is the name of the entry point in the *extension-file*. You can also set an optional *init-entry* attribute value for the initial entry point each time the DHCP server loads the file (see [init-entry, on page 379](#)). You can call this function from any extension point bound to this module. You can also list the extensions using **extension list**.

To attach and detach an extension, use **dhcp attachExtension** and **dhcp detachExtension** for the DHCP server, which require this syntax:

```
nrcmd> dhcp attachExtension extension-point extension-name [sequence-number]
nrcmd> dhcp detachExtension extension-point [sequence-number]
```

The *sequence-number* applies if you attach multiple extensions per extension point, in increasing sequence order ranging from 1 through 32. If omitted, it defaults to 1.

To view the currently registered extensions, use **dhcp listExtensions** command.

Related Topics

[Using Extensions, on page 351](#)

Determining Tasks

The task to which to apply an extension is usually some modification of the DHCP server processing so that it better meets the needs of your environment. You can apply an extension at each of these DHCP server processing points, from receiving a request to responding to the client:

1. Receive and decode the packet.
2. Look up, modify, and process any client-class.
3. Build a response type.
4. Determine the subnet (or link, in the case of DHCPv6).
5. Find any existing leases.
6. Serialize the lease requests.
7. Determine the lease acceptability for the client.
8. Gather and encode the response packet.
9. Update stable storage of the packet.
10. Return the packet.

A more complete list of these steps (along with the extension points to use at each step) appears in [DHCP Request Processing Using Extensions, on page 362](#).

For example, you might have an unusual routing hub that uses BOOTP configuration. This device issues a BOOTP request with an Ethernet hardware type (1) and MAC address in the *chaddr* field. It then sends out another BOOTP request with the same MAC address, but with a hardware type of Token Ring (6). Specifying two different hardware types causes the DHCP server to allocate two IP addresses to the device. The DHCP server normally distinguishes between a MAC address with hardware type 1 and one with type 6, and considers them different devices. In this case, you might want to write an extension that prevents the DHCP server from handing out two different addresses to the same device.

Deciding on Approaches

Many solutions are often available to a single problem. When choosing the type of extension to write, you should first consider rewriting the input DHCP packet. This is a good approach, because it avoids having to know the internal processing of the DHCP server.

For the problem described in [Determining Tasks, on page 353](#), you can write an extension to solve it in either of these ways:

- Drop the Token Ring (6) hardware type packet.
- Change the packet to an Ethernet packet and then switch it back again on exit.

Although the second way involves a more complex extension, the DHCP client could thereby use either reply from the DHCP server. The second approach involves rewriting the packet, in this case using the **post-packet-encode** extension point (see [post-packet-encode, on page 389](#)). Other approaches require other extensions and extension points.

Choosing Extension Languages

You can write extensions in Tcl or C/C++. The capabilities of each language, so far as the DHCP server is concerned, are similar, although the application programming interface (API) is slightly different to support the two very different approaches to language design:

- **Tcl**—Although scripting in Tcl might be somewhat easier than scripting in C/C++, it is interpreted and single-threaded, and might require more resources. However, you might be less likely than in C/C++ to introduce a serious bug, and there are fewer chances of a server failure. Cisco Prime Network Registrar currently supports Tcl version 8.4.
- **C/C++**—This language provides the maximum possible performance and flexibility, including communicating with external processes. However, the C/C++ API is more complex than the Tcl API. Also, the possibility of a bug in the extension causing a server failure is also more likely in C/C++.

Language-Independent API

The following concepts are independent of whether you write your extensions in Tcl or C/C++.

Related Topics

[Routine Signature, on page 354](#)

[Dictionaries, on page 355](#)

[Utility Methods in Dictionaries, on page 355](#)

[Configuration Errors, on page 355](#)

[Recognizing Extensions, on page 356](#)

[Multiple Extension Considerations, on page 357](#)

Routine Signature

You need to define the extension as a routine in a file, which can contain multiple extension functions. You then attach the extension to one or more of the DHCP server extension points. When the DHCP server reaches that extension point, it calls the routine that the extension defines. The routine returns with a success or failure. You can configure the DHCP server to drop a packet on an extension failure.

You can configure one file—Tcl source file or C/C++ .dll or .so file—as multiple extensions to the DHCP server by specifying a different entry point for each configured extension.

The server calls every routine entry point with at least three arguments, the three dictionaries—request, response, and environment. Each dictionary contains many data items, each being a key-value pair:

- The extension can retrieve data items from the DHCP server by performing a get method on a dictionary for a particular data item.
- The extension can alter data items by performing a put or remove operation on many of the same named data items.

Although you cannot use all dictionaries at every extension point, the calling sequence for all routines is the same for every extension point. The extension encounters an error if it tries to reference a dictionary that is not present at a particular extension point. (See [Extension Dictionaries, on page 373](#).)

Dictionaries

You access data in the request, response, and server through a dictionary interface. Extension points include three types of dictionaries—request, response, and environment:

- **Request dictionary**—Information associated with the DHCP request, along with all that came in the request itself. Data is string-, integer-, IP address-, and blob-valued.
- **Response dictionary**—Information associated with generating a DHCP response packet to return to the DHCP client. Data is string-, integer-, IP address-, and blob-valued.
- **Environment dictionary**—Information passed between the DHCP server and extension.

For a description of the dictionaries, see [Extension Dictionaries, on page 373](#).

You can also use the environment dictionary to communicate between an extension attached at different extension points. When encountering the first extension point at which some extension is configured, the DHCP server creates an environment dictionary. The environment dictionary is the only one in which the DHCP server does not fix the names of the allowable data items. You can use the environment dictionary to insert any string-valued data item.

Every extension point in the flow of control between the request and response for the DHCP client (all extension points except **lease-state-change**, depending on the cause of the change) share the same environment dictionary. Thus, an extension can determine that some condition exists, and place a sentinel in the environment dictionary so that a subsequent extension can avoid determining the same condition.

In the previous example, the extension at the **post-packet-decode** extension point determines that the packet was the interesting one—from a particular manufacturer device, BOOTP, and Token Ring—and then rewrites the hardware type from Token Ring to Ethernet. It also places a sentinel in the environment dictionary and then, at a very simple extension at the **post-packet-encode** extension point, rewrites the hardware type back to Token Ring.

Utility Methods in Dictionaries

Each dictionary has associated utility methods with which you can reset the trace level for an extension and log values to an output file.

Configuration Errors

Extensions can fail for many reasons. For example:

- The server cannot find the file.
- The entry point or **init-entry** point does not appear in the file.
- The extension itself can return a failure from an **init-entry** call.

By itself, an extension failure is not fatal and does not prevent the DHCP server from starting. However, if you configure that failed extension at any extension point, the server will not start. Therefore, to debug the configuration process, you can configure the extension at the **init-entry** point (see [init-entry, on page 379](#)) without attaching it to an extension point. When you complete that process successfully, you can attach your extension to an extension point.

Communicating with External Servers

You can write extensions that communicate with external servers or databases to affect the client class or validate incoming DHCP client requests. Writing such extensions is a complex task, requiring considerable skill and debugging expertise. Such extensions must be multithreaded, and need to communicate very swiftly with the external server if the DHCP server performance is to remain at an acceptable level.

Performance degradations can result from extensions stalling the threads that are processing requests. A thread stalls while an extension communicates with an external server. If this interaction takes more than 50 to 100 milliseconds, this severely affects server performance. This might or might not impact you in the particular environment in which you deploy this extension.

One way to avoid having to communicate with an external server synchronously (that is, stalling the incoming DHCP client request processing to communicate with the external server) is to avoid performing this communication while processing the DHCP client request. This sounds obvious, and it also sounds, on the face of it, impossible. However, due to the nature of the DHCP client-server protocol, there is a way to decouple the access to the external server from the DHCP client request processing.

To avoid this bottleneck, use a caching mechanism as part of the extension. When the server calls the extension for a request, have it check a cache (with proper locking to avoid multithreading problems) for the client data. If the client is:

- In the cache (and did not expire), have the extension accept or reject the request depending on the data in the cache.
- Not in the cache, have the extension queue a request to the external server (preferably over UDP), and then drop the DHCP client request. By the time the client retransmits the request, the data should be in the cache.

This caching mechanism requires the extension to have a receiver thread (started and stopped in the **init-entry** extension point). This thread reads the socket and updates the cache with the response. This thread (or a separate one) would also need to time out and remove old items from the cache. Using a single thread, however, might require setting a larger receive socket buffer size.

These techniques are only necessary if the load on the DHCP server is high and the speed of the external server is not high enough to support the required performance of the DHCP server load. However, this situation turns out to be all too common in practice. And, consider what can happen if the external server is unreachable (when connection timeouts are likely to be for minutes and not seconds).

Recognizing Extensions

The DHCP server only recognizes extensions when it initially configures itself at start or reload time. You can change an extension or the configuration for extensions in general. However, until you reload or restart the server, the changes have no effect. Forgetting to reload the DHCP server can be a frequent source of errors while debugging extensions.

The reason Cisco Prime Network Registrar requires a reload is to ensure minimum processing impact by preloading extensions and getting them ready at server configuration time. While this approach is useful in production mode, it might cause some frustration when you debug extensions.

Multiple Extension Considerations

You can register multiple extensions at any extension point. The DHCP server runs all the extensions attached to an extension point before resuming processing, the conditions being:

- An extension should not explicitly set a data item unless the extension explicitly requires that behavior. For example (as described for the drop environment dictionary data item in Table 31-5 on page 31-25), extensions can request dropping the client packet at most extension points.

The server calls the first extension registered at an extension point with *drop* set to False. One or more extensions can set this to True or False. If all extensions were to explicitly set *drop* to either True or False, then the server would take whatever action the last extension to run requested.

This might not be the desired behavior. Thus, for this data item, it is better for an extension to set *drop* to True only if it wants the packet to be dropped. That way, if all extensions play by this rule, the packet would be dropped if any of the extensions request it.

- An extension might want to return immediately if *drop* is True, as there may not be a need for the extension to do its processing if another one desires the packet to be discarded.
- If the environment dictionary is used to store items for use in later extension points, those data item names might want to use a prefix or suffix that is unique to that extension. This reduces the chance for data item name conflicts.
- At least one environment dictionary data item, the *user-defined-data* (see [Table 47: General Environment Dictionary Data Items](#)) that you can use to store data with a lease (for DHCPv4) or client (DHCPv6), requires special attention.

This data item can be difficult for more than one extension to use, unless those extensions take special care to preserve and recognize each other's values. Thus, it might not be possible for more than one extension to assume it can use this data item.

- You must specify whether the extensions should be run first, or last, if such a need exists. For example, you should run extensions that cause the server to drop certain packets first, because this reduces the processing burden on the server (assuming the remaining extensions return immediately if *drop* is true).

Tcl Extensions

If you choose to write your extensions in Tcl, you should understand the Tcl API, how to handle errors and Boolean variables, and how to initialize Tcl extensions. Cisco Prime Network Registrar uses Tcl version 8.4.

Related Topics

[Tcl Application Program Interface, on page 358](#)

[Dealing with Tcl errors, on page 358](#)

[Handling Boolean Variables in Tcl, on page 359](#)

[Configuring Tcl Extensions, on page 359](#)

[Init-Entry Extension Point in Tcl, on page 359](#)

Tcl Application Program Interface

Every Tcl extension has the same routine signature:

```
proc yourentry { request response environ } { # your-code }
```

To operate on the data items in any dictionary, you must treat these arguments as commands. Thus, to get the *giaddr* of the input packet, you would write:

```
set my_giaddr [ $request get giaddr ]
```

This sets the Tcl variable *my_giaddr* to the string value of the *giaddr* in the packet; for example, 10.10.1.5 or 0.0.0.0.

You could rewrite the *giaddr* in the input packet by using this Tcl statement:

```
$request put giaddr "1.2.3.4"
```

To configure one routine entry for multiple extension points and to alter its behavior depending on the extension point from which the server calls it, the DHCP server passes the ASCII name of the extension point in the environment dictionary under the key **extension-point**.

For some sample Tcl extensions, see the Cisco Prime Network Registrar directory; by default:

- **Linux**—/opt/nwreg2/local/examples/dhcp/tcl
- **Windows**—\Program Files\Cisco Prime Network Registrar\examples\dhcp\tcl

Dealing with Tcl errors

You generate a Tcl error if you:

- Reference a dictionary that is not available.
- Reference a dictionary data item that is not available.
- Request a put operation on an invalid data item, for example, an invalid IP address.

In these cases, the extension immediately fails unless you surround the statement with a catch error statement:

```
catch { $request put giaddr "1.2.3.a" } error
```

Dealing with Tcl errors

You generate a Tcl error if you:

- Reference a dictionary that is not available.
- Reference a dictionary data item that is not available.
- Request a put operation on an invalid data item, for example, an invalid IP address.

In these cases, the extension immediately fails unless you surround the statement with a catch error statement:

```
catch { $request put giaddr "1.2.3.a" } error
```

Configuring Tcl Extensions

To configure a Tcl extension, write it and place it in the extensions directory. For UNIX and Linux, this is the `/opt/nwreg2/local/extensions/dhcp/tcl` directory. For Windows, this is the `\Program Files\Cisco Prime Network Registrar\extensions\dhcp\tcl` directory.

When the DHCP server configures an extension during startup, it reads the Tcl source file into an interpreter. Any syntax errors in the source file that would render Tcl interpreter unable to load the file would also fail the extension. Typically, the DHCP server generates an error traceback in the log file from Tcl to help you to find the error.

Handling Boolean Variables in Tcl

In the environment dictionary, the Boolean variables are string-valued and have a value of **true** or **false**. The DHCP server expects an extension to set the value to **true** or **false**. However, in the request or response dictionaries, Boolean values are single-byte numeric format, and **true** is **1** and **false** is **0**. While more efficient for the C/C++ extensions, this approach does make the Tcl API a bit more complex.

Init-Entry Extension Point in Tcl

Tcl extensions support the **init-entry** extension point (see [init-entry](#), on page 379), and the arguments supplied in the *init-args* parameter to the command line appear in the environment dictionary associated with the key **arguments**.

Multiple Tcl interpreters can run in the DHCP server, for performance purposes, each in its own Tcl context. The server calls the Tcl extension once at the **init-entry** point for every Tcl context (interpreter) it runs. Ensure that your Tcl extension **init-entry** is robust, given multiple calls.

Information cannot flow between the Tcl contexts, but the **init-entry** can initialize global Tcl variables in each Tcl interpreter that any Tcl extension can access, regardless of the interpreter.

Note that all Tcl extensions share the Tcl interpreters. If your Tcl extension initializes global variables or defines procedures, ensure that these do not conflict with some other Tcl extension global variables or procedure names.

C/C++ Extensions

All DHCP C/C++ extensions are **dex** extensions, short for DHCP Extension.

Related Topics

- [C/C++ API](#), on page 360
- [Using Types in C/C++](#), on page 360
- [Building C/C++ Extensions](#), on page 360
- [Using Thread-Safe Extensions in C/C++](#), on page 360
- [Configuring C/C++ Extensions](#), on page 361
- [Debugging C/C++ Extensions](#), on page 361

C/C++ API

The routine signature for both the **entry** and **init-entry** routines for the C/C++ API is:

```
typedef int (DEXAPI * DexEntryPointFunction) (
int iExtensionPoint,
dex_AttributeDictionary_t* pRequest,
dex_AttributeDictionary_t* pResponse,
dex_EnvironmentDictionary_t* pEnviron );
```

Along with pointers to three structures, the integer value of the extension point is one of the parameters of each routine.

The C/C++ API is specifically constructed so that you do not have to link your shared library with any Cisco Prime Network Registrar DHCP server files. You configure the entry to your routine when you configure the extension. The necessary call-back information for the operations to perform on the request, response, and environment dictionaries is in the structures that comprise the three dictionary parameters passed to your extension routine.

The DHCP server returns all binary information in network order, which is not necessarily properly aligned for the executing architecture.

Using Types in C/C++

Many C/C++ routines are available that use types, for example, **getByType()**. These routines are designed for use in performance-sensitive environments. The reasoning behind these routines is that the extension can acquire pointers to types once, for example, in the **init-entry** point, and thereafter use the pointers instead of string-valued names when calling the routines of the C/C++ API. Using types in this manner removes one hash table lookup from the extension processing flow of execution, which should improve (at least fractionally) the performance of any extension.

Building C/C++ Extensions

The directories `/opt/nwreg2/local/examples/dhcp/dex` (UNIX and Linux) and `\Program Files\Cisco Prime Network Registrar\examples\dhcp\dex` (Windows) contain sample C/C++ extension code, as well as a short makefile designed to build the sample extensions. To build your own extensions, you need to modify this file. It has sections for Microsoft Visual C++ and GNU C++. Simply move the comment lines to configure the file for your environment.

Your extension needs to reference the include file `dex.h`. This file contains the information your program needs to use the C/C++ API. When building C/C++ extensions on Windows, remember to add your entry points to the `.def` file.

After you build the `.dll` (Windows) or `.so` (UNIX) file (all dex extensions are shared libraries), you need to move them into the `/opt/nwreg2/local/extensions/dhcp/dex` directory (UNIX), or the `\Program Files\Cisco Prime Network Registrar\extensions\dhcp\dex` directory (Windows). You can then configure them.

Using Thread-Safe Extensions in C/C++

The DHCP server is multithreaded, so any C/C++ extensions written for it must be thread-safe. Multiple threads, and possibly multiple processors, must be capable of calling these extensions simultaneously at the

same entry point. You should have considerable experience writing code for a multithreaded environment before designing C/C++ extensions for Cisco Prime Network Registrar.


Caution

All C/C++ extensions must be thread-safe. If not, the DHCP server will not operate correctly and will fail in ways that are extremely difficult to diagnose. All libraries and library routines that these extensions use must also be thread-safe.

On several operating systems, you must ensure that the runtime functions used are really thread-safe. Check the documentation for each function. Special thread-safe versions are provided (often *functionname_r*) on several operating systems. Because Windows provides different versions of libraries for multithreaded applications that are thread-safe, this problem usually does not apply.

Be aware that if any thread makes a non-thread-safe call, it affects any of the threads that make up the safe or locked version of the call. This can cause memory corruptions, server failures, and so on.

Diagnosing these problems is extremely difficult, because the cause of these failures are rarely apparent. To cause a server failure, you need very high server loads or multiprocessor machines with many processes. You might need running times of several days. Often, problems in extension implementation might not appear until after sustained periods of heavy load.

Because some runtime or third-party libraries might make non-thread-safe calls that you cannot detect, check your executables as to what externals are being linked (**nm** on UNIX).

If the routines of a library call the routines without the *_r* suffixes, displayed in the following table, the library is not thread-safe, and you cannot use it. The interfaces to the thread-safe versions of these library routines can vary based on operating system.

asctime_r	getgrid_r	getnetent_r	getrpcbynumber_r	lgamma_r
ctermid_r	getgrnam_r	getprotobyname_r	getrpcent_r	localtime_r
ctime_r	gethostbyaddr_r	getprotobyname_r	getservbyname_r	nis_sperror_r
fgetgrent_r	gethostbyname_r	getprotoent_r	getservbyport_r	rand_r
fgetpwent_r	gethostent_r	getpwnam_r	getservent_r	readdir_r
fgetspent_r	getlogin_r	getpwent_r	getspent_r	strtok_r
gamma_r	getnetbyaddr_r	getpwuid_r	getspnam_r	tmpnam_r
getgrent_r	getnetbyname_r	getrpcbyname_r	gmtime_r	ttyname_r

Configuring C/C++ Extensions

Because the .dll and .so files are active when the server is running, it is not a good idea to overwrite them. After you stop the server, you can overwrite the .dll and .so files with newer versions.

Debugging C/C++ Extensions

Because your C/C++ shared library runs in the same address space as the DHCP server, and receives pointers to information in the DHCP server, any bugs in your C/C++ extension can very easily corrupt the DHCP

server memory, leading to a server failure. For this reason, use extreme care when writing and testing a C/C++ extension. Frequently, you should try the approach to an extension with a Tcl extension and then code the extension in C/C++ for increased performance.

Related Topics

[Pointers into DHCP Server Memory in C/C++, on page 362](#)

[Init-Entry Entry Point in C/C++, on page 362](#)

Pointers into DHCP Server Memory in C/C++

The C/C++ extension interface routines return pointers into DHCP server memory in two formats:

- char* pointer to a series of bytes.
- Pointer to a structure called an `abytes_t`, which provides a pointer to a series of bytes with an associated length (defined in `dex.h`).

In both cases, the pointers into DHCP server memory are valid while the extension runs at that extension point. They are also valid for the rest of the extension points in the series processing this request. Thus, an `abytes_t` pointer returned in the **post-packet-decode** extension point is still valid in the **post-send-packet** extension point.

The pointers are valid for as long as the information placed in the environment dictionary is valid. However, there is one exception. One C/C++ routine, **getType**, returns a pointer to an `abytes_t` that references a type. These pointers are valid through the entire life of the extension. Typically, the server would call this routine in the **init-entry** extension point and save the pointers to the `abytes_t` structures that define the types in the static data of the shared library. Pointers to `abytes_t` structures returned by **getType** are valid from the **init-entry** call for initialization until the call for uninitialization.

Init-Entry Entry Point in C/C++

The DHCP server calls the **init-entry** extension point (see [init-entry, on page 379](#)) once when configuring each extension and once when unconfiguring it. The `dex.h` file defines two extension point values passed as the extension points for the configure and unconfigure calls: `DEX_INITIALIZE` for configure and `DEX_UNINITIALIZE` for unconfigure. The environment dictionary value of the extension-point data item is **initialize** or **uninitialize** in each call.

When calling the **init-entry** extension point for **initialize**, if the environment dictionary data item **persistent** contains the value **true**, you can save and use the environment dictionary pointer any time before the return from the **uninitialize** call. In this way, background threads can use the environment dictionary pointer to log messages in the server log file. Note that you must interlock all access to the dictionary to ensure that at most one thread processes a call to the dictionary at a time. You can use the saved dictionary pointer up to when the extension returns from the **uninitialize** call. This way, the background threads can log messages during termination.

DHCP Request Processing Using Extensions

The Cisco Prime Network Registrar DHCP server has extension points to which you can attach your own extensions. They have descriptive names that indicate where in the processing flow of control to use them.

Because the extension points are tied to the processing of input requests from DHCP clients, it is helpful to understand how the DHCP server handles requests. Request processing comes in three general stages:

1. Initial request processing (see [Table 43: Initial Request Processing Using Extensions](#))
2. DHCPv4 or DHCPv6 processing (see [Table 44: DHCPv4 or DHCPv6 Request Processing Using Extensions](#))
3. Final response processing (see [Table 45: Final Response Processing Using Extensions](#))

Table 43: Initial Request Processing Using Extensions

Client Request Processing Stage	Extension Point Used
1. Receive a packet from a DHCP client.	pre-packet-decode
2. Decode the packet.	post-packet-decode
3. Determines the client-classes.	
4. Modifies the client-class.	post-class-lookup
5. Processes the client-classes, looking up clients.	pre-client-lookup post-client-lookup
6. Build a response container from the request.	

Table 44: DHCPv4 or DHCPv6 Request Processing Using Extensions

Client Request Processing Stage	Extension Point Used
1. In DHCPv4, find a lease already associated with this client, if any, or locate a new lease for the client.	
2. Serialize all requests associated with this client (processing continues when the request reaches the head of the serialization queue).	
3. In DHCPv6, process the client request, generating leases if necessary. The server tries to provide the client with at least one preferred lease for each usable prefix per binding. You can generate leases and change lease states multiple times for a client request, but not for reserved leases.	generate-lease and lease-state-change (multiple calls are possible for both in DHCPv6)
4. Determine if the lease is (still) acceptable for this client (can occur multiple times in DHCPv6).	check-lease-acceptable
5. Initiate DNS Update operations as necessary (can occur multiple times in DHCPv6).	

Table 45: Final Response Processing Using Extensions

Client Response Processing Stage	Extension Point Used
1. Gather all the data to include in the response packet.	
2. Write to the lease database.	
3. Prepare the response packet for encoding.	pre-packet-encode

Client Response Processing Stage	Extension Point Used
4. Encode the response packet for transmission to the client.	post-packet-encode
5. Send the packet to the client.	post-send-packet
6. Release all context for the client and request.	environment-destroyer

These steps and additional opportunities for using extensions are explained in the following sections. The extension points are indicated in **bold**.

Related Topics

- [Enabling DHCPv6 Extensions, on page 364](#)
- [Receiving Packets, on page 365](#)
- [Decoding Packets, on page 365](#)
- [Determining Client-Classes, on page 365](#)
- [Modifying Client-Classes, on page 365](#)
- [Processing Client-Classes, on page 366](#)
- [Building Response Containers, on page 366](#)
- [Determining Networks and Links, on page 366](#)
- [Finding Leases, on page 367](#)
- [Serializing Lease Requests, on page 367](#)
- [Determining Lease Acceptability, on page 368](#)
- [DHCPv6 Leasing, on page 369](#)
- [Gathering Response Packet Data, on page 370](#)
- [Encoding Response Packets, on page 371](#)
- [Updating Stable Storage, on page 371](#)
- [Sending Packets, on page 371](#)
- [Processing DNS Requests, on page 371](#)
- [Tracing Lease State Changes, on page 371](#)
- [Controlling Active Leasequery Notifications, on page 372](#)

Enabling DHCPv6 Extensions

By default, extensions are assumed to support only DHCPv4. To write DHCPv6 extensions, you must implement an **init-entry** extension point that must:

1. Set the *dhcp-support* environment data item to **v4** (for DHCPv4 only, the preset value), **v6** (for DHCPv6 only), or **v4,v6** (for DHCPv4 and DHCPv6). This data item indicates to the server what the extension is willing to support.

2. Set the *extension-extensionapi-version* environment data item to **2**. (The *dhcp-support* data item is ignored unless the *extension-extension-api-version* is set to **2**.)

You might need to write separate extensions for DHCPv4 and DHCPv6, because of the differences in packet formats, DHCP protocol, and internal server data. However, the fundamentals of both kinds of extensions are very much the same.

The server calls these extension points at essentially the same places during processing, although it can call some DHCPv6 extension points multiple times due to the possibility of multiple lease requests per client.

Receiving Packets

The DHCP server receives DHCPv4 packets on port 67 and DHCPv6 packets on port 547 (the DHCP input ports) and queues them for processing. It attempts to empty the UDP input queue as quickly as possible and keeps all of the requests that it receives on an internal list for processing as soon as a free thread is available to process them. You can configure the length of this queue, and it will not grow beyond its maximum configured length.

Decoding Packets

When a free thread is available, the DHCP server allocates to it the task of processing an input request. The first action it takes is to decode the input packet to determine if it is a valid DHCP client packet. As part of this decoding process, the DHCP server checks all options to see if they are valid—if the lengths of the options make sense in the overall context of the request packet. It also checks all data in the DHCP request packet, but takes no action on any data in the packet at this stage.

Use the **pre-packet-decode** extension point to rewrite the input packet. After the DHCP server passes this extension point, it stores all information from the packet in several internal data structures to make subsequent processing more efficient.

Determining Client-Classes

If you configure an expression in the *client-class-lookup-id*, it is at this stage that the DHCP server evaluates the expression (see [Using Expressions, on page 315](#) for a description of expressions). The result of the expression is either <null>, or something converted to a string. The value of the string must be either a client-class name or <none>. In the case of <none>, the server continues to process the packet in the same way as if there were no *client-class-lookup-id* configured. In the case of a <null> response or an error evaluating the *client-class-lookup-id*, the server logs an error message and drops the packet (unless an extension configured at the **post-class-lookup** extension point specifically instructs the server not to drop the packet). As part of the process of setting the client-class, the DHCP server evaluates any *limitation-id* configured for that client-class and stores it with the request.

Modifying Client-Classes

After the DHCP server evaluates the *client-class-lookup-id* and sets the client-class, it calls any extension attached to the **post-class-lookup** extension point. You can use that extension to change any data that the client-class caused to become associated with the request, including the *limitation-id*. The extension also learns if the evaluation of the *client-class-lookup-id* dropped the packet. The extension not only finds out if it needs to drop the packet, it instructs the server not to drop the packet if it wants the server not to do so.

Also, an extension running at the **post-class-lookup** extension point can set a new client-class for the request, and uses the data from that client-class instead of the current one. This is the only extension point where setting the client-class actually uses that client-class for the request.

Processing Client-Classes

If you enabled client-class processing, the DHCP server performs it at this stage.

Use the **pre-client-lookup** extension point to affect the client to look up, possibly by preventing the lookup or supplying data that overrides the existing data. After the DHCP server passes the **pre-client-lookup** extension point, it looks up the client (unless the extension specifically prevents it) in the local database or in an LDAP database, if one was configured.

After the server looks up the client, it uses the data in the client entry to fill in additional internal data structures. The DHCP server uses data in the specified client-class entry to complete any data that the client entry does not specify. When the DHCP server retrieves all the data stored in the various places in the internal data structures for additional processing, it runs the next extension point.

Use the **post-client-lookup** extension point to review the operation of the client-class lookup process, such as examining the internal server data structures filled in from the client-class processing. You can also use the extension point to change any data before the DHCP server does additional processing.

Building Response Containers

At this stage, the DHCP server determines the request type and builds an appropriate response container based on the input. For example, if the request is a DHCPDISCOVER, the server creates a DHCPOFFER response to perform the processing. If the input request is a BOOTP request, the server creates a BOOTP response to perform the response processing.

For DHCPv6, a server creates an ADVERTISE or REPLY packet, depending on the request.

Determining Networks and Links

The DHCP server must determine the subnet from which every request originated and map that into a set of address pools, scopes, prefixes, or links that contain IP addresses.

For DHCPv4, internal to the DHCP server is the concept of a network, which, in this context, refers to a LAN segment or physical network. In the DHCP server, every scope or prefix belongs to a single network.

Some scopes or prefixes are grouped together on the same network because their network numbers and subnet masks are identical. Others are grouped because they are related through the primary-scope or prefix pointer.

The Cisco Prime Network Registrar DHCP server determines the network to use to process a DHCP client request in the following sequence:

1. Determining the source address, either the *giaddr* or, if the *giaddr* is zero, the address of the interface on which the request arrived.
2. Using this address to search for any scope or prefix that was configured in the server that is on the same subnet as this address. If the server does not find a scope or prefix, it drops the request.
3. After finding the scope or prefix, using its network in subsequent processing.

For DHCPv6 processing, see [Determining Links and Prefixes, on page 127](#).

Finding Leases

For DHCPv4, now that when the DHCP server establishes the network, it searches the hash table held at the network level to see if the network already knows the *client-id*. “Already knows,” in this context, means that this client previously received an offer or a lease on this network, and the lease was not offered to or leased by a different client since that time. Thus, a current lease or an available expired lease appears in the network level hash table. If the DHCP server finds a lease, it proceeds to the next step, which is to serialize all requests for the same IP address.

If the DHCP server does not find a lease, and if this is a BOOTP or DHCPDISCOVER request, the server looks for a reserved lease from a scope or prefix in the network.

If it finds a reserved lease, the server checks whether the scope or prefix and lease are both acceptable. The following must be true regarding the reserved lease and the scope or prefix that contains it:

- The lease must be available (not leased to another DHCP client).
- The scope or prefix must support the request type (BOOTP or DHCP).
- The scope or prefix must not be in a deactivated state.
- The lease must not be in a deactivated state.
- The selection tags must contain all of the client *selection-criteria* and none of the client *selection-criteria-excluded*.
- The scope or prefix must not be in a renew-only state.

If the reserved lease is acceptable, the server proceeds to the next step, which is to serialize all requests for the IP address. Having failed to find an existing or reserved lease for this client, the server now attempts to find any available IP addresses for this client.

The general process the DHCP server uses is to scan all of the scopes or prefixes associated with this network in round-robin order, looking for one that is acceptable for the client and also has available addresses. An acceptable scope or prefix has the following characteristics:

- If the client has *selection-criteria* associated with it, the selection tags must contain all of the client inclusion criteria.
- If the client has *selection-criteria-excluded* associated with it, the selection tags must contain none of the client exclusion criteria.
- The scope or prefix must support the client request type—If the client request is a DHCPREQUEST, you must enable the scope or prefix for DHCP. Likewise, if the request is a BOOTP request, you must enable the scope or prefix for BOOTP and dynamic BOOTP.
- It must not be in a renew-only state.
- It must not be in deactivated state.
- It must have an available address.

If the server does not find an acceptable scope or prefix, it logs a message and drops the packet.

For DHCPv6 processing, see [Determining Links and Prefixes, on page 127](#).

Serializing Lease Requests

Because multiple DHCP requests can be in process simultaneously for one client and lease, you must serialize DHCPv4 requests at the lease level. The server queues them on the lease and processes them in the order of queueing.

For DHCPv6, the server serializes on the client (per link) and not on the lease.

Determining Lease Acceptability

For DHCPv4, the DHCP server now determines if the lease is (still) acceptable for the client. In the case where this is a newly acquired lease for a first-time client, it will be acceptable. However, in the case where the server processes a renewal for an existing lease, the acceptability criteria might have changed since the server granted the lease, and you need to check its acceptability again.

If the client has a reservation that is different from the current lease, the server first determines if the reserved lease is acceptable. The criteria for release acceptability are:

- The reserved lease must be available.
- The reserved lease must not be in a deactivated state.
- The scope or prefix must not be in a deactivated state.
- If the request is BOOTP, the scope or prefix must support BOOTP.
- If the request is DHCP, the scope or prefix must support DHCP.
- If the client has any *selection-criteria*, the selection tags must contain all of the client inclusion criteria.
- If the client has any *selection-criteria-excluded*, the selection tags must contain none of the client exclusion criteria.
- If the client previously associated with this lease is not the current client, the scope or prefix must not be in a renew-only state.

If the reserved lease meets all of these criteria, the DHCP server considers the current lease unacceptable. If there is no reserved lease for this client, or the reserved lease did not meet the criteria for acceptability, the DHCP server examines the current lease for acceptability.

The criteria for acceptability are:

- The lease must not be in a deactivated state.
- The scope or prefix must not be in a deactivated state.
- If the request is BOOTP, the scope or prefix must support BOOTP. If the request is DHCP, the scope or prefix must support DHCP.
- If the client does not have a reservation for this lease, and the request is BOOTP, the scope or prefix must support dynamic BOOTP.
- If the client does not have a reservation for this lease, no other client can either.
- If the client has any *selection-criteria*, the selection tags must contain all of the client inclusion criteria.
- If the client has any *selection-criteria-excluded*, the selection tags must contain none of the client exclusion criteria.
- If the client previously associated with this lease is not the current client, the scope or prefix must not be in a renew-only state.



Tip At this point in the DHCP server processing, you can use the **check-lease-acceptable** extension point. You can use it to change the results of the acceptability test. Do this only with extreme care.

Upon determining that a lease is unacceptable, the DHCP server takes different actions, depending on the particular DHCP request currently being processed.

- **DHCPDISCOVER**—The DHCP server releases the current lease and attempts to acquire a different, acceptable lease for this client.
- **DHCPCREQUEST SELECTING**—The DHCP server sends a NACK to the DHCP client because the lease is invalid. The client should then immediately issue a DISCOVER request to acquire a new DHCP OFFER.

- **DHCPRENEW, DHCPREBIND**—The DHCP server sends a NACK to the DHCP client to attempt to force the DHCP client into the INIT phase (attempt to force the DHCP client into issuing a DHCPDISCOVER request). The lease is still valid until the client actually issues the request.
- **BOOTP**—The DHCP server releases the current lease and attempts to acquire a different, acceptable lease for this client.

**Caution**

Take extreme care with the **check-lease-acceptable** extension point. If the answer the extension point returns does not match the acceptability checks in the search for an available lease performed in a DHCPDISCOVER or dynamic BOOTP request, an infinite server loop can result (either immediately or on the next DHCPDISCOVER or BOOTP request). In this case, the server would acquire a newly available lease, determine that it was not acceptable, try to acquire a newly available lease, and determine that it was not acceptable, in a continuous loop.

DHCPv6 Leasing

The DHCP server processes IPv6 lease requests by scanning the client request for IA_NA, IA_TA, and IA_PD options (see [DHCPv6 Bindings, on page 189](#)). For each of these options, the server considers any leases that the client explicitly requests. If the lease already exists for the client and binding (IA option and IAID), the server determines if the lease is still acceptable. For leases that the client requests that do not already exist for the client, the server tries to give that lease to the client if:

- Another client or binding is not already using the lease.
- The prefix for the lease has the client-request flag set in its *allocation-algorithms* attribute.
- The lease is usable and on a usable prefix (see the [DHCPv6 Prefix Usability, on page 369](#)).

Next, the server tries to assure that clients are using reservations and that a client has a usable lease with a nonzero preferred lifetime on each usable prefix on the link. Thus, the server processes each of these bindings as follows:

1. Adds any client reservations (not already in use) to the binding, provided the reservation flag is set in the prefix *allocation-algorithms* attribute. The server uses the first binding of the appropriate type for the reservation; that is, it uses address leases for IA_NA bindings and prefix leases for IA_PD bindings.
2. If the client has no lease with a nonzero preferred lifetime on each prefix that the client can use, the server tries to allocate a lease to the client. The prefix *allocation-algorithms* flags control how the server allocates the lease.

Related Topics

[DHCPv6 Prefix Usability, on page 369](#)

[DHCPv6 Lease Usability, on page 370](#)

[DHCPv6 Lease Allocation, on page 370](#)

DHCPv6 Prefix Usability

A usable prefix:

- Is not deactivated.
- Did not expire.
- Allows leases of the binding type.

- Matches the client selection criteria, if any.
- Does not match the client selection exclusion criteria, if any.

DHCPv6 Lease Usability

A usable lease is:

- Not unavailable.
- Not revoked.
- Not deactivated.
- Not reserved for a different client.
- Not subject to *inhibit-all-renews* or *inhibit-renews-at-reboot*.
- Renewable if being renewed (IA_TA leases are not renewable).
- Leasable with a nonzero valid lifetime.

DHCPv6 Lease Allocation

When the server needs to allocate a new lease on a prefix, it calls any extensions registered at the **generate-lease** extension point if the prefix extension flag is set in the *allocation-algorithms* attribute. (See [generate-lease, on page 386](#).) The extensions can supply the address (IA_NA or IA_TA binding) or prefix (IA_PD binding) to be assigned, request that the server use its normal allocation algorithm (if enabled in *allocation-algorithms*), or request the server to skip assigning a lease on this prefix. The server might call the extension again if it supplied an address or prefix that is not valid or is already in use.

If extensions are not allowed, there are no extensions registered, or the extension requests the normal allocation algorithm of the server, the server allocates a randomly generated address or finds the first best-fit available prefix (as controlled by the prefix *allocation-algorithms* attribute) and creates the lease.

Once the server has a lease and does an acceptability check on it (see [DHCPv6 Lease Usability, on page 370](#)), the server calls any extensions registered at the **check-lease-acceptable** extension point to allow the extension to alter the acceptability of the lease. (See [check-lease-acceptable, on page 387](#).) You would typically only use this extension point to change an acceptable result into a unacceptable one; however, the server allows a unacceptable result to be changed to an acceptable one, although this is strongly discouraged because of possibly adverse consequences. If the lease is not acceptable, the server likely tries to allocate another lease; thus, use care to avoid an infinite loop. In some cases, you might need the **check-lease-acceptable** and **generate-lease** extension points for full control of the leases a client gets: **generate-lease** can request the server to skip allocation of the lease.

The server calls the **check-lease-acceptable** extension point for each client request for each lease.

Gathering Response Packet Data

In this stage of processing, the DHCP server collects all the data to send back in the DHCP response and determines the address and port to which to send the response. You can use the **pre-packet-encode** extension point to change the data sent back to the DHCP client in the response, or to change the address to which to send the DHCP response. (See [pre-packet-encode, on page 388](#).)



Caution

Any packets dropped at the **pre-packet-encode** extension point, whether they be DHCP or BOOTP packets, still show the address to be leased in the Cisco Prime Network Registrar lease state database, for as long as the remaining lease time. Because of this, it is advisable to drop packets at an earlier point.

Encoding Response Packets

In this stage, the DHCP encodes the information in the response data structure into a network packet. If this DHCP client requires DNS activity, the DHCP server queues a DNS work request to the DNS processing subsystem in the DHCP server. That request runs whenever it can, but generally not before sending the packet to the client. (See [pre-packet-encode](#), on page 388.)

Updating Stable Storage

At this stage, the DHCP server ensures that the on-disk copy of the information is up to date with respect to the IP address before proceeding. For DHCPv6, this can involve multiple leases.

Sending Packets

Use the **post-send-packet** extension point (see [post-send-packet](#), on page 389) for any processing that you want to perform outside of the serious time constraints of the DHCP request-response cycle. After the server sends the packet to the client, it calls this extension point.

Processing DNS Requests

Here is a simplified view of what the DHCP server does to add names to DNS:

1. **Builds up a name to use for the A record**—The DHCP server creates the name that it will use in the forward (A record) DNS request. For DHCPv6, these are AAAA records. The DNS name comes from a variety of sources including the client-requested-host-name and client-domain-name data items, which are usually populated from options in the DHCP request, and the DNS update configuration (including the host-name-generator/v6-host-name-generator expressions).
2. **Tries to add the name, asserting that none exists yet**—At this stage, the prerequisites for the DNS name update request indicate that the name should not exist. If it succeeds, the DHCP server proceeds to update the reverse record.
3. **Tries to add the name, asserting that the server should supply it**—The server tries to add the hostname, asserting that the host exists and has the same TXT record (or DHCID record for DHCPv6) as the one that was sent.
 - If this succeeds, the server proceeds to the next step.
 - If it fails, the server checks if it exhausted its naming retries, in which case it exits and logs an error.
 - If it did not exhaust its naming entries, it returns to the first step, which is to build up a name for the A record.

For DHCPv6, the server uses DHCID records instead of TXT records. Also, DHCPv6 clients can have multiple leases and the forward zones can be the same or potentially different.

4. **Updates the reverse record**—Now that the DHCP server knows which name to associate with the reverse (PTR) record, it can update the reverse record with no prerequisites, because it can assume it is the owner of the record. If the update fails, the DHCP server logs an error.

Tracing Lease State Changes

The server calls the **lease-state-change** extension point whenever (and only when) a lease changes state. The existing state is in the response dictionary **lease-state** data item. The new state is in the environment dictionary

under **new-state**. The **new-state** never equals the existing state (if it did, the server would not call the extension). You should consider this extension to be read-only and not make modifications to any dictionary items, because the server calls it in many different places. Use this extension point only for tracking changes to a lease state.

Controlling Active Leasequery Notifications

The server determines whether a lease is queued for active leasequery notifications based on the *leasequery-send-all* attribute of *dhcp-listener*. If this attribute is enabled, the DHCP server always sends a notification to an active leasequery client. If disabled or unset, the DHCP server only sends notifications which are necessary to maintain accurate state in the active leasequery client.

To allow customer written extensions to control the sending of a lease (such as only on specific state changes), a new data item, *active-leasequery-control*, has been added to both the request and response dictionaries. These data items have three values:

- 0—unspecified (the server determines whether to send the notification)
- 1—send (the server will send the notification)
- 2—do not send (the server will not send the notification)

The *active-leasequery-control* data item is initialized as 0, unspecified.



Note These data items may be written and read, but the value that is read is only the value that might have been previously written.

These data items can force the DHCP server to take specific actions after being written, but reading them without previously writing them will always return 0, unspecified. These data items will not let you determine the choices that the DHCP server makes when it comes to deciding whether or not to send a message to an active leasequery client concerning the changes (if any) made to a lease that is being processed. Thus, these data items are technically read/write, but reading them will only allow you to determine what you may have previously written into them.

These data items are examined (the response dictionary is examined first, then the request) when the lease is written to the internal lease state database as that is when the lease is also queued for active leasequery notification. This occurs after the *check-lease-acceptable* and *lease-state-change* extensions points, but prior to the *pre-packet-encode* extension point. Therefore, any changes made to these attributes at or after the *pre-packet-encode* extension point will be ignored.

Whether a lease is queued for active leasequery notification is determined as follows:

Response's <i>active-leasequery-control</i>	Request's <i>active-leasequery-control</i>	<i>Leasequery-send-all</i>	Action
0—unspecified	0—unspecified	False or unset	Conditional (see <i>leasequery-send-all</i> attribute description)
0—unspecified	0—unspecified	True	Sent
0—unspecified	1— send	Ignored	Sent
0—unspecified	2—don't send	Ignored	Not Sent

Response's active-leasequery-control	Request's active-leasequery-control	Leasequery-send-all	Action
1— send	Ignored	Ignored	Sent
2— don't send	Ignored	Ignored	Not Sent



Note The *active-leasequery-control* of response and request is examined prior to any examination of the *leasequery-send-all* attribute.

If either of these dictionary data items has a value other than unspecified, that value will override any value configured in the *leasequery-send-all* attribute of the dhcp listener.



Note You cannot control the sending of active leasequery information by writing a single extension that runs only at the *lease-state-change* extension point, because that extension point is only called when there is a change in state of a lease.

Lease state changes may not occur when you might expect them to. For example, if a lease is leased, and that same client goes through a DISCOVER/OFFER/REQUEST/ACK cycle, the *lease-state-change* extension point is not called since the lease does not actually go through a state change internally and it remains leased throughout the cycle. Thus, to gain absolute control over the transmission of information to active leasequery clients, you have to initialize the *active-leasequery-control* attribute in request processing, and then possibly alter it or override it by operating on the response dictionary value at the *lease-state-change* extension point.

Extension Dictionaries

Every extension is a routine with three arguments. These arguments represent the request dictionary, response dictionary, and environment dictionary. Not every dictionary is available to every extension. The following table shows the extensions points and the dictionaries that are available to them.

Table 46: Extension Points and Relevant Dictionaries

Extension Point	Dictionary
init-entry	Environment
pre-packet-decode	Request, Environment
post-packet-decode	Request, Environment
pre-client-lookup	Request, Environment
pre-dns-add-forward	Environment (Deprecated extension point)
post-client-lookup	Request, Environment
post-class-lookup	Request, Environment

Extension Point	Dictionary
generate-lease	Request, Response, Environment
lease-state-change	Response, Environment
check-lease-acceptable	Request, Response, Environment
pre-packet-encode	Request, Response, Environment
post-packet-encode	Request, Response, Environment
post-send-packet	Request, Response, Environment
environment-destructor	Environment



Note When the server sends DHCPv6 Reconfigure messages, it can call the **pre-packet-encode**, **post-packet-encode**, and **post-send-packet** extension points without a request.

For the request and response dictionaries, you can use the **isValid** method to probe if the dictionary is available for an extension point.

Each of the three dictionaries consists of name-value pairs. The environment dictionary, which is available to every extension point, is the simplest dictionary. The request and response dictionaries are more complex and their data is typed. Thus, when you set a value in one of these dictionaries, you need to match the data type to the value. You can use the dictionaries for getting, putting, and removing values.

Related Topics

[Environment Dictionary, on page 374](#)

[Request and Response Dictionaries, on page 376](#)

Environment Dictionary

The environment dictionary is available at all extension points. It is strictly a set of name-value pairs in which both the name and the value are strings.

The DHCP server uses the environment dictionary to communicate with extensions in different ways at different extension points. At some extension points, the server places information in the environment dictionary for the extension to modify. In others, the extension can place values in the environment dictionary to control the flow or data after the extension finishes its processing.

The environment dictionary is unique in that an extension can put any name-value pair in it. Although you do not get an error for using undocumented name-value pairs, the server does not recognize them. These name-value pairs can be useful for your extension points to communicate data among them.

The DHCP server creates the environment dictionary when a DHCP request arrives and the dictionary remains with that request through the processing. Thus, an extension that runs at the **post-packet-decode** extension point can put data into the environment dictionary, and then an extension run at the **pre-packet-encode** extension point might read that data from the dictionary.



Note The **init-entry** extension point has a unique environment dictionary.

Related Topics

[General Environment Dictionary Data Items, on page 375](#)

[Initial Environment Dictionary, on page 376](#)

General Environment Dictionary Data Items

The data items in the following table are valid in the environment dictionary at all extension points. (See the individual extension point sections for environment dictionary data items specific to each one.)

The data items are input, output, or both:

- Input—The DHCP server sets the value and inputs it to the extension.
- Output—The value is output to the DHCP server, which reads it, and acts upon it.

Table 47: General Environment Dictionary Data Items

Environment Data Item	Description
<i>drop</i> (input/output)	If the <i>drop</i> value is equal to the string true when the extension exits, the DHCP server drops the input packet and logs a message in the log file. Initially set to false . Available at most extension points, but not all (such as generate-lease). Note For recommendations on how to use <i>drop</i> for multiple extensions per extension point, see Multiple Extension Considerations, on page 357 .
<i>extension-name</i> (input)	Name with which the extension was configured. You can configure the same piece of code as several different extensions and at several different extension points. This allows one piece of code to do different things, depending on how you configure it. The code can also use this string to find itself in the extension-name-sequence string, for which it needs to know its own name.
<i>extension-name-sequence</i> (input)	Provides a comma-separated string representing the configured extensions for this extension point. It allows an extension to determine which extensions can run before and after it. The <i>extension-name</i> data item provides the currently running extension. For example, if you configure tlcfirst as the first extension and dexscript as the fifth, the <i>extension-name-sequence</i> would contain " tlcfirst,,,dexscript ".
<i>extension-point</i> (input)	Name of the extension point. For example, post-packet-decode .
<i>extension-sequence</i> (input)	String that is the sequence number of the extension at the extension point.

Environment Data Item	Description
<i>log-drop-message</i> (input/output)	If the <i>drop</i> value is equal to the string true , and the <i>log-drop-message</i> value is equal to the string false when the extension exits, then the DHCP server drops the input packet, but does not log a message in the log file. Does not apply to init-entry .
<i>trace-level</i> (output)	Setting this to a number makes that number the current setting of the <i>extension-trace-level</i> server attribute for all extensions processing this request.
<i>user-defined-data</i> (input/output)	Set with the <i>user-defined-data</i> attribute of a lease stored with the lease before request processing. You can have it written to disk before (but not with) a pre-packet-encode . If set to null, the server ignores the <i>user-defined-data</i> from the lease. You cannot remove a previous value by using a null string value. Appropriate for responses only. When the server writes the <i>user-defined-data</i> to a lease, the read-only <i>client-user-defined-data</i> response dictionary data item assumes its value. Note Be careful in using this data item in multiple extensions for an extension point. See Multiple Extension Considerations, on page 357 .

Initial Environment Dictionary

You can configure an extension with *init-args* and **init-entry**. Alternatively, you can specify configuration information for an extension to read out of the environment dictionary. You can set the DHCP property *initial-environment-dictionary* with a series of attribute-value pairs, and each pair is available in every environment dictionary. Using this capability, you can specify a variety of configuration and customizing information. Any extension can simply read this data directly out of the environment dictionary, without having to store it in some static data area, as is required with the *init-args* or **init-entry** approach.

You can read the values defined using the *initial-environment-dictionary* approach from any environment dictionary. You can also define new values for any attributes that appear in the *initial-environment-dictionary*. These new values are then available for the life of that environment dictionary (usually the life of the request packet being processed). However, this does not change the contents of any other environment dictionary. Any new environment dictionary (associated with a different request) sees the attribute-value pairs defined by the *initial-environment-dictionary* property of the DHCP server.

In addition, these *initial-environment-dictionary* attribute-value pairs do not appear in an enumeration of the values of the environment dictionary. They are only available if you request an attribute value not currently defined in the environment dictionary. The attribute-value pairs do not actually appear in the environment dictionary. Thus, if you define a new value for one of the attributes, that new value does appear in the environment dictionary. If you later delete the value, the original one is again available if you should request it.

Request and Response Dictionaries

The request and response dictionaries have a fixed set of accessible names. However, you cannot access all names from every extension point. These dictionaries make internal server data structures available to the extension for read-write or, in some cases, read-only access. Each data item has a particular data type. If you

omit the correct data type (for C/C++ extensions) on a put operation, or if the DHCP server cannot convert it to the correct data type (for Tcl extensions), the extension will fail.

The request dictionary is available at the beginning of the processing of a request. After the server creates a response, both the request and response dictionaries are available. It is an error to access a response dictionary before it is available.

In general, you cannot use an extension to change information data in the server. In some cases, however, you can use an extension to change configured data, but only for the duration of the processing for just that single request.

Appendix C contains details on the options and data items available for the received client request (the Request Dictionary) and for the response sent (the Response dictionary).

Related Topics

[Decoded DHCP Packet Data Items, on page 377](#)

[Using Parameter List Option, on page 378](#)

Decoded DHCP Packet Data Items

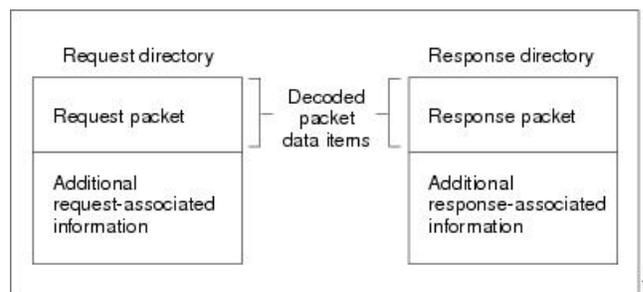
The DHCP protocol is a request-response UDP-based protocol and, thus, the stimulation for a DHCP server operation is usually a DHCP request from a client. The result is usually a DHCP response sent back to that client.

The DHCP extension facility makes the information input in the DHCP request available to extensions at most of the extension points, and the information to be sent as a response to a DHCP request available at the **pre-packet-encode** extension point (see [pre-packet-encode, on page 388](#)).

In addition to this DHCP packet-based information, there is additional data that the DHCP server uses when processing DHCP requests. This data becomes associated with either the DHCP request or the DHCP response as part of the architecture of the server. Much of this data is also made available to extensions, and much of it can be both read and written—in many cases altering the processing algorithms of the DHCP server.

The request and response dictionaries, therefore, contain two classes of data in each dictionary. They contain decoded packet data items as well as other request or response associated data items. The decoded packet data items are those data items directly contained in or derived from the DHCP request or DHCP response. Access to the decoded packet data items allows you to read and, in some cases, rewrite the DHCP request and DHCP response packet. The following figure shows the relationship between the request and the response dictionaries.

Figure 14: Extensions Request and Response Dictionaries



You can access information from the DHCP request packet, such as the *giaddr*, *ciaddr*, and all the incoming DHCP options by using the decoded packet data items in the request dictionary. Similarly, you can set the *giaddr* and *ciaddr*, and add and remove DHCP options in the outgoing DHCP response by accessing the decoded packet data items in the response dictionary.

It is important to realize that access to the packet information provided by the decoded packet data items is not all available to you. The specific data items available to that extension point are listed in the description of each extension point. Because the decoded packet data items are always accessible as a group, they are listed as a group.

You access DHCP options by name. If the option is not present, the server returns no data for that option. If you place an option into the decoded request or decoded response, it replaces any option with the same name already in the decoded request or decoded response, unless, in the put operation, you want the data specifically appended to existing data.

Some DHCP options can have multiple values. For example, the routers option can have one or more IP addresses associated with it. Access to these multiple values is by indexed operations on the option name.



Tip A **clear** operation on the request or response dictionary removes all the options in the decoded packet.

Using Parameter List Option

There is one option, *dhcp-parameter-request-list*, that the DHCP server specially handles in two ways, available as either a:

- Multiple-valued option of bytes under the name *dhcp-parameter-request-list*.
- Blob (sequence of bytes) option under the name *dhcp-parameter-request-list-blob*.

You can get or put the option using either name. The DHCP server handles the *dhcp-parameter-request-list* (and its *-blob* variant as well) differently in the response dictionary than in the request dictionary. When you access this option in the request dictionary, it is just another DHCP option in the request dictionary. In the response dictionary, however, special processing takes place.

You can use the *dhcp-parameter-request-list* option in the response dictionary to control the order of the options returned to the DHCP or BOOTP client. When you put the option in the response dictionary, the DHCP server reorders the existing options so that the ones listed in the option are first and in the order that they appear in the list. Then, the remaining options appear in their current order after the last ones that were in the list. The DHCP server retains the list, and uses it to order any future options that it puts into the response, until it replaces the list with a new one.

When an extension does a get operation for the *dhcp-parameter-request-list* in the response dictionary, it does not look in the decoded response packet to find an option. Instead, the DHCP server synthesizes one that contains the list of all options currently in the decoded response packet.

Extension Point Descriptions

The following sections describe each extension point, their actions, and data items. For all the extension points, you can read the **extension-point** and set the *trace-level* data item values in the environment dictionary. For most extension points, you can also tell the server to drop the packet.

Related Topics

- [init-entry, on page 379](#)
- [post-packet-decode, on page 381](#)
- [pre-packet-decode, on page 380](#)
- [post-class-lookup, on page 382](#)
- [pre-client-lookup, on page 383](#)
- [post-client-lookup, on page 385](#)
- [generate-lease, on page 386](#)
- [check-lease-acceptable, on page 387](#)
- [lease-state-change, on page 388](#)
- [pre-packet-encode, on page 388](#)
- [post-packet-encode, on page 389](#)
- [pre-dns-add-forward, on page 389](#)
- [post-send-packet, on page 389](#)
- [environment-destructor, on page 390](#)

init-entry

The **init-entry** extension point is an additional one that the DHCP server calls when it configures or unconfigures the extension, which occurs when starting, stopping, or reloading the server. This entry point has the same signature as the others for the extension, but you can use only the environment dictionary. You do not configure the **init-entry** extension with **dhcp attachExtension** in the CLI, but you do so implicitly by defining an **init-entry** on an already configured extension. See the following table for the environment dictionary data items specific to **init-entry**.

Table 48: init-entry Environment Dictionary Data Items

Environment Data Item	Description
<i>dhcp-support</i> (input/output)	Version or versions of DHCP for which the server should call the registered extension points for the extension. Can be v4 , v6 , or v4,v6 .
<i>extension-extensionapi-version</i> (output)	Minimum version of the extension API required by the extension. Set it to 2 if the extension uses API features introduced in Cisco Prime Network Registrar Release 7.0. Set it to 1 for earlier releases.
<i>init-args</i> (input)	Configure arguments by setting <i>init-args</i> on an existing extension point. These arguments are present for both the configure and unconfigure calls of the init-entry entry point. The extension point name for the configure call is initialize , and for the unconfigure call is uninitialize .

Environment Data Item	Description
<i>server-dhcp-support</i> (input)	<p>The server sets this data item to indicate what the server is configured to support. Can be v4, v6, or v4,v6, depending on the DHCP server <i>dhcp-support</i> attribute setting (which requires setting expert attribute <i>visibility=3</i>) and whether any prefixes are configured:</p> <ul style="list-style-type: none"> • If <i>dhcp-support</i> =both and prefixes are not configured, then <i>server-dhcp-support</i> is set to v4. • If <i>dhcp-support</i> =both and one or more prefixes are configured, then <i>server-dhcp-support</i> is set to v4,v6. • If <i>dhcp-support</i> =v4, then <i>server-dhcp-support</i> is set to v4. • If <i>dhcp-support</i> =v6 and one or more prefixes are configured, then <i>server-dhcp-support</i> is set to v6.
<i>server-extensionapi-version</i> (input)	Version of the server extension API. The value is 2 for Cisco Prime Network Registrar 7.0 and later, and 1 for earlier releases.



Note You must supply an **init-entry** extension point to enable extension points for DHCPv6 (or disable them for DHCPv4).

In addition to configuring an **init-entry** with the name of the entry point, you can also configure a string of arguments that the DHCP server loads in the environment dictionary under the string *arguments* before calling the **init-entry** point. Using *arguments*, you can create a customized extension by giving it different initialization arguments and thus not require a change to the code to elicit different behavior.



Note The order in which the server calls extensions at the **init-entry** extension point can be different from reload to reload, or release to release.



Caution An extension, when called to uninitialized, must terminate any threads it creates and clean up after itself before returning. Once the extension returns, the DHCP server unloads the extension from memory, which could result in a server failure if a thread an extension created is left running.

pre-packet-decode

The dictionaries available for **pre-packet-decode** are request and environment.

This extension point is the first one the DHCP server encounters when a request arrives. The server calls it after receiving a packet but before it decodes the packet (at the **post-packet-decode** extension point). An extension can use this extension point to examine a packet and alter it before the server decodes it, or cause the server to drop it.

Two key data items in the request dictionary are for use with pre-packet-decode are client-packet and packet. These can be used to examine the received packet, modify the packet, and write it back.

**Caution**

The request dictionary *client-packet* and *packet* data items used for **pre-packet-decode** are available at any extension point that has a request dictionary. However, you should not directly alter or replace the packet at any extension point other than **pre-packet-decode**, because doing so can have unexpected side effects. For example, the server might never pick up the changes to the packet, or options data can change unexpectedly during processing.

An extension that uses **getBytes** with *client-packet* or *packet* directly alters the bytes of packet by writing into the returned buffer. However, an extension must use **put** or **putBytes** to adjust the length of the packet (and the operation can fail if the packet is too big). For DHCPv6, adjusting the length of the client portion of the packet, if relayed, requires updating the lengths in the Relay Message options in the packet.

It is up to an extension to handle parsing the packet to locate what it needs and properly alter the packet, if that is the intent.

Because the server has not yet decoded the received packet, most request dictionary data items are not available (as the server normally fills them in from the received packet). Thus, this extension point must extract data directly from the packet. The extension must also properly handle incorrectly formatted packets.

If you enable incoming-packet-detail logging, the server logs the received packet after calling the extensions registered at this extension point. If DHCP server debug tracing is configured with V is 3 or more, the server also logs the packet before calling the extensions registered for this extension point, if at least one extension is registered.

post-packet-decode

The dictionaries available for **post-packet-decode** are request and environment.

Related Topics

[Extension Description, on page 381](#)

[Overriding Client Identifiers, on page 382](#)

Extension Description

This extension point immediately follows the decoding of the input packet and precedes any processing on the data in the packet. The primary activity for an extension at this point is to read information from an input packet and do something with it. For example, you might use it to rewrite the input packet.

The **post-packet-decode** extension point is one of the easiest extension points to use. If you can express the change in server behavior as a rewrite of the input DHCP or BOOTP packet, you should use this extension point. Because the packet was decoded, but not processed in any way, the number of side effects are very limited.

The **post-packet-decode** extension point is the only one at which you can modify the decoded input packet and ensure that the server recognizes all the modifications. You can have the extension drop the packet and terminate further processing by using the **drop** data item in the environment dictionary.

Overriding Client Identifiers

To override client identifiers (IDs), you can set an expression value for the *override-client-id* attribute for a client-class or use the *override-client-id* data item at the **post-packet-decode** extension point. The extension method maps the client to a different identifier than the server.

There is a variant of the extension data item where you can get or put the override client ID as a string: *override-client-id-string*. You can also request the data type of the override client ID through the read-only *override-client-id-data-type* data item.

Different values are returned based on how you put and get the *override-client-id* or its *override-client-id-string* variant (see the following table for some examples).

Table 49: Puts and Gets of Client ID Overrides

Action	Data Item Used	Put Value	Get Value
put	<i>override-client-id</i>	01:02:03:04	
putBytes	<i>override-client-id</i>	01 02 03 04	
get	<i>override-client-id</i>		01:02:03:04 (blob)
getBytes	<i>override-client-id</i>		01 02 03 04 (raw bytes)
get[Bytes]	<i>override-client-id-string</i>		01:02:03:04 (blob-as-string)
get[Bytes]	<i>override-client-id-data-type</i>		blob
put[Bytes]	<i>override-client-id-string</i>	01:02:03:04 test	
get[Bytes]	<i>override-client-id-string</i>		01:02:03:04 (string) test (string)
get[Bytes]	<i>override-client-id</i>		30:31:3a:30:32:3a:30:33:3a:30:34 (blob) 74:65:73:74 (blob of "test")
get[Bytes]	<i>override-client-id-data-type</i>		nstr

The equivalent *client-override-client-id* data items (that you can use in later extension points where the response dictionary is valid) function the same way, although they are read-only.



Note When using *[v6-]override-client-id* expressions, leasequery by client-id requests may need to specify the *override-client-id* attribute to correctly retrieve the information on the lease(s) for the client.

post-class-lookup

The dictionaries available for **post-class-lookup** are request and environment.

The server calls this extension point only if there is a *client-class-lookup-id*; otherwise, it is similar to a **post-packet-decode**. The server calls the **post-class-lookup** extension point after evaluating the *client-class-lookup-id* and setting the client-class data for this client.

On input to this extension point, the environment dictionary has the *drop* data item set to true or false. You can change this setting by extension to drop the packet (or not drop it), and the server recognizes the change. The server also looks at the *log-drop-message* to decide whether to log the drop.

The extension point can also set the *client-class-name* in the environment dictionary, which sets the named client-class for this packet, regardless of the previous client-class. This setting has an effect only if the *drop* environment dictionary data item value is false on exiting the extension point.

pre-client-lookup

The dictionaries available for **pre-client-lookup** are request and environment.

You can use this extension point only if you enabled client-class processing for your DHCP server. This extension point allows an extension to perform any or all of these actions:

- Modify the client that the server looks up during client-class processing.
- Specify individual data items to override those found from the client entry or client-class it specifies.
- Instruct the server to skip the client lookup altogether. In this case, the only client data used is data that the extension supplied in the environment dictionary.

Although the request dictionary is available to make decisions about the operation of an extension running at this extension point, the environment dictionary controls all the operations.

Related Topics

[Environment Dictionary for pre-client-lookup, on page 383](#)

[post-client-lookup, on page 385](#)

[Overriding Client Identifiers, on page 382](#)

Environment Dictionary for pre-client-lookup

The environment dictionary data items in the table below are the control data items available at **pre-client-lookup** for clients and client-classes.

If you set the environment dictionary data items in [Table 51: pre-client-lookup Environment Dictionary Override Data Items](#), their values override those determined from the client lookup (either in the internal database or from LDAP). If you do not add anything to the dictionary, the server uses what is available in the client lookup.

Table 50: pre-client-lookup Environment Dictionary Control Data Items

Environment Data Item	Description
<i>client-specifier</i> (input/output)	Name of the client the client-class processing code looks up, in CNRDB or LDAP. If you change the name at this extension point, the DHCP server looks up the client you specify.

Environment Data Item	Description
<i>default-client-class-name</i> (output)	<p>Instructs the server to use the value associated with the <i>default-client-class-name</i> option as the <i>class-name</i> if:</p> <ul style="list-style-type: none"> The <i>client-specifier</i> data item was not specified in the pre-client-lookup script. The server could not locate the specific client entry. <p>The <i>default-client-class-name</i> data item then assumes precedence over the <i>class-name</i> associated with the default client.</p>
<i>release-by-ip</i> (output)	<p>Applies to DHCPRELEASE requests only. If set to true, instructs the server to release the lease by the IP address if it cannot retrieve the lease by the <i>client-id</i> as derived from the DHCPRELEASE request.</p>
<i>skip-client-lookup</i> (input/output)	<p>The value is determined by the server configuration. If set to true, the DHCP server skips the normal client lookup that it would have performed immediately upon exit from this extension.</p> <p>The only data items used to describe this client are those placed in the environment dictionary (see the table below).</p>

Table 51: pre-client-lookup Environment Dictionary Override Data Items

Environment Data Item	Description
<i>action</i> (input/output)	<p>Convert this string to a number and use the result as the action. The numbers you can use are 0 (for none) and 1 (for exclude).</p>
<i>authenticate-until</i> (input/output)	<p>Absolute time, measured in seconds, from January 1, 1970. Use to indicate the time at which the client authentication expires.</p> <p>When the client authentication expires, the DHCP server uses the values in the client <i>unauthenticated-client-class</i> option instead of its client-class to fill in missing data items in the client entry.</p>
<i>client-class-name</i> (input/output)	<p>Use the client-class specified by this data item to fill in the missing information in the client entry. If there is no client-class corresponding to the name specified, the DHCP server logs a warning and continues processing.</p> <p>If you specify none, the DHCP server acts as if the client entry did not include the client-class name.</p>
<i>domain-name</i> (output)	<p>Use this domain name for the client DNS operations in preference to the one specified in the DNS update configuration. The DNS server shown as the primary server for the domain in the scope or prefix must also be the primary server for the domain you specified.</p> <p>If the domain name is not overridden in the client or client-class entry, the DHCP server uses the domain name from the scope or prefix.</p> <p>If the client entry or the extension contains the word none, the DHCP server uses the domain name from the scope or prefix.</p>

Environment Data Item	Description
<i>host-name</i> (output)	Use this for the client in preference to the <i>host-name</i> options specified in the input packet, or any data from the client or client-class entry. If you set this to none , the DHCP server does not use any information from the client or client-class entry, but uses the name from the client request.
<i>policy-name</i> (input/output)	Use this policy as the policy specified for the client entry, overriding any policy specified by that client entry.
<i>selection-criteria</i> (input/output)	List of comma-separated strings, each specifying (for this input packet) the selection criteria for the client. Any scope or prefix the client uses must have all of these selection tags. Use this data item to override any criteria specified in the client or client-class entry. If you do, the DHCP server does not use the client entry selection criteria, independent of whether they were stored in the local or LDAP database. If you set this data item to none , the DHCP server does not use selection tags for the packet. If you set this to a null string, the DHCP server treats it as if it were not set and uses the selection criteria from the client or client-class entry.
<i>unauthenticated-client-class-name</i> (input/output)	Name of the client-class to use if the server does not authenticate the client. If you want to indicate without specifying the <i>unauthenticated-client-class-name</i> , use an invalid client-class name as the value of this data item. You can use the value none or any name that is not a client-class name. The DHCP server logs an error that the client-class is not present.

post-client-lookup

The dictionaries available for **post-client-lookup** are request and environment.

You can use this extension point to examine the results of the entire client-class processing operation, and take an action based on those results. You might want to use it to rewrite some of the results, or to drop the packet. If you want to override the hostname in the packet returned from the client-class processing from an extension running at the **post-client-lookup** extension point, set the hostname to the *client-requested-host-name* data item in the request dictionary. This causes Cisco Prime Network Registrar to look to the server as though the packet came in with whatever string you specified in that data item.

You also can use this extension point to place some data items in the environment dictionary to affect the processing of an extension running at the **pre-packet-encode** extension point (see [pre-packet-encode](#), on page 388), where it might load different options into the response packet or take other actions.

Environment Dictionary for post-client-lookup

The environment dictionary data items included in the following table are available at **post-client-lookup**.

Table 52: *post-client-lookup Environment Dictionary Data Items*

Environment Data Item	Description
<i>client-specifier</i> (input)	Name of the client that the client-class processing looked up.
<i>cnr-ldap-query-failed</i> (input)	<p>The DHCP server sets this attribute to ease recovery from LDAP server failures so that a post-client-lookup script can respond to an LDAP server failure.</p> <p>The DHCP server, after a client lookup, sets this flag to true if the LDAP query failed because of an LDAP server error. If the server received a response from the LDAP server, one of two conditions occurs:</p> <ul style="list-style-type: none"> • It sets the flag to false. • The <i>cnr-ldap-query-failed</i> attribute does not appear in the environment dictionary.

generate-lease

The dictionaries available for **generate-lease** are request, response, and environment. This extension point is available for DHCPv6 only. See the table below for the environment dictionary data items that apply to this extension point.

Table 53: *generate-lease Environment Dictionary Data Items*

Environment Data Item	Description
<i>attempts</i> (input)	Number of times that the server calls this extension for a single lease.
<i>default-prefix-length</i> (input)	Set to the default prefix length (from policies). The Expert mode <i>longest-prefix-length</i> and <i>shortest-prefix-length</i> data items, if not set, default to the <i>default-prefix-length</i> .
<i>extension-point</i> (input)	Name of the extension point. For example, post-packet-decode .
<i>extension-sequence</i> (input)	String that is the sequence number of this extension at this extension point.
<i>generated-address</i> (output)	Address the extension wants the server to use for the lease.
<i>generated-prefix</i> (output)	Delegated DHCPv6 prefix the extension wants the server to use for the lease.
<i>prefix-length</i> (input)	Set to the requested or default prefix length.
<i>skip-lease</i> (output)	Set to TRUE if the extension does not want the server to generate the lease.

You can use this extension point to generate a DHCPv6 address or prefix and allow the extension to control the address or prefix.

The server calls **generate-lease** only if the prefix is configured to allow extensions to be called during address allocation or prefix delegation—the extension flag must be set in the *allocation-algorithms* attribute for the prefix. When the server calls the generate-lease extension:

- The server sets the prefix context for the response dictionary to the prefix on which the lease is to be created. (Calling **setObject** with DEX_PREFIX and DEX_INITIAL will return to this context.)

- No lease context exists, because the server has not yet created a lease. However, lease-binding data items, in particular *lease-binding-type* and *lease-binding-iaid* are available. (Calling `setObject` with `DEX_LEASE` and `DEX_INITIAL` returns to this context and also sets the prefix, because a lease context sets three contexts: lease, binding, and prefix.)
- The server sets the *skip-lease* environment dictionary data item to false.
- The server sets the (read-only) *attempts* environment dictionary data item with the number of times (starting with 1) it called the extension to create this lease.
- For prefix delegation, the following environment dictionary data items are available:
 - **prefix-length**—Prefix length (requested or default prefix length).
 - **default-prefix-length**—Default prefix length (from policies).
 - **longest-prefix-length**—Longest allowable prefix (from policies).
 - **shortest-prefix-length**—Shortest allowable prefix (from policies).

When the extension returns, it can:

- Request an explicit address (for stateful address assignment) by setting the address on the *generated-address* environment dictionary data item. If the address is not available for the client (that is, if the address is already in use) or is not contained by the prefix, the server might call this extension again.
- Request an explicit prefix (for prefix delegation assignment) by setting the prefix on the *generated-prefix* environment dictionary data item. If the prefix is not available for the client or is not contained by the prefix, the server might call this extension again. The prefix is not available for the client under the following conditions:
 - if the prefix is already in use
 - if it is contained in a shorter prefix that has already been delegated
 - if a longer prefix contained in it has already been delegated by the server

The server will not reject the prefix if it is shorter or longer than allowed by the policy.

- Cause the server not to assign a lease by setting the *skip-lease* environment dictionary data item to true.
- Allow normal address assignment or prefix delegation to occur by not setting any of the above.

The server calls the extension point at most 500 times for each lease (this limit is the same one that currently applies when the server randomly generates leases). The server calls an extension multiple times only if the extension supplies an unusable address or delegated prefix (that is not in range for the prefix or already exists).



Note You cannot request the server to drop the packet at this extension point.

check-lease-acceptable

The dictionaries available for **check-lease-acceptable** are request, response, and environment.

This extension point comes immediately after the server determines whether the current lease is acceptable for this client. You can use this extension to examine the results of that operation, and to cause the routine to return different results. See [Determining Lease Acceptability, on page 368](#).

The *acceptable* data item is available in the environment dictionary at this extension point. This is a read/write data item that the DHCP server initializes depending on if the lease is acceptable for this client. You can read

and change this result in an extension. Setting the acceptable data item to true indicates that it is acceptable; setting it to false indicates that it is unacceptable.

lease-state-change

The dictionaries available for **lease-state-change** are response and environment.

The server calls this extension point when a lease state changes for either all state changes or when exiting the state specified in the exiting-state environment data item (see the following table). The existing state is in the lease-state response dictionary data item. The new state is in the environment dictionary data item *new-state*. The server never calls the extension point if the new state matches the existing one.

Use this extension point mainly for read-only purposes, although you can place data in the environment dictionary so that other extension points can get it later.

The **lease-state-change** can also have a different environment dictionary, such as for lease expirations.

Table 54: lease-state-change Environment Dictionary Data Items

Environment Data Item	Description
exiting-state (output)	<p>For an extension attached to the lease-state-change extension point, if specified, the lease-state-change extension point is called only if the current state of the lease is the state specified by exiting-state. The extension is only called when the specified state is exited. If not specified, and the extension is attached to the lease-state-change extension point, the extension will be called for all state changes. If specified, the exiting-state must specify a valid lease state: available, offered, leased, expired, unavailable, released, other-available, pending-available, revoked.</p> <p>There is no strict state transition table. In a failover environment, the server that receives a binding update message sets the state to whatever its partner informs it to be, without requiring specific state transitions.</p>

Related Topics

[Environment Dictionary for lease-state-change, on page 388](#)

Environment Dictionary for lease-state-change

The current state is in the *lease-state* lease information data item in the response dictionary, and the state being changed to is in the environment dictionary under the *new-state* data item. The response lease-state and environment new-state data items are read-only.

pre-packet-encode

The dictionaries available for **pre-packet-encode** are request, response, and environment.



Note For DHCPv6 Reconfigure messages, there is no request dictionary (because Reconfigure is a server-initiated message). Thus, enabled extensions should check the response *msg-type* for ADVERTISE or REPLY or use **isValid** on the request to ensure that the Reconfigure message exists.

post-packet-encode

The dictionaries available for **post-packet-encode** are request, response, and environment.



Note For DHCPv6 Reconfigure messages, there is no request dictionary (because Reconfigure is a server-initiated message). Thus, enabled extensions should check the response *msg-type* for ADVERTISE or REPLY or use **isValid** on the request to ensure that the request dictionary exists.

The server calls this extension point after encoding a packet, but before sending it to the client. The server can thereby examine and alter the packet before it sends the packet to the client, or the extension can cause the server to drop the packet (although the server might have made changes to its internal and on-disk data that will not be backed out if the packet is dropped).

The *client-packet* and *packet* data items were added to the response dictionary with similar behavior as described for the request dictionary in [pre-packet-decode, on page 380](#). Note that this extension point is the only one that can request the response *client-packet* or *packet* data items, because no packet exists at any other extension point. Also, the server does not process the changes made to the packet; the server simply sends the altered packet to the client.

If you enable outgoing-packet-detail logging, the server logs the packet after calling the extensions registered at this extension point. If DHCP server debug tracing is configured with $X \geq 3$, the server also logs the packet before calling the extensions registered for this extension point, but only if at least one extension is registered.

pre-dns-add-forward

Instead of using the `pre-dns-add-forward` extension point, you can use the `host-name-generator` (for DHCPv4) and `v6-host-name-generator` (for DHCPv6) expressions configurable on the DNS update configurations.



Note The **pre-dns-add-forward** extension point is deprecated and documentation for it removed. A future Cisco Prime Network Registrar release may remove the extension point completely. Instead, use an earlier request extension point (such as **post-client-lookup**) to set the required options, such as the *client-fqdn* option.

post-send-packet

Use the **post-send-packet** extension point for any processing that you want to perform outside of the serious time constraints of the DHCP request-response cycle. After the server sends the packet to the client, it calls this extension point.



Note For DHCPv6 Reconfigure messages, there is no request dictionary (because Reconfigure is a server-initiated message). Thus, enabled extensions should check the response *msg-type* for ADVERTISE or REPLY or use **isValid** on the request to ensure that the request dictionary exists.

environment-destroyer

The **environment-destroyer** extension point allows an extension to clean up any context that it might be holding. The only dictionary available for this extension point is environment.

The environment dictionary is available for all extension points called for a single client request. Because some extensions may need to maintain context information between the multiple extension points called for a single client request, and because the server might drop requests at several places during processing, an extension cannot reliably release context that it might have created for that request. The environment-destroyer extension point now makes it possible to reliably remove this context when processing of a request has completed, for whatever reason.



Note The server calls all extensions attached to the **environment-destroyer** extension point, even if the server did not call each extension at any other attachment point.



CHAPTER 13

DHCP Server Status Dashboard

The Cisco Prime Network Registrar server status dashboard in the web user interface (web UI) presents a graphical view of the system status, using graphs, charts, and tables, to help in tracking and diagnosis. These dashboard elements are designed to convey system information in an organized and consolidated way, and include:

- Significant protocol server and other metrics
- Alarms and alerts
- Database inventories
- Server health trends

The dashboard is best used in a troubleshooting desk context, where the system displaying the dashboard is dedicated for that purpose and might be distinct from the systems running the protocol servers. The dashboard system should point its browser to the system running the protocol servers.

You should interpret dashboard indicators in terms of deviations from your expected normal usage pattern. If you notice unusual spikes or drops in activity, there could be communication failures or power outages on the network that you need to investigate.

- [Opening the Dashboard, on page 391](#)
- [Display Types, on page 392](#)
- [Customizing the Display, on page 395](#)
- [Selecting Dashboard Elements to Include, on page 397](#)
- [DHCP Metrics, on page 399](#)

Opening the Dashboard

Starting from Cisco Prime Network Registrar 9.0, the Dashboard feature is available on the regional cluster also. It provides System Metrics chart by default. It allows you to display the server specific (DHCP, DNS, and CDNS) charts for various clusters. This can be configured in the Chart Selections page.

To open the dashboard in the web UI, from the **Operate** menu, choose **Dashboard**.

Display Types

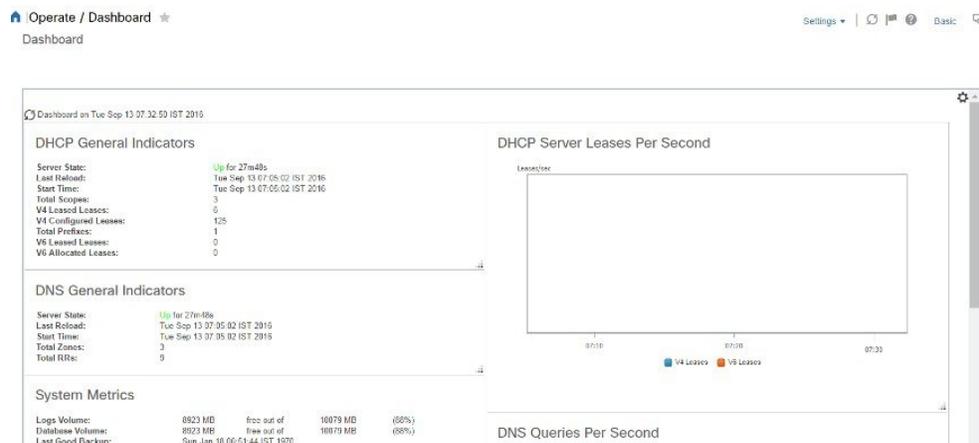
Provided you have DHCP and DNS privileges through administrator roles assigned to you, the preset display of the dashboard consists of the following tables (See the table below for an example):

- **System Metrics**—See the *"DHCP General Indicators"* section in *Cisco Prime Network Registrar 9.0 Administrator Guide*.
- **DHCP General Indicators**—See [DHCP Server Status Dashboard](#), on page 391.
- **DNS General Indicators**—See the *"DNS General Indicators"* section in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide*.



Tip These are just the preset selections. See [Selecting Dashboard Elements to Include](#), on page 397 for other dashboard elements you can select. The dashboard retains your selections from session to session.

Figure 15: Preset Dashboard Elements



Each dashboard element initially appears as a table or a specific chart type, depending on the element:

- **Table**—See [Tables](#), on page 393.
- **Line chart**—See [Line Charts](#), on page 394.
- **Stacked area chart**—See [Stacked Area Charts](#), on page 394.

General Status Indicators

Note the green box next to each dashboard element name in the above image. This box indicates that the server sourcing the information is functioning normally. A yellow box indicates that server operation is less than optimum. A red box indicates that the server is down. These indicators are the same as for the server health on the Manage Servers page in the regular web UI.

Graphic Indicators for Levels of Alert

Graphed lines and stacked areas in the charts follow a standard color and visual coding so that you can immediately determine key diagnostic indicators at a glance. The charts use the following color and textural indicators:

- **High alerts or warnings**—Lines or areas in red, with a hatched texture.
- **All other indicators**—Lines or areas in various other colors distinguish the data elements. The charts do not use green or yellow.

Magnifying and Converting Charts

If Magnified Chart is the selected Chart Link, you can magnify a chart in a separate window by clicking the chart. In magnified chart view, you can choose an alternative chart type from the one that comes up initially (see [Other Chart Types, on page 395](#)).



Note Automatic refresh is turned off for magnified charts. To get the most recent data, click the **Refresh** icon next to the word Dashboard at the top left of the page.

To convert a chart to a table, see the *"Displaying Charts as Tables"* section. You cannot convert tables to a graphic chart format.

Legends

Each chart initially includes a color-coded legend. To turn off the legend display on the main dashboard page, see the *"Displaying or Hiding Chart Legends"* section. Removing the legend renders the graphic chart size relatively larger, which can be helpful if you have many charts displayed. You cannot remove legends in magnified views.

Tables

Dashboard elements rendered as tables have data displayed in rows and columns. The following dashboard elements are preset to consist of (or include) tables:

- System Metrics
- DHCP DNS Updates
- DHCP Address Current Utilization
- DHCP General Indicators
- DNS General Indicators
- Caching DNS General Indicators



Note If you view a table in Expert mode, additional data might appear.

Line Charts

Dashboard elements rendered as line charts can include one or more lines plotted against the x and y axes. The three types of line charts are described in the following table.

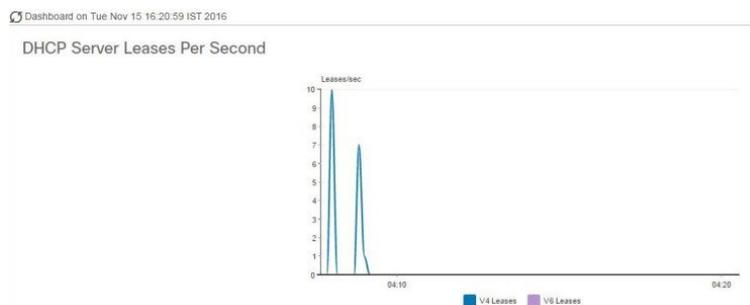
Table 55: Line Chart Types

Type of Line Chart	Description	Dashboard Elements Rendered
Raw data line chart	Lines plotted against raw data.	<ul style="list-style-type: none"> • Java Virtual Machine (JVM) Memory Utilization (Expert mode only) • DHCP Buffer Capacity • DHCP Failover Status (two charts) • DNS Network Errors • DNS Related Servers Errors
Delta line chart	Lines plotted against the difference between two sequential raw data.	<ul style="list-style-type: none"> • DNS Inbound Zone Transfers • DNS Outbound Zone Transfers
Rate line chart	Lines plotted against the difference between two sequential raw data divided by the sample time between them.	<ul style="list-style-type: none"> • DHCP Server Request Activity (see the image below) • DHCP Server Response Activity • DHCP Response Latency • DNS Query Responses • DNS Forwarding Errors



Tip To get the raw data for a chart that shows delta or rate data, enter Expert mode, set the Chart Link to Data Table, then click the chart. The Raw Data table is below the Chart Data table.

Figure 16: Line Chart Example



Stacked Area Charts

Dashboard elements rendered as stacked area charts have multiple related metrics plotted as trend charts, but stacked one on top of the other, so that the highest point represents a cumulative value. The values are independently shaded in contrasting colors. (See the image below for an example of the DHCP Server Request Activity chart shown in [Figure 16: Line Chart Example, on page 394](#) rendered as a stacked area chart.)

Figure 17: Stacked Area Chart Example

They are stacked in the order listed in the legend, the left-most legend item at the bottom of the stack and the right-most legend item at the top of the stack. The dashboard elements that are pre-set to stacked area charts are:

- DHCP Server Request Activity
- DHCP Server Response Activity
- DHCP Response Latency
- DNS Outbound Zone Transfers
- DNS Inbound Zone Transfers

Other Chart Types

The other chart types available for you to choose are:

- **Line**—One of the line charts described in [Table 55: Line Chart Types, on page 394](#).
- **Stacked Area**—Charts described in the [Stacked Area Charts, on page 394](#).
- **Pie**—Shows a single percentage pie chart of the data averaged over the time sampled.
- **Bar**—Multiple related current value metrics plotted side by side as groups of bars that show the actual data sampled.
- **Stacked Bar**—Addition total of the actual samples. This chart shows more distinct data points than the stacked area chart.



Tip Each chart type shows the data in distinct ways and in different interpretations. You can decide which type best suits your needs.

Getting Help for the Dashboard Elements

You can open a help window for each dashboard element by clicking the title of the element.

Customizing the Display

To customize the dashboard display, you can:

- Refresh the data and set an automatic refresh interval.
- Expand a chart and render it in a different format.
- Convert a graphic chart to a table.
- Download data to comma-separated value (CSV) output.
- Display or hide chart legends.
- Configure server chart types.
- Reset to default display

Each chart supports:

- Resizing
- Drag and drop to new cell position
- Minimizing
- Closing

Each chart has a help icon with a description of the chart and a detailed help if you click the chart title.



Note The changes made to the dashboard/chart will persist only if you click **Save** in the Dashboard window.

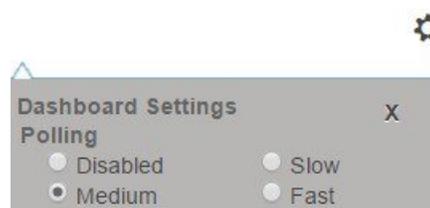
Refreshing Displays

Refresh each display so that it picks up the most recent polling by clicking the **Refresh** icon.

Setting the Polling Interval

You can set how often to poll for data. Click the **Dashboard Settings** icon in the upper-right corner of the dashboard display. There are four options to set the polling interval of the cached data, which polls the protocol servers for updates. (See the image below)

Figure 18: Setting the Chart Polling Interval



You can set the cached data polling (hence, automatic refresh) interval to:

- **Disabled**— Does not poll, therefore does not automatically refresh the data.
- **Slow**— Refreshes the data every 30 seconds.

- **Medium**— Refreshes the data every 20 seconds.
- **Fast** (the preset value)— Refreshes the data every 10 seconds.

Displaying Charts as Tables

You can choose to display a graphic chart as a table when you magnify the chart by clicking it. At the middle of the top of the dashboard display are the controls for the chart links (see the image below)

Figure 19: Specifying Chart Conversion to Table Format



Click the **Data Table** radio button. When you click the chart itself, it opens as a table. The preset display format is Magnified Chart.

Exporting to CSV Format

You can dump the chart data to a comma-separated value (CSV) file (such as a spreadsheet) when you magnify the chart by clicking it. In the Chart Link controls at the top of the page (see the above image), click the **CSV Export** radio button, then click the chart. A Save As window appears, where you can specify the name and location of the CSV file.

Displaying or Hiding Chart Legends

You can include or exclude the color-coded legends for charts on the main dashboard page. You might want to remove the legends as you become more familiar with the data and track it on a slightly larger chart display. In the upper-right of the dashboard display are the controls for the legend display (see the image below). The preset value is Visible.

Figure 20: Displaying or Hiding Chart Legends and Selecting Chart



Selecting Dashboard Elements to Include

You can decide how many dashboard elements you want to display on the page. At times, you might want to focus on one server activity only, such as for the DHCP server, and exclude all other metrics for the other servers. In this way, the dashboard becomes less crowded, the elements are larger and more readable. At other times, you might want an overview of all server activities, with a resulting smaller element display.

You can select the dashboard elements to display from the main Dashboard page by clicking **Chart Selections** in the Dashboard Settings dialog. Clicking the link opens the Chart Selection page (see [Figure 21: Selecting Dashboard Elements, on page 398](#)).

Configuring Server Chart Types

You can set the default chart types on the main dashboard view. You can customize the server charts in the dashboard to display only the specific chart types as default.

To set up default chart type, check the check box corresponding to the Metrics chart that you want to display and choose a chart type from the **Type** drop-down list. The default chart types are consistent and shared across different user sessions (see the image below).



Note You can see either the CDNS or DNS Metrics in the **Dashboard Settings > Chart Selection** page based on the service configured on the server.



Tip The order in which the dashboard elements appear in the Chart Selection list does not necessarily determine the order in which the elements will appear on the page. An algorithm that considers the available space determines the order and size in a grid layout. The layout might be different each time you submit the dashboard element selections. To change selections, check the check box next to the dashboard element that you want to display.

Figure 21: Selecting Dashboard Elements

Chart Selections

Change Chart Selection: Default | Chart Columns: 2

▼ Host Metrics

Chart

System Metrics

▼ DHCP Metrics

Chart	Type	Clusters
<input type="checkbox"/> DHCP Address Current Utilization	Table	
<input type="checkbox"/> DHCP Buffer Capacity	Stacked Area Chart	
<input type="checkbox"/> DHCP DNS Updates	Table	
<input type="checkbox"/> DHCP Failover Status	Stacked Area Chart	
<input type="checkbox"/> DHCP General Indicators	Table	
<input type="checkbox"/> DHCP Response Latency	Stacked Area Chart	
<input type="checkbox"/> DHCP Server Leases Per Second	Stacked Area Chart	
<input checked="" type="checkbox"/> DHCP Server Request Activity	Line Chart	
<input type="checkbox"/> DHCP Server Response Activity	Stacked Area Chart	

OK Close

The above image displays the Charts Selection table in the regional web UI. The Clusters column is available only in regional dashboard and it displays the list of local clusters configured. You can add the local cluster by clicking the Edit icon and then by selecting the local cluster name from the Local Cluster List dialog box.

To change selections, check the check box next to the dashboard element that you want to display.

Specific group controls are available in the drop-down list, **Change Chart Selection**, at the top of the page. To:

- Uncheck all check boxes, choose **None**.
- Revert to the preset selections, choose **Default**. The preset dashboard elements for administrator roles supporting DHCP and DNS are:
 - Host Metrics: System Metrics
 - DHCP Metrics: General Indicators
 - DNS Metrics: General Indicators
- Select the DHCP metrics only, choose **DHCP** (see the "DHCP Metrics" section in *Cisco Prime Network Registrar 9.0 DHCP User Guide*).
- Select the DNS metrics only, choose **DNS** (see the "Dashboard and Authoritative DNS Metrics" section in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide*).
- Select the DNS metrics only, choose **CDNS** (see the "Caching DNS Metrics" section in *Cisco Prime Network Registrar 9.0 Authoritative and Caching DNS User Guide*)
- Select all the dashboard elements, choose **All**.

Click **OK** at the bottom of the page to save your choices, or **Cancel** to cancel the changes.

DHCP Metrics

These DHCP metric elements are available in the dashboard:

- **DHCP Server Request Activity**—See [DHCP Server Request Activity, on page 399](#)
- **DHCP Server Response Activity**—See [DHCP Server Response Activity, on page 400](#)
- **DHCP Buffer Capacity**—See [DHCP Buffer Capacity, on page 401](#)
- **DHCP Response Latency**—See [DHCP Response Latency, on page 401](#)
- **DHCP DNS Updates**—See [DHCP DNS Updates, on page 402](#)
- **DHCP Address Current Utilization**—See [DHCP Address Current Utilization, on page 402](#)
- **DHCP Failover Status**—See [DHCP Failover Status, on page 403](#)
- **DHCP General Indicators**—See [DHCP General Indicators, on page 404](#)
- **DHCP Server Lease Data**—See [DHCP Server Lease Data, on page 404](#)

DHCP Server Request Activity

The DHCP Server Request Activity dashboard element rendered as a stacked area chart traces the totals in the change rate of incoming DHCP packet activity. The chart is available if you choose **DHCP Metrics: DHCP Server Request Activity** in the Chart Selection list.

The resulting stacked area chart plots the following trends:

- **V4 Discovers**—Number of DHCPv4 discover packets.
- **V4 Requests**—Number of DHCPv4 request packets.

- **V4 Other**—Number of DHCPv4 release, decline, or info-request packets.
- **V4 Lease Queries**—Number of DHCPv4 lease query packets.
- **V6 Solicits**—Number of DHCPv6 solicit packets.
- **V6 Requests/Renews/Rebinds**—Number of DHCPv6 request, renew, and rebind packets.
- **V6 Other**—Number of DHCPv6 release, decline, or information-request packets.
- **V6 Lease Queries**—Number of DHCPv6 lease query packets.
- **Invalid Packets**—Combined number of invalid DHCPv4 and DHCPv6 packets.

How to Interpret the Data

The DHCP Server Request Activity data shows the pattern of server traffic based on incoming DHCP requests. The trend should be fairly consistent, with spikes in the number of Invalid packets being a sign that there is some misconfigured data on the network. Note that DHCPv4 and DHCPv6 invalid packet activity is grouped together.

Troubleshooting Based on the Results

Check your DHCP server configurations if there is a sudden spike in activity, especially in the number of invalid request packets. Set your server logging to report where the activity is occurring. Spikes or drops in activity can indicate network or power outages that are worth investigating. Spikes in activity can also indicate a faulty client, malicious client activity, or a recovery after a power failure or outage that results in pent-up requests.

DHCP Server Response Activity

The DHCP Server Response Activity dashboard element rendered as a stacked area chart traces the totals in the change rate of outgoing DHCP packet activity. The chart is available if you choose **DHCP Metrics: DHCP Server Response Activity** in the Chart Selection list.

The resulting stacked area chart plots the following trends:

- **V4 Offers**—Number of DHCPv4 offer packets.
- **V4 Acks**—Number of DHCPv4 acknowledgment packets.
- **V4 Other Client**—Number of other outgoing DHCPv4 client packets.
- **V4 Lease Queries**—Number of outgoing DHCPv4 lease query packets.
- **V6 Advertises**—Number of DHCPv6 advertise packets.
- **V6 Replies**—Number of DHCPv6 reply packets.
- **V6 Reconfigures**—Number of DHCPv6 reconfigure packets.
- **V6 Lease Query Replies**—Number of DHCPv6 lease query reply packets.
- **Total Dropped**—Combined number of dropped DHCPv4 and DHCPv6 packets.

How to Interpret the Data

The DHCP Server Response Activity data shows the pattern of server traffic to answer DHCP requests. The trend should be fairly consistent, with spikes in the number of Total Dropped packets being a sign that there is some misconfigured data on the network. Note that DHCPv4 and DHCPv6 dropped packet activity is grouped together.

Troubleshooting Based on the Results

Check your DHCP server configurations if there is a sudden spike in activity, especially in the number of total dropped response packets. The response activity should match the request activity, except for the normal time shift, and the same diagnostics apply.

DHCP Buffer Capacity

The DHCP Buffer Capacity dashboard element rendered as a table shows the number of allocated requests and responses, and a line chart that plots the number of requests and responses in use. The element is available if you choose **DHCP Metrics: DHCP Buffer Capacity** in the Chart Selection list.

The resulting table and line chart plots:

- **Requests in Use**—Trend in the number of in-use request buffers.
- **Responses in Use**—Trend in the number of in-use response buffers.

How to Interpret the Data

The DHCP Buffer Capacity data shows the pattern in the use of DHCP request and response buffers. If the buffers begin to increase in an abnormal pattern, there are measures you can take without trying to compensate by increasing the number of allocated buffers.

Troubleshooting Based on the Results

If you see increasing and consistent exceeding of the buffer threshold, find the reason why the server is running slowly. Possible reasons include high degrees of logging, slow DHCP extensions or LDAP servers, or overload, such as with chatty clients or frequent rebooting of cable modem termination systems (CMTSs). You might need to increase the buffer sizes.

DHCP Response Latency

The DHCP Response Latency dashboard element rendered as a stacked area chart shows the trend in the response packet latency (the time interval between the request packet and its ensuing response). The chart is available if you choose **DHCP Metrics: DHCP Response Latency** in the Chart Selection list.



Tip You must also set the *collect-sample-counters* DHCP server attribute for this data to display, with the *enhanced-sample-counters* attribute also set for further granularity. These attribute values are preset. If you are concerned about achieving maximum performance, unset these attributes. (See the *"Displaying Statistics"* section in *Cisco Prime Network Registrar 9.0 Administrator Guide*.)

The resulting stacked area chart plots response latencies at the intervals:

- Less than 50 milliseconds
- 50 to 200 milliseconds
- 200 to 500 milliseconds
- 500 to 1000 milliseconds (note that if the *enhanced-sample-counters* attribute is not set, all values below 1 second appear in this grouping)
- 1 to 2 seconds
- 2 to 3 seconds

- 3 to 4 seconds
- More than 4 seconds

How to Interpret the Data

The chart shows the trend in response packet latency as an indicator of how long it takes to respond to incoming packets. The gradations in the latency periods are stacked.

Troubleshooting Based on the Results

High response packet latency is similar to high buffer usage for troubleshooting purposes. Look for slow LDAP servers or DHCP extensions, high levels of logging, or disk I/O bottlenecks.

DHCP DNS Updates

The DHCP DNS Updates dashboard element rendered as a table shows the related DNS server and its current state, and how many pending DNS updates are occurring between it and the DHCP server. The table is available if you choose **DHCP Metrics: DHCP DNS Updates** in the Chart Selection list.

The resulting table shows:

- **Server**—Related DNS server IP address
- **State**—Related DNS server state
- **Pending Updates**—Total number of pending updates

How to Interpret the Data

A high level of pending updates to a specific DNS server indicates that the server is unreachable or unavailable, or its address is wrong.

Troubleshooting Based on the Results

Check into the reachability of the associated DNS servers if the pending update rate spikes, or ensure that the address of the associated server is correct.

DHCP Address Current Utilization

The DHCP Address Current Utilization dashboard element rendered as a table shows the DHCPv4 address utilization (how many assigned addresses exist) for a particular address aggregation, which can be a scope, network, or network plus selection tag. The table is available if you choose **DHCP Metrics: DHCP Address Current Utilization** in the Chart Selection list.

The resulting table shows:

- **Name**—Aggregation name (or address).
- **In Use**—Number of in-use addresses.
- **Total**—Total number of addresses.
- **Utilization**—Percentage of utilized addresses.
- **Mode** (appears in Expert mode only)—Aggregation mode (scope, network, or selection-tags).

How to Interpret the Data

The chart shows a table with four columns: the scope name, its in-use and total addresses, and the percentage of address utilization based on the previous two columns. The chart is available only if the DHCP server *enhanced-sample-counters* attribute is enabled.

- If an SNMP trap configuration in scope mode applies, the Name column displays the scope name. Otherwise, it shows the network IP address.
- If traps are not enabled (or if the DHCP server *default-free-address-config* or *v6-default-free-address-config* attribute is not set), the network address is appended with an asterisk (*).
- If a selection tag applies, its name is also appended. See the "Handling SNMP Notification Events" section in *Cisco Prime Network Registrar 9.0 Administrator Guide* for details on SNMP traps.
- If you do not define a *default-free-address-config* (or *v6-default-free-address-config*) attribute, Cisco Prime Network Registrar creates an internal, unlisted trap configuration named **default-aggregation-addr-trap-config**.

Because of this, do not use the name `default-aggregation-addr-trap-config` for a trap configuration you create.

Troubleshooting Based on the Results

If the percentage of utilized addresses is high, the addresses reached a saturation point. It might be necessary to reassign addresses from a different scope.

DHCP Failover Status

The DHCP Failover Status dashboard element rendered as two parallel trend charts that show the current and partner server state and the binding updates and acknowledgments sent and received between the two failover partners. The charts are available if you choose **DHCP Metrics: DHCP Failover Status** in the Chart Selection list.



Note The failover status is only for the first failover pair in the related servers list.

The display is a table along with two rate line trend charts that shows the failover status for the first failover pair for the related servers:

- **Local State**—Local DHCP server failover state along with when it occurred.
- **Partner State**—Partner server failover state along with when it occurred.
- **DHCP Failover Status Updates Received**—The first trend chart shows a comparison of the number of binding updates received and binding acknowledgments sent.
- **DHCP Failover Status Updates Sent**—The second trend chart shows a comparison of the number of binding updates sent and binding acknowledgments received.

How to Interpret the Data

Along with some state data, the display is split into two line trend charts that are inverses of each other. Each chart compares the binding updates with the acknowledgments. The top chart pairs the binding updates received with the acknowledgments sent; the bottom chart pairs the binding updates sent with the acknowledgments received.

Troubleshooting Based on the Results

If the Partner State value is other than 10, check the configuration of the partner server. The updates sent and received data should also be fairly level.

DHCP General Indicators

The DHCP General Indicators dashboard element rendered as a table shows the server state, reload data, and lease counts. The table is available if you choose **DHCP Metrics: DHCP General Indicators** in the Chart Selection list.

The resulting table shows:

- **Server State**—Up or Down (based on whether statistics are available) and its duration.
- **Last Reload**—Date and time of the last server reload.
- **Start Time**—Date and time of the last server process (Cisco Prime Network Registrar server agent) startup.
- **Total Scopes**—Total number of configured DHCPv4 scopes.
- **V4 Leased Leases**—Number of active DHCPv4 leases, including reservations.
- **V4 Configured Leases**—Number of configured DHCPv4 leases, including reservations and ranges.
- **Total Prefixes**—Number of configured DHCPv6 prefixes.
- **V6 Leased Leases**—Number of active DHCPv6 leases, including reservations and delegated prefixes (which each count as one lease).
- **V6 Allocated Leases**—Number of allocated DHCPv6 leases, including reservations and delegated prefixes (which each count as one lease).

How to Interpret the Data

The table indicates the server state, process start time (via the Cisco Prime Network Registrar server agent), and reload data, and also provides lease statistics. The top set of data compares the DHCPv4 leases actually in effect with those configured; the bottom set of data does the same for DHCPv6 leases.

Time of last reload is important for determining if recent changes to the server configuration occurred from a reload operation. It can also help pinpoint when server changes were last applied, if other indicators show a marked, unexpected behavioral change. Be sure to preserve log files since the last reload.

Troubleshooting Based on the Results

A drop or increase in leases might indicate a power or network outage, but it can also indicate a normal variation depending on lease times and usage patterns. The number of scopes or prefixes indicated might also require some evaluation and possible reconfiguration. If the server state is Down, all the DHCP chart indicators show a red status box, so no data will be available. In the case of a server that is down, restart the server.

DHCP Server Lease Data

The DHCP Server Lease Data dashboard element, rendered as chart, shows the number of leases per second for the DHCP server. This chart is available if you choose **DHCP Metrics: DHCP Server Lease Data** in the Chart Selection page.

The chart displays:

- **V4 Leases**—Number of IPv4 leases per second.

- **V6 Leases**—Number of IPv6 leases per second.



APPENDIX **A**

DHCP Options

DHCP provides a framework for passing configuration information to hosts on a TCP/IP network. Configuration parameters and other control information are carried in tagged data items that are stored in the options field of the DHCP message. The data items themselves are also called options.

This appendix contains DHCP options and BOOTP vendor extensions from RFC 2132, and includes the validation type for each option, as indicated in [Table 65: DHCPv4 Options by Number](#), on page 428.

This appendix also contains the standard Microsoft client options and several tables displaying the options sorted by categories.

- [Option Descriptions](#), on page 407
- [Option Tables](#), on page 428

Option Descriptions

The following sections describe the DHCP options in detail:

- [RFC 1497 Vendor Extensions](#), on page 407
- [IP Layer Parameters Per Host](#), on page 409
- [IP Layer Parameters Per Interface](#), on page 410
- [Link Layer Parameters Per Interface](#), on page 411
- [TCP Parameters](#), on page 411
- [Application and Service Parameters](#), on page 411
- [DHCPv4 Extension Options](#), on page 417
- [DHCPv6 Options](#), on page 420
- [Microsoft Client Options](#), on page 419
- [Options by Number](#), on page 428
- [Options by Cisco Prime Network Registrar Name](#), on page 435
- [Option Validation Types](#), on page 445

RFC 1497 Vendor Extensions

The table below lists the vendor extensions as defined in RFC 1497.

Table 56: RFC 1497 Vendor Extension Options

Option Name	No.	Length	Description
Pad	0	1 octet	Causes the subsequent fields to align on word boundaries.
End	255	1 octet	End of valid information in the vendor field. Subsequent octets should be filled with the Pad options.
Subnet Mask	1	4 octets	Client subnet mask, as per RFC 950. If both the Subnet Mask and the Router option are specified in a DHCP reply, the Subnet Mask option must be first.
Time Offset	2	4 octets	Offset of the client subnet, in seconds, from Universal Time (UT). The offset is expressed as a twos-complement 32-bit integer. A positive offset indicates a location east of the zero meridian and a negative offset indicates a location west of the zero meridian.
Router	3	4 octets minimum; multiples of 4	List of IP addresses for routers on the client subnet. Routers should be in order of preference.
Time Server	4	4 octets minimum; multiples of 4	List of RFC 868 compliant time servers available to the client. Servers should be in order of preference.
Name Server Option	5	4 octets minimum; multiples of 4	List of IEN 116 name servers available to the client. Servers should be in order of preference.
Domain Name Server	6	4 octets minimum; multiples of 4	List of Domain Name System (STD 13, RFC 1035) name servers available to the client. Servers should be in order of preference.
Log Server	7	4 octets minimum; multiples of 4	List of MIT-LCS UDP log servers available to the client. Servers should be in order of preference.
Cookie Server	8	4 octets minimum; multiples of 4	List of RFC 865-compliant cookie servers available to the client. Servers should be in order of preference.
LPR Server	9	4 octets minimum; multiples of 4	List of RFC 1179-compliant line printer servers available to the client. Servers should be in order of preference.
Impress Server	10	4 octets minimum; multiples of 4	List of Imagen Impress servers available to the client. Servers should be in order of preference.
Resource Location Server	11	4 octets minimum; multiples of 4	List of RFC 887-compliant resource location servers available to the client. Servers should be in order of preference.
Host Name	12	1 octet minimum	Name of the client. The name may or may not be qualified with the local domain name. See RFC 1035 for the character set restrictions.
Boot File Size	13	2 octets	Number of 512-octet blocks in the default boot file.

Option Name	No.	Length	Description
Merit Dump File	14	1 octet minimum	Path name of a file to which the client core image should be placed in the event the client crashes. The path is formatted as a character string consisting of characters from the NVT ASCII character set.
Domain Name	15	1 octet minimum	Domain name that the client should use when resolving hostnames through the Domain Name System.
Swap Server	16	4 octets	IP address of the client swap server.
Root Path	17	1 octet minimum	Path name that contains the client root disk. The path is formatted as a character string consisting of characters from the NVT ASCII character set.
Extensions Path	18	1 octet minimum	Uses a string to specify a file, retrievable through TFTP. The file contains information that can be interpreted in the same way as the 64-octet vendor-extension field within the BOOTP response, with these exceptions: the length of the file is unconstrained, and all references to instances of this option in the file are ignored.

IP Layer Parameters Per Host

The table below lists the options that affect the operation of the IP layer on a per-host basis.

Table 57: IP Layer Parameters Per Host Options

Option Name	No.	Length	Description
IP Forwarding Enable/Disable	19	1 octet	Specifies whether the client should configure its IP layer for packet forwarding. Values: 0=disable; 1=enable
Non-Local Source Routing Enable/Disable	20	1 octet	Specifies whether the client should configure its IP layer to allow forwarding of datagrams with non-local source routes. Values: 0=disable; 1=enable
Policy Filter	21	8 octets minimum; multiples of 8	Policy filters for non-local source routing. The filters consist of a list of IP addresses and masks that specify destination/mask pairs with which to filter incoming source routes. Any source-routed datagram whose next-hop address does not match one of the filters should be discarded by the client.
Maximum Datagram Reassembly Size	22	2 octets	Maximum size datagram that the client should be prepared to reassemble. Value: 576 minimum
Default IP Time-to-live	23	1 octet	Default TTL that the client should use on outgoing datagrams. Values: 1 to 255

Option Name	No.	Length	Description
Path MTU Aging Timeout	24	4 octets	Timeout (in seconds) to use when aging Path MTU values (defined in RFC 1191).
Path MTU Plateau Table	25	2 octets minimum; multiples of 2	Table of MTU sizes to use when performing Path MTU Discovery as defined in RFC 1191. The table is formatted as a list of 16-bit unsigned integers, ordered from smallest to largest. Value: 68 minimum

IP Layer Parameters Per Interface

The table below lists the options that affect the operation of the IP layer on a per-interface basis. A client can issue multiple requests, one per interface, to configure interfaces with their specific parameters.

Table 58: IP Layer Parameters Per Interface Options

Option Name	No.	Length	Description
Interface MTU	26	2 octets	MTU to use on this interface. The minimum legal value for the MTU is 68.
All Subnets are Local	27	1 octet	Specifies whether or not the client can assume that all subnets of the IP network to which the client is connected use the same MTU as the subnet of that network to which the client is directly connected. Values: 1=all subnets share same MTU; 0=some directly-connected subnets can have smaller MTUs
Broadcast Address	28	4 octets	Broadcast address in use on the client subnet.
Perform Mask Discovery	29	1 octet	Specifies whether or not the client should perform subnet mask discovery using ICMP. Values: 0=disable; 1=enable
Mask Supplier	30	1 octet	Specifies whether or not the client should respond to subnet mask requests using ICMP. Values: 0=do not respond; 1=respond
Perform Router Discovery	31	1 octet	Specifies whether or not the client should solicit routers using the Router Discovery mechanism defined in RFC 1256. Values: 0=disable; 1=enable
Router Solicitation Address	32	4 octets	Address to which the client should transmit router solicitation requests.
Static Route	33	8 octets minimum; multiples of 8	List of static routes that the client should install in its routing cache. If multiple routes to the same destination are specified, they are in descending order of priority. The routes consist of a list of IP address pairs. The first address is the destination address, and the second address is the router for the destination. The default route (0.0.0.0) is an illegal destination for a static route.

Link Layer Parameters Per Interface

The table below lists the options that affect the operation of the data link layer on a per-interface basis.

Table 59: Link Layer Parameters Per Interface Options

Option Name	No.	Length	Description
Trailer Encapsulation	34	1 octet	Specifies whether or not the client should negotiate the use of trailers (RFC 893) when using the ARP protocol. Values: 0=do not use; 1=use
ARP Cache Timeout	35	4 octets	Timeout in seconds for ARP cache entries.
Ethernet Encapsulation	36	1 octet	Specifies whether or not the client should use Ethernet Version 2 (RFC 894) or IEEE 802.3 (RFC 1042) encapsulation if the interface is an Ethernet. Value: 0=use RFC 894 encapsulation; 1=use RFC 1042 encapsulation

TCP Parameters

The table below lists the options that affect the operation of the TCP layer on a per-interface basis.

Table 60: TCP Parameter Options

Option Name	No.	Length	Description
TCP Default TTL	37	1 octet	Default TTL that the client should use when sending TCP segments. Value: minimum 1
TCP Keepalive Interval	38	4 octets	Interval (in seconds) that the client TCP should wait before sending a keepalive message on a TCP connection. The time is specified as a 32-bit unsigned integer. A value of zero indicates that the client should not generate keepalive messages on connections unless specifically requested by an application. Value: 32-bit unsigned; 0=do not generate keepalive messages unless specifically requested.
TCP Keepalive Garbage	39	1 octet	Specifies the whether or not the client should send TCP keep-alive messages with an octet of garbage for compatibility with older implementations. Values: 0=do not send; 1=send

Application and Service Parameters

The table below lists some miscellaneous options used to configure miscellaneous applications and services.

Table 61: Application and Service Parameter Options

Option Name	No.	Length	Description
Network Information Service (NIS) Domain	40	1 octet minimum	Name of the client NIS domain. The domain is formatted as a character string consisting of characters from the NVT ASCII character set.
Network Information Service (NIS) Servers	41	4 octets minimum; multiples of 4	List of IP addresses indicating NIS servers available to the client. Servers should be in order of preference.
Network Time Protocol Servers	42	4 octets minimum; multiples of 4	List of IP addresses indicating NTP servers that are available to the client. Servers should be in order of preference.

Option Name	No.	Length	Description
Vendor-Specific Information	43	1 octet minimum	

Option Name	No.	Length	Description
			<p>This option is used by clients and servers to exchange vendor-specific information. The information is an opaque object of n octets, presumably interpreted by vendor-specific code on the clients and servers. The definition of this information is vendor specific. The vendor is indicated in the <i>dhcp-class-identifier</i> option. Servers not equipped to interpret the vendor-specific information sent by a client must ignore it (although it can be reported). Clients that do not receive desired vendor-specific information should make an attempt to operate without it, although they can do so (and announce they are doing so) in a degraded mode.</p> <p>If a vendor potentially encodes more than one item of information in this option, then the vendor should encode the option using encapsulated vendor-specific options as described here.</p> <p>The encapsulated vendor-specific options field should be encoded as a sequence of code, length, and value fields of identical syntax to the DHCP options field with these exceptions:</p> <ul style="list-style-type: none"> • There should not be a magic cookie field in the encapsulated vendor-specific extensions field. • Codes other than 0 or 255 can be redefined by the vendor within the encapsulated vendor-specific extensions field, but should conform to the tag-length-value syntax defined in section 2.

Option Name	No.	Length	Description
			Code 255 (END), if present, signifies the end of the encapsulated vendor extensions, not the end of the vendor extensions field. If the code 255 is not present, then the end of the enclosing vendor-specific information field is taken as the end of the encapsulated vendor-specific extensions field.
NetBIOS over TCP/IP Name Server	44	4 octets minimum; multiples of 4	List of RFC 1001/1002 NBNS name servers in order of preference.
NetBIOS over TCP/IP Datagram Distribution Server	45	4 octets minimum; multiples of 4	List of RFC 1001/1002 NBDD servers in order of preference.
NetBIOS over TCP/IP Node Type	46	1 octet	Allows NetBIOS over TCP/IP client, which are configured as described in RFC 1001/1002. Values: Single hexadecimal octet that identifies the client type: <ul style="list-style-type: none"> • 0x1=B-node (broadcast node) • 0x2=P-node (point-to-point node) • 0x4=M-node (mixed node) • 0x8=H-node
NetBIOS over TCP/IP Scope	47	1 octet minimum	NetBIOS over TCP/IP scope parameter for the client as specified in RFC 1001/1002.
X Window System Font Server	48	4 octets minimum; multiples of 4	List of X Window System Font servers available to the client. Servers should be in order of preference.
X Window System Display Manager	49	4 octets minimum; multiples of 4	List of IP addresses of systems that are running the X Window System Display Manager and are available to the client. Addresses should be in order of preference.

Option Name	No.	Length	Description
Network Information Service (NIS+) Domain	64	1 octet minimum	Name of the client NIS+ domain. The domain is formatted as a character string consisting of characters from the NVT ASCII character set.
Network Information Service (NIS+) Servers	65	4 octets minimum; multiples of 4	List of IP addresses indicating NIS+ servers available to the client. Servers should be in order of preference.
Mobile IP Home Agent	68	0 octet minimum; multiples of 4; expected, 4 octets containing a single home agent address	List of IP addresses indicating mobile IP home agents available to the client. Agents should be in order of preference. Value: 32-bit address; 0=no home agents available
Simple Mail Transport Protocol (SMTP) Server	69	4 octets minimum; multiples of 4	List of SMTP servers available to the client. Servers should be in order of preference.
Post Office Protocol (POP3) Server	70	4 octets minimum; multiples of 4	List of POP3 servers available to the client. Servers should be in order of preference.
Network News Transport Protocol (NNTP) Server	71	4 octets minimum; multiples of 4	List of NNTP servers available to the client. Servers should be in order of preference.
Default World Wide Web (WWW) Server	72	4 octets minimum; multiples of 4	List of World Wide Web (WWW) servers available to the client. Servers should be in order of preference.
Default Finger Server	73	4 octets minimum; multiples of 4	List of Finger servers available to the client. Servers should be in order of preference.
Default Internet Relay Chat (IRC) Server	74	4 octets minimum; multiples of 4	List of IRC servers available to the client. Servers should be in order of preference.
StreetTalk Server	75	4 octets minimum; multiples of 4	List of StreetTalk servers available to the client. Servers should be in order of preference.
StreetTalk Directory Assistance (STDA) Server	76	4 octets minimum; multiples of 4	List of STDA servers available to the client. Servers should be in order of preference.

DHCPv4 Extension Options

The table below lists the DHCPv4 extension options.

Table 62: DHCPv4 Extensions

Option Name	No.	Length	Description
Requested IP Address	50	4 octets	Used in a client request (DHCPDISCOVER) to allow the client to request that a particular IP address be assigned.
IP Address Lease Time	51	4 octets	Used in a client request (DHCPDISCOVER or DHCPREQUEST) to allow the client to request a lease time for the IP address. In a server reply (DHCPOFFER), a DHCP server uses this option to specify the lease time it is willing to offer. Value: seconds, as 32-bit unsigned integer
Option Overload	52	1 octet	Indicates that the DHCP sname or file fields are being overloaded by using them to carry DHCP options. A DHCP server inserts this option if the returned parameters will exceed the usual space allotted for options. If this option is present, the client interprets the specified additional fields after it concludes interpretation of the standard option fields. Values: 1=file field is used to hold options; 2=sname field is used to hold options; 3=both fields are used to hold options
DHCP Message Type	53	1 octet	Used to convey the type of DHCP message. The preset value is 1 (DHCPDISCOVER). Values: 1=DHCPDISCOVER; 2=DHCPOFFER; 3=DHCPREQUEST; 4=DHCPDECLINE; 5=DHCPACK; 6=DHCPNAK; 7=DHCPRELEASE; 8=DHCPINFORM; 13=LEASEQUERY
Server Identifier	54	4 octets	Used in DHCPOFFER and DHCPREQUEST messages, and can optionally be included in the DHCPACK and DHCPNAK messages. DHCP servers include this option in the DHCPOFFER in order to allow the client to distinguish between lease offers. DHCP clients use the contents of the server identifier field as the destination address for any DHCP messages unicast to the DHCP server. DHCP clients also indicate which of several lease offers is being accepted by including this option in a DHCPREQUEST message. The identifier is the IP address of the selected server.
Parameter Request List	55	1 octet minimum	Used by a DHCP client to request values for specified configuration parameters. The list of requested parameters is specified as n octets, where each octet is a valid DHCP option code as defined in this document. The client can list the options in order of preference. The DHCP server does not have to return the options in the requested order, but must try to insert the options in the order that the client requested.

Option Name	No.	Length	Description
Message	56	1 octet minimum	Used by a DHCP server to provide an error message to a DHCP client in a DHCPNAK message in the event of a failure. A client can use this option in a DHCPDECLINE message to indicate why the client declined the offered parameters. The message consists of <i>n</i> octets of NVT ASCII text, which the client can display on an available output device.
Maximum DHCP Message Size	57	2 octets	Maximum-length DHCP message that a server is willing to accept. The length is specified as an unsigned 16-bit integer. A client can use the maximum DHCP message size option in DHCPDISCOVER or DHCPREQUEST messages, but should not use the option in DHCPDECLINE messages. Value: 576 minimum
Renewal (T1) Time Value	58	4 octets	Time interval from address assignment until the client transitions to RENEWING state. Value: seconds, as 32-bit unsigned integer
Rebinding (T2) Time Value	59	4 octets	Time interval from address assignment until the client transitions to REBINDING state. Value: seconds, as 32-bit unsigned integer
Vendor Class Identifier	60	1 octet minimum	Used by DHCP clients to optionally identify the vendor type and configuration of a DHCP client. The information is a string of <i>n</i> octets, interpreted by servers. Vendors can choose to define specific vendor class identifiers to convey particular configuration or other identification information about a client. For example, the identifier can encode the client hardware configuration. Servers not equipped to interpret the class-specific information sent by a client must ignore it (although it can be reported). Servers that respond should only use option 43 to return the vendor-specific information to the client.
Client-Identifier	61	2 octets minimum	<p>Used by DHCP clients to specify their unique identifier. DHCP servers use this value to index their database of address bindings. This value is expected to be unique for all clients in an administrative domain.</p> <p>DHCP servers should treat identifiers as opaque objects. The client identifier can consist of type-value pairs similar to the <i>htype/chaddr</i> fields. For instance, it can consist of a hardware type and hardware address. In this case, the type field should be one of the ARP hardware types defined in STD2. A hardware type of 0 (zero) should be used when the value field contains an identifier other than a hardware address (for example, a fully qualified domain name).</p> <p>For correct identification of clients, each client-identifier must be unique among the client-identifiers used on the subnet to which the client is attached. Vendors and system administrators are responsible for choosing client-identifiers that meet this requirement for uniqueness.</p>

Option Name	No.	Length	Description
TFTP Server name	66	1 octet minimum	Identifies a TFTP server when the <i>sname</i> field in the DHCP header has been used for DHCP options.
Bootfile name	67	1 octet minimum	Identifies a bootfile when the file field in the DHCP header has been used for DHCP options.
Relay Agent Information	82		Identifies the DHCP relay agent information (see RFC 3046)
iSNS	83	14 bytes minimum	Identifies the Internet Storage Name Service (see RFC 4174)
BCMCS Controller Domain	88	Variable	List of Broadcast and Multicast Service (BCMCS) controller domains (see RFC 4280)
BCMCS Address	89	4 octets minimum	List of IP addresses for the BCMCS controller (see RFC 4280)
Lease Query Client Last Transaction Time	91	4 octets	Time of the most recent access of the client sending a DHCPLEASEQUERY (see RFC 4388).
Lease Query Associated IP Addresses	92	4 octets minimum	All IP addresses associated with the client specified in a particular DHCPLEASEQUERY message (see RFC 4388).

Microsoft Client Options

The table below lists the standard Microsoft client options.

Table 63: Microsoft DHCP Client Options

Option Name	No.	Description
dhcp-lease-time	51	14 days
domain-name	15	A domain name such as cisco.com
domain-name-servers	6	IP address of the name servers
netbios-name-servers	44	WINS server address
netbios-node-type	46	Identifies the NetBIOS client type; note that Cisco Prime Network Registrar displays a warning if it is not present
routers	3	IP address of the router for this subnet

DHCPv6 Options

The table below lists the DHCPv6 options, along with their defined data types. All the option packets include at least an option length (option-len) and a variable length data field. There can also be additional parameter settings, as described in the table. Many of these options are described in RFC 3315.

Table 64: DHCPv6 Options

Cisco Prime Network Registrar Name (Type)	No.	Description
client-identifier AT_BLOB	1	DUID identifying a client between a client and a server.
server-identifier AT_BLOB	2	DUID identifying a server between a client and a server.
ia-na AT_BLOB	3	Nontemporary Addresses option with the associated parameters and addresses. Parameters are the unique ID, time the client contacts the addresses in the IA to extend the lifetime, and time the client contacts any available server to extend the lifetime of the addresses.
ia-ta AT_BLOB	4	Temporary Addresses option with the associated parameters and addresses.
iaaddr AT_BLOB	5	IPv6 addresses associated with an IA_NA or IA_TA. (The IAADDR must be encapsulated in the options field of an IA_NA or IA_TA option.) The IAADDR option includes preferred and valid lifetime fields, and the options field that encapsulates the options specific to this address.
oro AT_SHORT	6	Option Request option (ORO) that identifies a list of options in a message between a client and a server. A client can include this option in a Solicit, Request, Renew, Rebind, Confirm, or Information-request message to inform the server about options the client wants from the server. A server can include this option in a Reconfigure message to indicate which option updates the client should request.
preference AT_INT8	7	A server sends this option to a client to affect what server the client selects.
elapsed-time AT_SHORT	8	A client sends this option to a server to indicate how long the client has been trying to complete a message exchange.
relay-message AT_BLOB	9	DHCP message in a Relay-forward or Relay-reply message.

Cisco Prime Network Registrar Name (Type)	No.	Description
auth AT_BLOB	11	Authenticates the identity and contents of a DHCP message. The parameters are the authentication protocol, the authentication algorithm, the replay detection method (RDM), and the authentication information.
server-unicast AT_IP6ADDR	12	The server sends this option to a client to indicate that the client can unicast messages to the server.
status-code AT_BLOB	13	Returns a status indication related to the DHCP message or option in which it appears. The parameters are the status code and status message.
rapid-commit AT_ZEROSIZE	14	Signals use of the two-message exchange for address assignment.
user-class AT_TYPECNT	15	Clients use this option to identify the type or category of user or applications it represents. A zero type count value field followed by user data (as a blob).
vendor-class AT_VENDOR_CLASS	16	Clients use this option to identify the vendor that manufactured the hardware on which they are running.
vendor-opts AT_VENDOR_OPTS	17	Clients and servers use this option to exchange vendor-specific information. The enterprise ID for the CableLabs vendor is 4491; the suboptions for CableLabs are listed in Table 71: DHCPv6 Options , on page 457.
interface-id AT_BLOB	18	Relay agents use this option to identify the interface on which the client message is received.
reconfigure-message AT_INT8	19	The server includes this in a Reconfigure message to indicate whether the client should respond with a Renew or Information-request message.
reconfigure-accept AT_ZEROSIZE	20	Clients use this option to announce to the server whether the client is willing to accept Reconfigure messages.
sip-servers-name AT_DNSNAME	21	Domain names of the SIP outbound proxy servers for the client. See RFC 3319.
sip-servers-address AT_IP6ADDR	22	IPv6 addresses of the SIP outbound proxy servers for the client.
dns-servers AT_IP6ADDR	23	IPv6 addresses of DNS recursive name servers.

Cisco Prime Network Registrar Name (Type)	No.	Description
domain-list AT_DNSNAME	24	Domain names in the domain search list.
ia-pd AT_BLOB	25	IPv6 prefix delegation identity association and its associated parameters and prefixes. Parameters are the unique ID, time the client contacts the addresses in the IA to extend the lifetime, and time the client contacts any available server to extend the lifetime of the addresses.
iaprefix AT_BLOB	26	IPv6 prefixes associated with an IA_PD. The prefix must be encapsulated in the options field of an IA_PD option. Parameters are the valid and preferred lifetimes, prefix length, and the prefix.
nis-servers AT_IP6ADDR	27	List of IPv6 addresses of Network Information Service (NIS) servers available to the client (see RFC 3898).
nisp-servers AT_IP6ADDR	28	List of IPv6 addresses of NIS+ servers available to the client.
nis-domain-name AT_DNSNAME	29	Conveys the NIS domain name to the client.
nisp-domain-name AT_DNSNAME	30	Conveys the NIS+ domain name to the client.
sntp-servers AT_IP6ADDR	31	List of Simple Network Time Protocol (SNTP) servers available to the client (see RFC 4075).
info-refresh-time AT_TIME	32	Sets an upper bound for how long a client should wait before refreshing DHCPv6 information (see RFC 4242).
bcmcs-server-d AT_DNSNAME	33	List of BCMCS controller domains (see RFC 4280).
bcmcs-server-a AT_IP6ADDR	34	List of IPv6 addresses for the Broadcast and Multicast Service (BCMCS) controller (see RFC 4280).
geoconf-civic AT_BLOB	36	DHCP civic addresses configuration.
remote-id AT_BLOB	37	Relay agents that terminate switched or permanent circuits can add this option to identify remote hosts (see RFC 4649).
relay-agent-subscriber-id AT_BLOB	38	Allows assignment and activation of subscriber-specific actions (see RFC 4580).

Cisco Prime Network Registrar Name (Type)	No.	Description
client-fqdn AT_BLOB	39	DHCP client FQDN.
pana-agent AT_IP6ADDR	40	Carries a list of 32-bit (binary) IPv4 addresses indicating PANA Authentication Agents (PAAs) available to the PANA client (PaC)
new-posix-timezone AT_NSTRING	41	POSIX time zone, for example, EST5EDT4, M3.2.0/02:00,M11.1.0/02:00.
new-tzdb-timezone AT_NSTRING	42	POSIX time zone database name, for example, Europe/Zurich.
ero AT_SHORT	43	Relay agent Echo Request option to inform the server of the list of relay agent options to echo back.
lq-query AT_BLOB	44	Used only in a LEASEQUERY message; identifies the query being performed. The option includes the query type, link-address (or 0::0), and options to provide data needed for the query.
client-data AT_CONTAINER6	45	Encapsulates the data for a single client on a single link in a LEASEQUERY-REPLY message.
clt-time AT_TIME	46	Client last transaction time encapsulated in the <i>client-data</i> option; identifies how long ago the server last communicated with the client (in seconds).
lq-relay-data AT_BLOB	47	Used only in a LEASEQUERY-REPLY message; provides the relay agent data used when the client last communicated with the server.
lq-client-links AT_IP6ADDR	48	Used only in a LEASEQUERY-REPLY message; identifies the links on which the client has one or more bindings. It is used in reply to a query when no link-address was specified and the client is found to be on more than one link.
mip6-hnidf AT_DNSNAME	49	Defines the Home Network ID FQDN option.
mip6-vdinf AT_CONTAINER6	50	Defines the Visited Home Network Information option.
lost-server AT_DNSNAME	51	A DHCPv6 client will request a LoST server domain name in an Options Request Option (ORO) (see RFC 3315). This option contains a single domain name and must contain precisely one root label.

Cisco Prime Network Registrar Name (Type)	No.	Description
capwap_ac_v6 AT_IP6ADDR	52	Carries a list of 128-bit (binary) IPv6 addresses indicating one or more Control and Provisioning of Wireless Access Point (CAPWAP) Access Controllers (ACs) available to the Wireless Termination Point (WTP) (see RFC 5417).
relay-id AT_BLOB	53	A DHCPv6 server MAY associate Relay-ID options from Relay-Forward messages it processes with prefix delegations and/or lease bindings that result.
mos-address AT_IP6ADDR	54	Mobility Sever (MoS) IPv6 Address for DHCP v4.
mos-fqdn AT_BLOB	55	Mobility Sever (MoS) Domain Name List for DHCPv6.
ntp-server AT_BLOB	56	Serves as a container for server location information related to one Network Time Protocol (NTP) server or Simple Network Time Protocol (SNTP) server. This option can appear multiple times in a DHCPv6 message. Each instance of this option is to be considered by the NTP client or SNTP client as a server to include in its configuration. The option itself does not contain any value. Instead, it contains one or several suboptions that carry NTP server or SNTP server location.
access-domain AT_DNSNAME	57	Defines the domain name associated with the access network. This option contains a single domain name and, as such, must contain precisely one root label.
sip-ua-cs-domains AT_DNSNAME	58	Defines the list of domain names in the Session Initiation Protocol (SIP) User Agent Configuration Service Domains.
bootfile-url AT_NSTRING	59	Informs the client about a URL to a boot file.
bootfile-param AT_TYPECNT	60	Sent by the server to the client. It consists of multiple UTF-8 (see RFC3629) strings for specifying parameters for the boot file.
client-arch-type AT_SHORT	61	Provides parity with the Client System Architecture Type option (option 93) defined for DHCPv4.
nii AT_BLOB	62	Provides parity with the Client Network Interface Identifier option (option 94) defined for DHCPv4.
geoloc AT_BLOB	63	Specifies the coordinate-based geographic location of the client, to be provided by the server.

Cisco Prime Network Registrar Name (Type)	No.	Description
aftr-name AT_DNSNAME	64	Defines a fully qualified domain name of the AFTR tunnel endpoint.
erp-local-domain-name AT_DNSNAME	65	Contains the name of the local ERP domain.
rsoo AT_CONTAINER6	66	Encapsulates whatever options the relay agent wishes to provide to the DHCPv6 server.
pd-exclude AT_BLOB	67	Used to exclude exactly one prefix from a delegated prefix.
vpn-id AT_BLOB	68	Used to identify a VPN.
mip6-idinf AT_CONTAINER6	69	Used by relay agents and DHCP servers to provide information about the home network identified.
mip6-udinf AT_CONTAINER6	70	Provides information about a home network specified by the DHCP server administrator.
mip6-hnp AT_BLOB	71	Defines the prefix for a home network.
mip6-haa AT_IP6ADDR	72	Used by DHCP servers and relay agents to specify the home agent IP address.
mip6-haf AT_DNSNAME	73	Specifies the Home Agent FQDN to look up one or more A or AAAA records containing IPv4 or IPv6 addresses for the home agent, as needed.
rdnss-selection AT_BLOB	74	Informs resolvers which RDNSS can be contacted when initiating forward or reverse DNS lookup procedures.
krb-principal-name AT_BLOB	75	Sent by the client to the DHCPv6 server, which uses it to select a specific set of configuration parameters, either for a client or for a Kerberos application server.
krb-realm-name AT_NSTRING	76	Specifies to a DHCPv6 server which realm the client wants to access.
krb-default-realm-name AT_NSTRING	77	Specifies a default realm name for the Kerberos system (clients and Kerberos application servers).

Cisco Prime Network Registrar Name (Type)	No.	Description
krb-kdc AT_BLOB	78	Provides configuration information about a KDC.
client-linklayer-address AT_BLOB	79	Indicates the client link layer address.
link-address AT_IP6ADDR	80	Indicates to the server the link on which the client is located.
radius AT_BLOB	81	Provides a mechanism to exchange authorization and identification information between the DHCPv6 relay agent and DHCPv6 server.
sol-max-rt AT_TIME	82	Overrides the default value of sol-max-rt.
inf-max-rt AT_TIME	83	Overrides the default value of inf-max-rt.
addrsel AT_BLOB	84	Provides the policy table and some other configuration parameters.
addrsel-table AT_BLOB	85	Provides the Address Selection Policy Table options.
v6-pcp-server AT_IP6ADDR	86	Configures a list of IPv6 addresses of a PCP server. This option supports only single instance (RFC 7291).
dhcpv4_msg AT_BLOB	87	Carries a DHCPv4 message that is sent by the client or the server. Such messages exclude any IP or UDP headers.
dhcp4_o_dhcp6_server AT_IP6ADDR	88	Carries a list of DHCP 4o6 servers' IPv6 addresses that the client should contact to obtain IPv4 configuration.
s46-rule AT_BLOB	89	Conveys the Basic Mapping Rule (BMR) and Forwarding Mapping Rule (FMR).
s46-br AT_IP6ADDR	90	Conveys the the IPv6 address of the Border Relay.
s46-dmr	91	Conveys values for the Default Mapping Rule (DMR).
s46-v4v6bind AT_BLOB	92	Specifies the full or shared IPv4 address of the CE. The IPv6 prefix field is used by the CE to identify the correct prefix to use for the tunnel source.

Cisco Prime Network Registrar Name (Type)	No.	Description
s46-portparams AT_BLOB	93	Specifies optional port set information that MAY be provided to CEs.
s46-cont-mape AT_CONTAINER6	94	Specifies the container used to group all rules and optional port parameters for a specified domain (Softwire46 MAP-E domain).
s46-cont-mapt AT_CONTAINER6	95	Specifies the container used to group all rules and optional port parameters for a specified domain (Softwire46 MAP-T domain).
s46-cont-lw AT_CONTAINER6	96	Specifies the container used to group all rules and optional port parameters for a specified domain (Softwire46 Lightweight 4over6 domain).
4rd AT_CONTAINER6	97	Indicates the DHCPv6 option for 4rd (IPv4 Residual Deployment).
4rd-map-rule AT_BLOB	98	Indicates the Mapping-Rule Parameters of 4rd domains.
4rd-non-map-rule AT_BLOB	99	Indicates the Non-Mapping-Rule Parameters of 4rd domains.
lq-base-time AT_INT	100	Current time the message was created to be sent by the DHCPv6 server to the requestor of the Active or Bulk Leasequery if the requestor asked for the same in an Active or Bulk Leasequery request (RFC 7653).
lq-start-time AT_INT	101	Specifies a query start time to the DHCPv6 server (RFC 7653).
lq-end-time AT_INT	102	Specifies a query end time to the DHCPv6 server (RFC 7653).
captive-portal AT_NSTRING	103	Informs the client that it is behind a captive portal and provides the URI to access an authentication page (RFC 7710).
mpl-parameters AT_BLOB	104	Provides a means to distribute a configuration of an MPL Domain or a default value for all MPL Domains (a wildcard) within the network managed by the DHCP server (RFC 7774).
ani-att AT_BLOB	105	Used for exchanging the type of access technology the client uses to attach to the network (RFC 7839).
ani-network-name AT_NSTRING	106	Name of the access network to which the mobile node is attached (RFC 7839).

Cisco Prime Network Registrar Name (Type)	No.	Description
ani-ap-name AT_NSTRING	107	Name of the access point (physical device name) to which the mobile node is attached (RFC 7839).
ani-ap-bssid AT_BLOB	108	48-bit Basic SSSID (BSSID) of the access point to which the mobile node is attached (RFC 7839).
ani-operator-id AT_BLOB	109	Variable-length Private Enterprise Number (PEN) encoded in a network byte order (RFC 7839).
ani-operator-realm AT_NSTRING	110	Realm of the operator (RFC 7839).
ipv6-address-andsf AT_IP6ADDR	143	Allows the mobile node (MN) to locate an ANDSF server (RFC 7839).

Option Tables

The following tables display the DHCP options in various ways. They show the options sorted numerically, by Cisco Prime Network Registrar name, and by category.

DHCP options have a prescribed format and allowed values for their option parameters. [Table 65: DHCPv4 Options by Number](#), on page 428 lists each DHCP option and parameter type (in the Validation column). The parameter formats and allowed values come from the DHCP and Internet RFCs. All the DHCP options appear, but clients control only some, and the CLI only others.

Options by Number

The table below shows the DHCPv4 options sorted by option number, and includes the validation type. (See [Table 67: Validation Types](#), on page 446 for details on the option validation types found in the Validation column.) A **0+** in the Comments column means a repeat count of zero or more occurrences, **1+** means one or more occurrences, **2n** means multiple occurrences in multiples of 2. Comments also indicate whether the option includes suboptions, and, if so, how many.



Tip For the syntax for adding more complex option data values for suboptions, see [Adding Complex Values for Suboptions](#), on page 170.

Table 65: DHCPv4 Options by Number

No.	Cisco Prime Network Registrar Name	Protocol Name	Validation	Comments
0	pad	Pad	AT_NOLEN	

No.	Cisco Prime Network Registrar Name	Protocol Name	Validation	Comments
1	subnet-mask	Subnet Mask	AT_IPADDR	
2	time-offset	Time Offset	AT_STIME	Replaced by tz-options (RFC 4833)
3	routers	Router	AT_IPADDR	1+
4	time-servers	Time Server	AT_IPADDR	1+
5	name-servers	Name Server	AT_IPADDR	1+
6	domain-name-servers	Domain Server	AT_IPADDR	1+
7	log-servers	Log Server	AT_IPADDR	1+
8	cookie-servers	Quotes Server	AT_IPADDR	1+
9	lpr-servers	LPR Server	AT_IPADDR	1+
10	impress-servers	Impress Server	AT_IPADDR	1+
11	resource-location-servers	RLP Server	AT_IPADDR	1+
12	host-name	Host Name	AT_NSTRING	
13	boot-size	Boot File Size	AT_SHORT	
14	merit-dump	Merit Dump File	AT_NSTRING	
15	domain-name	Domain Name	AT_NSTRING	
16	swap-server	Swap Server	AT_IPADDR	
17	root-path	Root Path	AT_NSTRING	
18	extensions-path	Extension File	AT_NSTRING	
19	ip-forwarding	Forward On/Off	AT_BOOL	
20	non-local-source-routing	SrcRte On/Off	AT_BOOL	
21	policy-filters	Policy Filter	AT_IPADDR	2n
22	max-dgram-reassembly	Maximum DG Assembly	AT_SHORT	
23	Default-ip-ttl	Default IP TTL	AT_RANGEBYTE	
24	path-mtu-aging-timeout	MTU Timeout	AT_TIME	
25	path-mtu-plateau-tables	MTU Plateau	AT_RANGESHORT	1+

No.	Cisco Prime Network Registrar Name	Protocol Name	Validation	Comments
26	interface-mtu	MTU Interface	AT_RANGESHORT	
27	all-subnets-local	MTU Subnet	AT_BOOL	
28	broadcast-address	Broadcast Address	AT_IPADDR	
29	perform-mask-discovery	Mask Discovery	AT_BOOL	
30	mask-supplier	Mask Supplier	AT_BOOL	
31	router-discovery	Router Discovery	AT_BOOL	
32	router-solicitation-address	Router Request	AT_IPADDR	
33	static-routes	Static Route	AT_IPADDR	2n
34	trailer-encapsulation	Trailers	AT_BOOL	
35	arp-cache-timeout	ARP Timeout	AT_TIME	
36	ieee802.3-encapsulation	Ethernet	AT_BOOL	
37	default-tcp-ttl	Default TCP TTL	AT_RANGEBYTE	
38	tcp-keepalive-interval	Keepalive Time	AT_TIME	
39	tcp-keepalive-garbage	Keepalive Data	AT_BOOL	
40	nis-domain	NIS Domain	AT_NSTRING	
41	nis-servers	NIS Servers	AT_IPADDR	1+
42	ntp-servers	NTP Servers	AT_IPADDR	1+
43	vendor-encapsulated-options	Vendor Specific	AT_BLOB	NM
44	netbios-name-servers	NetBIOS Name Server	AT_IPADDR	1+
45	netbios-dd-servers	NetBIOS Distribution Server	AT_IPADDR	1+
46	netbios-node-type	NetBIOS Node Type	AT_RANGEBYTE	
47	netbios-scope	NetBIOS Scope	AT_NSTRING	
48	font-servers	X Window Font	AT_IPADDR	1+
49	x-display-managers	X Window Manager	AT_IPADDR	1+

No.	Cisco Prime Network Registrar Name	Protocol Name	Validation	Comments
50	dhcp-requested-address	Address Request	AT_IPADDR	
51	dhcp-lease-time	Address Time	AT_TIME	NM
52	dhcp-option-overload	Overload	AT_OVERLOAD	
53	dhcp-message-type	DHCP Message Type	AT_MESSAGE	NM (See the DHCP Message Type option in Table 62: DHCPv4 Extensions , on page 417)
54	dhcp-server-identifier	DHCP Server ID	AT_IPADDR	
55	dhcp-parameter-request-list	Parameter List	AT_INT8	0+
56	dhcp-message	DHCP Message	AT_NSTRING	NM
57	dhcp-max-message-size	DHCP Maximum Message Size	AT_SHORT	NM
58	dhcp-renewal-time	Renewing Time	AT_TIME	NM
59	dhcp-rebinding-time	Rebinding Time	AT_TIME	NM
60	dhcp-class-identifier	Class Identifier	AT_NSTRING	
61	dhcp-client-identifier	Client Identifier	AT_BLOB	
62	netwareip-domain	NetWare/IP Domain	AT_NSTRING	
63	netwareip-information	NetWare/IP Option	AT_BLOB	
64	nis+-domain	NIS Domain Name	AT_NSTRING	
65	nis+-servers	NIS Server Address	AT_IPADDR	1+
66	tftp-server	TFTP Server Name	AT_NSTRING	
67	boot-file	Bootfile Name	AT_NSTRING	
68	mobile-ip-home-agents	Mobile IP Home Agent	AT_IPADDR	0+
69	smtp-servers	SMTP Server	AT_IPADDR	1+
70	pop3-servers	POP3 Server	AT_IPADDR	1+
71	nntp-servers	NNTP Server	AT_IPADDR	1+
72	www-servers	WWW Server	AT_IPADDR	1+

No.	Cisco Prime Network Registrar Name	Protocol Name	Validation	Comments
73	finger-servers	Finger Server	AT_IPADDR	1+
74	irc-servers	IRC Server	AT_IPADDR	1+
75	streettalk-servers	StreetTalk Server	AT_IPADDR	1+
76	streettalk-directory-assistance-servers	STDA Server	AT_IPADDR	1+
77	dhcp-user-class-id	User Class ID	AT_TYPECNT	Suboptions (2)
78	slp-directory-agent	Service Location Protocol Directory Agent	AT_BLOB	Suboptions (2)
79	slp-service-scope	SLP Service Scope	AT_BLOB	Suboptions (2)
80	rapid-commit	Rapid Commit	AT_ZEROSIZE	
81	client-fqdn	Client FQDN	AT_BLOB	Suboptions (4)
82	relay-agent-info	Relay Agent Information	AT_BLOB	For suboptions, see Table 70: DHCPv4 and BOOTP Options , on page 450
83	iSNS	Internet Storage Name Service (RFC4174)	AT_BLOB	Suboptions (7)
85	nds-servers	NDS Servers	AT_IPADDR	1+
86	nds-tree	NDS Tree Name	AT_NSTRING	
87	nds-context	NDS Context	AT_NSTRING	
88	bcmcs-servers-d	BCMCS Controller Domain (RFC 4280)	AT_DNSNAME	1+
89	bcmcs-servers-a	BCMCS Address	AT_IPADDR	1+
90	authentication	Authentication	AT_BLOB	Suboptions (5)
91	lq-client-last-transaction-time	Lease Query Client Last Transaction Time	AT_TIME	
92	lq-associated-ip	Lease Query Associated IP Addresses	AT_IPADDR	1+

No.	Cisco Prime Network Registrar Name	Protocol Name	Validation	Comments
93	pxe-client-arch		AT_SHORT	
94	pxe-client-network-id		AT_BLOB	Suboptions (2)
95	ldap-url		AT_NSTRING	
97	pxe-client-machine-id		AT_BLOB	Suboptions (2)
98	user-auth		AT_NSTRING	
99	geoconf-civic	Civic Addresses Configuration	AT_BLOB	
100	posix-timezone	IEEE 1003.1 String	AT_NSTRING	
101	tzdb-timezone	Time Zone Database	AT_NSTRING	
112	netinfo-parent-server-addr		AT_IPADDR	
113	netinfo-parent-server-tag		AT_NSTRING	
114	initial-url		AT_NSTRING	
116	auto-configure	Autoconfiguration	AT_RANGEBYTE	
117	name-service-search	Name Service Search	AT_SHORT	1+
118	subnet-selection	Subnet Selection	AT_IPADDR	
119	domain-search	Domain Search	AT_DNSNAME	1+
120	sip-servers	SIP Servers	AT_BLOB	Suboptions (2)
121	classless-static-route	Classless Static Route	AT_BLOB	
122	cablelabs-client-configuration	CableLabs Client Configuration	AT_BLOB	Suboptions (10) (see the Table 70: DHCPv4 and BOOTP Options , on page 450)
123	geo-conf	GeoConf Option	AT_BLOB	
124	v-i-vendor-class	Vendor-Identifying Vendor Class	AT_VENDOR_CLASS	NM

No.	Cisco Prime Network Registrar Name	Protocol Name	Validation	Comments
125	v-i-vendor-opts	Vendor-Identifying Vendor-Specific Info	AT_VENDOR_OPTS	See also the cablelabs-125 suboptions in Table 70: DHCPv4 and BOOTP Options , on page 450
128	mcons-security-server	--	AT_IPADDR	
136	pana-agent		AT_IPADDR	1+
137	lost-server		AT_DNSNAME	
138	capwap-ac-v4		AT_IPADDR	1+
139	mos-address		AT_BLOB	Suboptions (3) 0+
140	mos-fqdn		AT_BLOB	Suboptions (3) 0+
141	sip-ua-cs-domains		AT_DNSNAME	0+
142	andsf-v4	Access Network Discovery and Selection Function	AT_IPADDR	
144	geoloc	Geospatial Location with Uncertainty	AT_BLOB	
145	forcerenew-nonce-capable	Forcerenew Nonce Authentication	AT_INT8	1+
146	rdnss-selection	RDNSS Selection	AT_BLOB	Suboptions (4)
150	tftp-server-address	TFTP	AT_IPADDR	1+
151	status-code		AT_BLOB	Suboptions (2)
152	base-time		AT_DATE	
153	start-time-of-state		AT_TIME	
154	query-start-time		AT_DATE	
155	query-end-time		AT_DATE	
156	dhcp-state		AT_INT8	
157	data-source		AT_INT8	

No.	Cisco Prime Network Registrar Name	Protocol Name	Validation	Comments
158	v4-pcp-server	Port Control	AT_BLOB	Suboptions (2)
159	v4-portparams		AT_BLOB	Suboptions (3)
160	captive-portal	DHCP Captive-Portal	AT_NSTRING	
161	cisco-leased-ip	Cisco	AT_IPADDR	
162	cisco-client-requested-host-name	Cisco	AT_NSTRING	
163	cisco-client-last-transaction-time	Cisco	AT_INT	
185	vpn-id	VPN Identifier	AT_BLOB	NM: Suboptions (2)
209	pxlinux-config-file		AT_NSTRING	
210	pxlinux-path-prefix		AT_NSTRING	
211	prelinux-reboot-time		AT_TIME	
212	6rd		AT_BLOB	Suboptions (4)
213	access-domain		AT_NSTRING	
220	subnet-alloc	Subnet Allocation	AT_TIME	Suboptions (5)
221	cisco-vpn-id	Cisco VPN Identifier	AT_NSTRING	Suboptions (2)
251	cisco-auto-configure	Cisco Autoconfiguration	AT_RANGEBYTE	
255	end	End	AT_NOLEN	NM

Options by Cisco Prime Network Registrar Name

The table below lists the DHCP options by Cisco Prime Network Registrar name. (For each option validation type, cross-reference it by number to [Table 65: DHCPv4 Options by Number](#), on page 428 and check the Validation column.)

Table 66: DHCP Options by Cisco Prime Network Registrar Name

Cisco Prime Network Registrar Name	No.	Option Name	Category
4rd	97	IPv4 Residual Deployment via IPv6 (4rd)	DHCPv6

Cisco Prime Network Registrar Name	No.	Option Name	Category
4rd-map-rule	98	4rd Map Rule	DHCPv6
4rd-non-map-rule	99	4rd Non Map Rule	DHCPv6
6rd	212	IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)	DHCPv4
access-domain	213	Access Network Domain Name	DHCPv4
access-domain	57	Access Network Domain Name	DHCPv6
addrsel	84	Address Selection	DHCPv6
addrsel-table	85	Address Selection Policy Table	DHCPv6
aftr-name	64	AFTR tunnel endpoint domain name	DHCPv6
all-subnets-local	27	All Subnets Are Local	Interface
andsf-v4	142	ANDSF IPv4 Address for DHCPv4	DHCPv4
ani-ap-bssid	108	DHCPv6 Access-Point-BSSID	DHCPv6
ani-ap-name	107	DHCPv6 Access-Point-Name	DHCPv6
ani-att	105	DHCPv6 Access-Technology-Type	DHCPv6
ani-network-name	106	DHCPv6 Network-Name	DHCPv6
ani-operator-id	109	DHCPv6 Operator-Identifier	DHCPv6
ani-operator-realm	110	DHCPv6 Operator-Realm	DHCPv6
arp-cache-timeout	35	ARP Cache Timeout	Interface
associated-ip	92	Lease Query Associated IP	DHCPv4
auth	11	Authentication	DHCPv6
authentication	90	Authentication	--
auto-configure	116	Auto-Configuration	DHCPv4
base-time	152	base-time	DHCPv4
bcmcs-server-a	34	BCMCS Address v6	DHCPv6
bcmcs-server-d	33	BCMCS Controller Domain v6	DHCPv6
bcmcs-servers-a	89	BCMCS Address	DHCPv4
bcmcs-servers-d	88	BCMCS Controller Domain	DHCPv4
boot-file	67	Bootfile Name	BOOTP

Cisco Prime Network Registrar Name	No.	Option Name	Category
boot-size	13	Boot File Size	BOOTP
bootfile-param	60	Boot File Parameters	DHCPv6
bootfile-url	59	Boot File Uniform Resource Locator (URL)	DHCPv6
broadcast-address	28	Broadcast Address	Interface
captive-portal	103	Captive-Portal DHCPv6	DHCPv6
captive-portal	160	Captive-Portal DHCPv4	DHCPv4
capwap_ac_v4	138	CAPWAP AC	DHCPv4
capwap_ac_v6	52	CAPWAP AC	DHCPv6
cisco-auto-configure	251	Cisco Autoconfiguration	DHCPv4
cisco-client-last-transaction-time	163	Cisco Client Last Transaction Time	DHCPv4
cisco-client-requested-host-name	162	Cisco Client Requested Host Name	DHCPv4
cisco-leased-ip	161	Cisco Leased IP Address	DHCPv4
cisco-vpn-id	221	Cisco VPN Identifier	DHCPv4
classless-static-route	121	Classless Static Route	DHCPv4
client-arch-type	61	Client System Architecture Type	DHCPv6
client-data	45	Leasequery Reply Client Data	DHCPv6
client-fqdn	81	DHCP Client FQDN	DHCPv4
client-fqdn	39	DHCP Client FQDN	DHCPv6
client-identifier	1	Client Identifier	DHCPv6
client-last-transaction-time	91	Leasequery Client Last Transaction Time	DHCPv4
client-linklayer-address	79	DHCPv6 Client Link-Layer Address	DHCPv6
clt-time	46	Leasequery Client Last Transaction Time	DHCPv6
cookie-servers	8	Cookie Server	BOOTP
data-source	157	data-source	DHCPv4
default-ip-ttl	23	Default IP Time-to-Live	Host IP

Cisco Prime Network Registrar Name	No.	Option Name	Category
default-tcp-ttl	37	TCP Default TTL	Interface
dhcp-class-identifier	60	Vendor Class Identifier	DHCPv4
dhcp-client-identifier	61	Client-Identifier	Basic
dhcp-lease-time	51	IP Address Lease Time	Lease Information, MS DHCP Client
dhcp-max-message-size	57	Maximum DHCP Message Size	DHCPv4
dhcp-message	56	Message	DHCPv4
dhcp-message-type	53	DHCP Message Type	DHCPv4
dhcp-option-overload	52	Option Overload	DHCPv4
dhcp-parameter-request-list	55	Parameter Request List	DHCPv4
dhcp-rebinding-time	59	Rebinding (T2) Time Value	Lease Information, MS DHCP Client
dhcp-renewal-time	58	Renewing (T1) Time Value	Lease Information, MS DHCP Client
dhcp-requested-address	50	Requested IP Address	DHCPv4
dhcp-server-identifier	54	Server Identifier	DHCPv4
dhcp-state	156	State of IP Address	DHCPv4
dhcp-user-class-id	77	User Class ID	DHCPv4
dhcp4_o_dhcp6_server	88	DHCP 4o6 Server Address	DHCPv6
dhcpv4_msg	87	DHCPv4 Message	DHCPv6
dns-servers	23	DNS Recursive Name Server	DHCPv6
domain-list	24	Domain Search List	DHCPv6
domain-name	15	Domain Name	Basic, MS DHCP Client
domain-name-servers	6	Domain Name Server	Basic, MS DHCP Client
domain-search	119	Domain Search	DHCPv4
elapsed-time	8	Elapsed Time	DHCPv6
end	255	End	--
ero	43	Relay Agent Echo Request Option	DHCPv6

Cisco Prime Network Registrar Name	No.	Option Name	Category
erp-local-domain-name	65	Local ERP domain name	DHCPv6
extensions-path	18	Extensions Path	BOOTP
finger-servers	73	Finger Server	Servers
font-servers	48	X Window System Font Server	Servers
geo-conf	123	GeoConf	DHCPv4
geoconf-civic	99	Civic Addresses Configuration	DHCPv4
geoconf-civic	36	Civic Addresses Configuration	DHCPv6
geoloc	63	Geolocation	DHCPv6
host-name	12	Host Name	Basic
ia-na	3	Identity Association for Nontemporary Addresses	DHCPv6
ia-pd	25	Prefix Delegation	DHCPv6
ia-ta	4	Identity Association for Temporary Addresses	DHCPv6
iaaddr	5	IA Address	DHCPv6
iaprefix	26	IA Prefix	DHCPv6
ieee802.3-encapsulation	36	Ethernet Encapsulation	Interface
impress-servers	10	Impress Server	BOOTP
inf-max-rt	83	Max Information-Request Timeout	DHCPv6
info-refresh-time	32	Information Refresh Time	DHCPv6
initial-url	114	URL	DHCPv4
interface-id	18	Interface Identifier	DHCPv6
interface-mtu	26	Interface MTU	Interface
ip-forwarding	19	IP Forwarding Enable/Disable	Host IP
ipv6-address-andsf	143	ANDSF IPv6 Address	DHCPv6
irc-servers	74	IRC Server	Servers
iSNS	83	iSNS	DHCPv4
krb-default-realm-name	77	Kerberos Default Realm Name	DHCPv6

Cisco Prime Network Registrar Name	No.	Option Name	Category
krb-kdc	78	Kerberos KDC	DHCPv6
krb-principal-name	75	Kerberos Principal Name	DHCPv6
krb-realm-name	76	Kerberos Realm Name	DHCPv6
ldap-url	95	Lightweight Directory Access Protocol (LDAP) Servers	DHCPv4
link-address	80	Link Address	DHCPv6
log-servers	7	Log Server	Servers
lost-server	51	Location-to-Service Translation (LoST) Server DHCPv6	DHCPv6
lost-server	137	LoST Server DHCPv4	DHCPv4
lpr-servers	9	LPR Server	Servers
lq-associated-ip	92	Leasequery Associated IP Address	DHCPv4
lq-base-time	100	Leasequery Base Time	DHCPv6
lq-client-last-transaction-time	91	Leasequery Client Transaction Time	DHCPv4
lq-client-links	48	Leasequery Client Link Reply	DHCPv6
lq-end-time	102	Leasequery End Time	DHCPv6
lq-query	44	Leasequery	DHCPv6
lq-relay-data	47	Leasequery Relay Agent Reply	DHCPv6
lq-start-time	101	Leasequery Start Time	DHCPv6
mask-supplier	30	Mask Supplier	Interface
max-dgram-reassembly	22	Maximum Datagram Reassembly Size	Host IP
mcns-security-server	128	--	Servers
mip6-haa	72	MIPv6 Home Agent Address	DHCPv6
mip6-haf	73	MIPv6 Home Agent FQDN	DHCPv6
mip6-hnidf	49	MIPv6 Home Network ID FQDN	DHCPv6
mip6-hnp	71	MIPv6 Home Network Prefix	DHCPv6
mip6-idinf	69	MIPv6 Identified Home Network Information	DHCPv6

Cisco Prime Network Registrar Name	No.	Option Name	Category
mip6-udinf	70	MIPv6 Unrestricted Home Network Information	DHCPv6
mip6-vdinf	50	MIPv6 Visited Home Network Information	DHCPv6
mobile-ip-home-agents	68	Mobile IP Home Agent	Servers
mos-address	139	MoS IPv4 Address	DHCPv4
mos-address	54	MoS IPv6 Address	DHCPv6
mos-fqdn	140	MoS Domain Name List	DHCPv4
mos-fqdn	55	MoS Domain Name List	DHCPv6
mpl-parameters	104	MPL Parameters	DHCPv6
name-servers	5	Name Server	BOOTP
name-service-search	117	Name Service Search	DHCPv4
nds-context	87	NDS Context	NetWare Client
nds-servers	85	NDS Servers	NetWare Client
nds-tree	86	NDS Tree Name	NetWare Client
netbios-dd-servers	45	NetBIOS over TCP/IP Datagram Distribution Server	WINS/NetBIOS
netbios-name-servers	44	NetBIOS over TCP/IP Name Server	WINS/NetBIOS, MS DHCP Client
netbios-node-type	46	NetBIOS over TCP/IP Node Type	WINS/NetBIOS, MS DHCP Client
netbios-scope	47	NetBIOS over TCP/IP Scope	WINS/NetBIOS, MS DHCP Client
netinfo-parent-server-addr	112	NetInfo Parent Server Address	DHCPv4
netinfo-parent-server-tag	113	NetInfo Parent Server Tag	DHCPv4
netwareip-domain	62	NetWare/IP Domain Name	NetWare Client
netwareip-information	63	NetWare/IP Information	NetWare Client
new-posix-timezone	41	POSIX time zone string	DHCPv6
new-tzdb-timezone	42	POSIX time zone database name	DHCPv6
nii	62	Client Network Interface Identifier	DHCPv6

Cisco Prime Network Registrar Name	No.	Option Name	Category
nis+-domain	64	NIS+ Domain	Servers
nis+-servers	65	Network Information Service (NIS+) Servers	Servers
nis-domain	40	NIS Domain	Servers
nis-domain-name	29	NIS Domain Name	DHCPv6
nis-servers	41	Network Information Service (NIS) Servers	Servers
nis-servers	27	NIS Servers	DHCPv6
nisp-domain-name	30	NIS+ Domain Name	DHCPv6
nisp-servers	28	NIS+ Servers	DHCPv6
nntp-servers	71	NNTP Server	Servers
non-local-source-routing	20	Non-Local Source Routing	Host IP
ntp-server	56	Message	DHCPv6
ntp-servers	42	NTP Servers	Servers
option-time	8	Option Time	DHCPv6
oro	6	Option Request Option	DHCPv6
pad	0	Pad	--
pana-agent	40	PANA Authentication Agent DHCPv6	DHCPv6
pana-agent	136	PANA Authentication Agent DHCPv4	DHCPv4
path-mtu-aging-timeout	24	Path MTU Aging Timeout	Host IP
path-mtu-plateau-tables	25	Path MTU Plateau Table	Host IP
pd-exclude	67	Prefix Exclude	DHCPv6
perform-mask-discovery	29	Perform Mask Discovery	Interface
policy-filters	21	Policy Filter	Host IP
pop3-servers	70	POP3 Server	Servers
posix-timezone	100	IEEE 1003.1 String	DHCPv4
preference	7	Preference	DHCPv6
pxe-client-arch	93	Client System Architecture Type	DHCPv4

Cisco Prime Network Registrar Name	No.	Option Name	Category
pxe-client-machine-id	97	Client Machine Identifier	DHCPv4
pxe-client-network-id	94	Client Network Interface Identifier	DHCPv4
pxelinux-config-file	209	Configuration File	DHCPv4
pxelinux-path-prefix	210	Path Prefix	DHCPv4
pxelinux-reboot-time	211	Reboot Time	DHCPv4
query-end-time	155	query-end-time	DHCPv4
query-start-time	154	query-start-time	DHCPv4
radius	81	DHCPv6 RADIUS	DHCPv6
rapid-commit	80	Rapid Commit	DHCPv4
rapid-commit	14	Rapid Commit	DHCPv6
rdnss-selection	74	RDNSS Selection DHCPv6	DHCPv6
rdnss-selection	146	RDNSS Selection DHCPv4	DHCPv4
reconfigure-accept	20	Reconfigure Accept	DHCPv6
reconfigure-message	19	Reconfigure Message	DHCPv6
relay-agent-info	82	DHCP Relay Agent Information	DHCPv4
relay-agent-subscriber-id	38	Relay Agent Subscriber ID	DHCPv6
relay-id	53	Relay ID	
relay-message	9	Relay Message	DHCPv6
remote-id	37	Relay Agent Remote ID	DHCPv6
resource-location-servers	11	Resource Location Server	BOOTP
root-path	17	Root Path	BOOTP
router-discovery	31	Perform Router Discovery	Interface
router-solicitation-address	32	Router Solicitation Address	Interface
routers	3	Router	Basic, MS DHCP Client
rsoo	66	Relay-Supplied Options	DHCPv6
s46-br	90	Softwire46 (S46) Border Relay (BR)	DHCPv6
s46-cont-lw	96	S46 Lightweight 4over6 Container	DHCPv6

Cisco Prime Network Registrar Name	No.	Option Name	Category
s46-cont-mape	94	S46 MAP-E Container	DHCPv6
s46-cont-mapt	95	S46 MAP-T Container	DHCPv6
s46-dmr	91	S46 Default Mapping Rule (DMR)	DHCPv6
s46-portparams	93	S46 Port Parameters	DHCPv6
s46-rule	89	S46 Rule	DHCPv6
s46-v4v6bind	92	S46 IPv4/IPv6 Address Binding	DHCPv6
server-identifier	2	DHCPv6 Server Identifier	DHCPv6
server-unicast	12	Server Unicast	DHCPv6
sip-servers	120	SIP Servers	DHCPv4
sip-servers-name	21	SIP Servers Domain Name List	DHCPv6
sip-servers-address	22	SIP Servers IPv6 Address List	DHCPv6
sip-ua-cs-domains	141	SIP UA Configuration Service Domains	DHCPv4
sip-ua-cs-domains	58	SIP User Agent Configuration Service Domains	DHCPv6
slp-directory-agent	78	SLP Directory Agent	DHCPv4
slp-service-scope	79	SLP Service Scope	DHCPv4
smtp-servers	69	SMTP Server	Servers
sntp-servers	31	SNTP Configuration	DHCPv6
sol-max-rt	82	SOL_MAX_RT	DHCPv6
start-time-of-state	153	start-time-of-state	DHCPv4
static-routes	33	Static Route	Interface
status-code	13	Status Code	DHCPv6
status-code	151	Status Code	DHCPv4
streettalk-directory-assistance-servers	76	STDA Server	Servers
streettalk-servers	75	StreetTalk Server	Servers
subnet-alloc	220	Subnet Allocation	DHCPv4

Cisco Prime Network Registrar Name	No.	Option Name	Category
subnet-mask	1	Subnet Mask	Basic
subnet-selection	118	Subnet Selection	DHCPv4
swap-server	16	Swap Server	BOOTP
tcp-keepalive-garbage	39	TCP Keepalive Garbage	Interface
tcp-keepalive-interval	38	TCP Keepalive Interval	Interface
tftp-server	66	TFTP Server Name	Servers
time-offset	2	Time Offset	BOOTP
time-servers	4	Time Server	BOOTP
trailer-encapsulation	34	Trailer Encapsulation	Interface
tzdb-timezone	101	TZ Database String	DHCPv4
user-auth	98	User Authentication	DHCPv4
user-class	15	User Class	DHCPv6
v-i-vendor-class	124	Vendor Identifying Vendor Class	DHCPv4
v-i-vendor-opts	125	Vendor Identifying Vendor Options	DHCPv4
v4-pcp-server	158	DHCPv4 PCP server	DHCPv4
v4-portparams	159	DHCPv4 Port Parameters	DHCPv4
v6-pcp-server	86	DHCPv6 PCP Server	DHCPv6
vendor-class	16	Vendor Class	DHCPv6
vendor-encapsulated-options	43	Vendor Specific Information	DHCPv4
vendor-opts	17	Vendor Specific Information	DHCPv6
vpn-id	68	VPN Identifier	DHCPv6
vpn-id	185	VPN Identifier	DHCPv4
www-servers	72	WWW Server	Servers
x-display-managers	49	X Window System Display Manager	Servers

Option Validation Types

The table below defines the DHCP option validation types. Note that you cannot use some of them to define custom options.

Table 67: Validation Types

Validation	Description—Web UI Equivalent
AT_BLOB	List of binary bytes—binary
AT_BOOL	Boolean—boolean
AT_CONTAINER6	DHCPv6 Option Container (not usable for custom options)
AT_DATE	Bytes representing a date—date
AT_DNSNAME	DNS name—DNS name
AT_INT	Unsigned 32-bit integer—unsigned 32-bit
AT_INT8	8-bit integer—unsigned 8-bit
AT_INTI	Unsigned 32-bit integer (Intel)—unsigned 32-bit (Intel)
AT_IPADDR	32-bit IP address—IP address
AT_IP6ADDR	128-bit IPv6 address—IPv6 address
AT_MACADDR	Bytes representing a MAC address—MAC address
AT_MESSAGE	Unsigned 8-bit message (not usable for custom options)
AT_NOLEN	No length (used for PAD and END only)
AT_NSTRING	Sequence of ASCII characters—string
AT_OVERLOAD	Overload bytes (not usable for custom options)
AT_RANGEBYTE	Range of bytes (not usable for custom options)
AT_RANGESHORT	Range of shorts (not usable for custom options)
AT_RDNSNAME	Relative DNS name—relative DNS name
AT_SHORT	Unsigned 16-bit integer—unsigned 16-bit
AT_SHRTI	Unsigned 16-bit integer (Intel)—unsigned 16-bit (Intel)
AT_SINT	Signed 32-bit integer—signed 32-bit
AT_SINT8	8-bit integer—signed 8-bit
AT_SINTI	Signed 32-bit integer (Intel)—signed 32-bit (Intel)
AT_SSHORT	Signed 16-bit integer—signed 16-bit
AT_SSHRTI	Signed 16-bit integer (Intel)—signed 16-bit (Intel)
AT_STIME	Signed 32-bit signed integer representing time—signed time
AT_STRING	Unrestricted sequence of ASCII characters—string

Validation	Description—Web UI Equivalent
AT_TIME	Unsigned 32-bit integer representing time—unsigned time
AT_TYPECNT	Type requiring two child definition: size of the type field, and type of data—counted-type: For the DHCPv4 dhcp-user-class-id option (77), the repeating pattern is: [len (1 byte)] [data, of single type] For the DHCPv6 user-class option (15), the repeating pattern is: [len (2 byte)] [data, of single type]
AT_VENDOR_CLASS	Vendor-class option (enterprise ID followed by opaque data; if DHCPv4, enterprise ID is followed by EID length)—vendor-class
AT_VENDOR_OPTS	Vendor-specific options data (enterprise ID followed by TLVs of vendor-specific data; if DHCPv4, enterprise ID is followed by EID length)—vendor-opts
AT_ZEROSIZE	32 bits of zero size (no longer used for PAD and END)



Note AT_TIME takes the value entered in seconds, by default. For example, if you enter 60, it is taken as 60 seconds and if you enter 60s/60m/2h, it is taken as 60 seconds/60 minutes/2 hours and displayed as 60s/60m/2h.



APPENDIX **B**

DHCP Extension Dictionary

This appendix describes the DHCP extension dictionary entries and the application program interface (API) to the extension dictionary. It describes the data items available in the request and response dictionaries, and the calls to use when accessing dictionaries from Tcl extensions and shared libraries.

The appendix contains the following sections;

- [Extension Dictionary Entries, on page 449](#)
- [Extension Dictionary API, on page 482](#)
- [Handling Objects and Options, on page 499](#)
- [Examples of Option and Object Method Calls, on page 501](#)

Extension Dictionary Entries

A dictionary is a data structure that contains key-value pairs. There are two types of dictionaries: the attribute dictionaries that the request and response dictionaries use, and the environment dictionary. This section describes the request and response dictionaries; the environment dictionary entries are described in [Tcl Environment Dictionary Methods, on page 486](#).

Decoded DHCP Packet Data Items

The decoded DHCPv4 packet data items represent the information in the DHCP packet, and are available in both the request and response dictionaries. These dictionaries provide access to considerably more internal server data structures than just the decoded request and decoded response.

All of the options followed by an asterisk (*) are multiple, which means that there can be more than one value associated with each option. In the DHCP/BOOTP packet, all of these data items appear in the same option. However, in the extension interface, these multiple data items are accessible through indexing.

You can access options that do not have names in [Table 70: DHCPv4 and BOOTP Options , on page 450](#) as option-*n*, where *n* is the option number. All fields are read/write. [Table 68: DHCPv4 and BOOTP Fields , on page 449](#) describes the field values for the DHCPv4 packets; [Table 69: DHCPv6 Fields , on page 450](#) describes the field values for the DHCPv6 messages.

Table 68: DHCPv4 and BOOTP Fields

Name	Value
chaddr	blob (sequence of bytes)

Name	Value
ciaddr	IP address
file	string
flags	16-bit unsigned integer
giaddr	IP address
hlen	8-bit unsigned integer
hops	8-bit unsigned integer
htype	8-bit unsigned integer
op	8-bit unsigned integer
secs	16-bit unsigned integer
siaddr	IP address
sname	string
xid	32-bit unsigned integer
yiaddr	IP address

Table 69: DHCPv6 Fields

Name	Value
hop-count	8-bit unsigned integer
link-address	IPv6 address
msg-type	8-bit unsigned integer
peer-address	IPv6 address
xid	32-bit unsigned integer

The table below lists the DHCP and BOOTP options for DHCPv4.

Table 70: DHCPv4 and BOOTP Options

Name (*=multivalue)	Number	Value
all-subnets-local	27	byte-valued boolean
authentication	90	blob (sequence of bytes); 5 fields
auto-configure	116	8-bit unsigned integer
arp-cache-timeout	35	unsigned time

Name (*=multivalued)	Number	Value
bcmcs-servers-a*	89	IP address
bcmcs-servers-d*	88	DNS name
boot-file	67	string
boot-size	13	16-bit unsigned integer
broadcast-address	28	IP address
cablelabs-125(v-i-vendor-info ID: 4491)	125	binary
oro	1	suboptions: Option request, 8-bit unsigned integer (8-bit unsigned integers)
tftp-servers	2	IP addresses of TFTP servers
erouter-container	3	Erouter container options (binary; TLV encoded options)
packetcable-mib-env	4	MIB environment indicator (8-bit enumeration)
modem-capabilities	5	Modem capabilities encoding (binary; TLV5 encoded data)
acs-server	6	ACS Server suboptions (binary)
radius-server	7	RADIUS Server suboptions (binary)
dhcpv6-servers	123	DHCPv6 server suboptions (binary)
ip-pref	124	IPv4 or IPv6 preference (8-bit enumeration)
cablelabs-client-configuration	122	blob (sequence of bytes)
		suboptions:
[ccc-]primary-dhcp-server	1	IP address
[ccc-]secondary-dhcp-server	2	IP address
[ccc-]provisioning-server	3	blob (the first byte must be the type byte, with 0 for RFC 1035 encoding, and 1 for IP address encoding, for which the address must be in network order)
[ccc-]as-backoff-retry-blob	4	12-byte blob (3 unsigned 4-byte integers, which must be in network order); configures the Kerberos AS-REQ/AS-REP timeout, back-off, and retry mechanism

Name (*=multivalued)	Number	Value
[ccc-]ap-backoff-retry- blob	5	12-byte blob (3 unsigned 4-byte integers, which must be in network order); configures the Kerberos AP-REQ/AP-REP timeout, back-off, and retry mechanism
[ccc-]kerberos-realm	6	variable-length blob (an RFC 1035 style name); a Kerberos realm name is required
[ccc-]use-tgt	7	1-byte unsigned integer boolean; indicates whether to use a Ticket Granting Ticket (TGT) when obtaining a service ticket for one of the application servers
[ccc-]provisioning-timer	8	1-byte unsigned integer; defines the maximum time allowed for the provisioning process to finish
[ccc-]ticket-control- mask	9	2-byte unsigned integer, in host order
[ccc-]kdc-addresses- blob	10	variable-length (multiple of 4) IP address, in network order
cisco-autoconfigure	251	bounded byte
cisco-client-last-transaction- time	163	unsigned 32-bit integer
cisco-client-requested-host- name	162	string
cisco-leased-ip	161	IP address
cisco-vpn-id	221	blob (structured)
classless-static-route	121	blob (structured)
client-fqdn	81	blob (sequence of bytes); 4 fields: flags, rcode-1, rcode-2, and domain-name
cookie-servers*	8	IP address
default-ip-ttl	23	8-bit unsigned integer
default-tcp-ttl	37	8-bit unsigned integer
dhcp-class-identifier	60	string
dhcp-client-identifier	61	blob (sequence of bytes)
dhcp-lease-time	51	unsigned time
dhcp-max-message-size	57	16-bit unsigned integer
dhcp-message	56	string
dhcp-message-type	53	8-bit unsigned integer

Name (*=multivalued)	Number	Value
dhcp-option-overload	52	8-bit unsigned integer
dhcp-parameter-request-list*	55	8-bit unsigned integer
dhcp-parameter-request-list-blob*	55	blob (sequence of bytes)
dhcp-rebinding-time	59	unsigned time
dhcp-renewal-time	58	unsigned time
dhcp-requested-address	50	IP address
dhcp-server-identifier	54	IP address
dhcp-user-class-id	77	set of counted len byte arrays; 2 fields: typcnt-size and user-data
domain-name	15	string
domain-name-servers*	6	IP address
domain-search	119	blob (sequence of bytes)
extensions-path	18	string
finger-servers*	73	IP address
font-servers*	48	IP address
geo-conf	123	blob (sequence of bytes)
geoconf-civic	99	blob (sequence of bytes)
host-name	12	string
ieee802.3-encapsulation	36	byte-valued boolean
impress-servers*	10	IP address
initial-url	114	string
interface-mtu	26	16-bit unsigned integer
ip-forwarding	19	byte-valued boolean
irc-servers*	74	IP address
iSNS	83	blob (sequence of bytes); 7 fields
ldap-url	95	string
log-servers*	7	IP address
lost-server	137	DNS Name (see RFC 5223)

Name (*=multivalued)	Number	Value
lpr-servers*	9	IP address
lq-associated-ip*	92	IP address
lq-client-last-transaction-time	91	unsigned time
mask-supplier	30	byte-valued boolean
max-dgram-reassembly	22	16-bit unsigned integer
mcns-security-server	128	IP address
merit-dump	14	string
mobile-ip-home-agents*	68	IP address
name-servers*	5	IP address
name-service-search*	117	16-bit unsigned integer
nds-servers*	85	IP address
nds-tree	86	string
nds-context	87	string
netbios-dd-servers*	45	IP address
netbios-name-servers*	44	IP address
netbios-node-type	46	8-bit unsigned integer
netbios-scope	47	string
netinfo-parent-server-addr	112	IP address
netinfo-parent-server-tag	113	string
netwareip-domain	62	string
netwareip-information	63	blob (sequence of bytes)
nis+-servers*	65	IP address
nis+domain	64	string
nis-domain	40	string
nis-servers*	41	IP address
nntp-servers*	71	IP address
non-local-source-routing	20	byte-valued boolean
ntp-servers*	42	IP address

Name (*=multivalued)	Number	Value
pana agent	136	IP address(es) (see RFC 5192)
path-mtu-aging-timeout	24	unsigned time
path-mtu-plateau-tables*	25	16-bit unsigned integer
perform-mask-discovery	29	byte-valued boolean
policy-filters*	21	IP address (there can be two policy filters, each one having its own IP address)
pop3-servers*	70	IP address
posix-timezone	100	string (see RFC 4833)
pxe-client-arch	93	16-bit unsigned integer
pxe-client-machine-id	97	blob (sequence of bytes); 2 fields: type-flag and uuid
pxe-client-network-id	94	blob (sequence of bytes); 2 fields: type-flag and version
rapid-commit	80	null-length
relay-agent-info suboptions:	82	blob (sequence of bytes)
	suboptions:	
relay-agent-circuit-id- data	1	blob (sequence of bytes)
relay-agent-remote-id- data	2	blob (sequence of bytes)
relay-agent-device- class-data	4	4-byte unsigned integer
relay-agent-subnet- selection-data	5	IP address
subscriber-id	6	string identifying the network client or subscriber
radius-attributes	7	supported attributes are user, class, and framed-pool
authentication	8	binary
v-i-vendor-opts	9	vendor options
cisco-subnet-selection	150	IP address
cisco-vpn-id	151	binary
cisco-server-id-override	152	IP address
relay-agent-vpn-id-data	181	string

Name (*=multivalued)	Number	Value
relay-agent-server-id-override-data	182	IP address
Note The relay-agent-circuit-id, relay-agent-remote-id, and relay-agent-device-class suboptions, which returned the two bytes (suboption code and data length) preceding the suboption data, are deprecated, but still available.		
resource-location-servers*	11	IP address
root-path	17	string
router-discovery	31	byte-valued boolean
router-solicitation-address	32	IP address
routers*	3	IP address
sip-servers	120	blob (sequence of bytes); 2 fields: flag and sip-server-list
slp-directory-agent*	78	blob (sequence of bytes); 2 fields: mandatory and agent-ip-list
slp-service-scope*	79	blob (sequence of bytes); 2 fields: mandatory and slp-scope-list
smtp-servers*	69	IP address
static-routes*	33	IP address
streettalk-directory-assistance-servers*	76	IP address
streettalk-servers*	75	IP address
subnet-alloc	220	blob (sequence of bytes); 5 fields: flags, subnet-request, subnet-info, subnet-name, and subnet-suggested-lease-time
subnet-mask	1	IP address
subnet-selection	118	IP address
swap-server	16	IP address

Name (*=multivalued)	Number	Value
tcp-keepalive-internal	38	unsigned time
tcp-keepalive-garbage	39	byte-valued boolean
tftp-server	66	string
time-offset	2	signed time
time-servers*	4	IP address
trailer-encapsulation	34	byte-valued boolean
tzdb-timezone	101	string (see RFC 4833)
user-auth	98	string
vendor-encapsulated-options	43	blob (sequence of bytes)
v-i-vendor-class	124	blob (sequence of bytes)
v-i-vendor-info	125	blob (sequence of bytes)
vpn-id	185	blob (structured); 2 fields: flag and vpn-id
www-servers*	72	IP address
x-display-managers*	49	IP address

The table below lists the DHCPv6 options.



Note Access to these options is available using the `putOption`, `getOption`, and `removeOption` methods only.

Table 71: DHCPv6 Options

Name (*=multivalued)	Number	Value
auth	11	binary; 5 fields: protocol, algorithm, replay-detection-method, replay-detection, and auth-info
bcmcs-server-a*	34	IPv6 address
bcmcs-server-d*	33	DNS name
cablelabs-17(vendor-opts ID: 4491)	17 suboptions:	vendor-opts; 27 suboptions
oro	1	16-bit unsigned integer
device-type	2	string
embedded-components-list	3	string

Name (*=multivalue)	Number	Value
device-serial-number	4	string
hardware-version-number	5	string
software-version-number	6	string
boot-rom-version	7	string
vendor-oui	8	string
model-number	9	string
vendor-name	10	string
ecm-cfg-encaps	15	string
tftp-servers	32	IPv6 address
config-file-name	33	string
syslog-servers	34	IPv6 address
modem-capabilities	35	binary
device-id	36	binary
rfc868-servers	37	IPv6 address
time-offset	38	signed time
ip-pref	39	8-bit unsigned integer
acs-server	40	binary; 2 suboptions
	suboptions:	
flag	0	8-bit unsigned integer
server	0	
radius-server	41	binary; 2 suboptions
	suboptions:	
flag	0	8-bit unsigned integer
server	0	
cmts-capabilities	1025	binary
cm-mac-address	1026	binary
erouter-container	1027	binary

Name (*=multivalued)	Number	Value
cablelabs-client-configuration	2170 suboptions:	binary; 2 suboptions (various data types)
primary-dhcp-server	1	IP address
secondary-dhcp-server	2	IP address
cablelabs-client-configuration-v6	2171 suboptions:	binary; 9 suboptions (various data types)
primary-dhcpv6-server-selector-id	1	binary
secondary-dhcpv6-server-selector-id	2	binary
provisioning-server	3	binary
as-backoff-retry	4	binary
ap-backoff-retry	5	binary
kerberos-realm	6	DNS name
use-tgt	7	unsigned 8-bit
provisioning-timer	8	unsigned 8-bit
ticket-control-mask	9	unsigned 16-bit
cablelabs-correlation-id	2172	unsigned 32-bit
client-data	45	binary (options) (see RFC 5007)
client-fqdn	39	binary; 2 fields: flags and domain-name
client-identifier	1	binary
clt-time	46	32-bit unsigned time (see RFC 5007)
dns-servers*	23	IPv6 address
domain-list*	24	DNS name
elapsed-time	8	unsigned 16-bit
ero	43	unsigned 16-bit (see RFC 4994)
geoconf-civic	36	binary
ia-na	3	binary; 3 fields: ia-id, t1, and t2
ia-pd	25	binary; 3 fields: ia-id, t1, and t2

Name (*=multivalued)	Number	Value
ia-prefix	26	binary; 4 fields: preferred-lifetime, valid-lifetime, prefix-length, and prefix
ia-ta	4	binary; 1 suboption: iaid
iaaddr	5	binary; 3 fields: address, preferred-lifetime, and valid-lifetime
info-refresh-time	32	unsigned time
interface-id	18	binary
lost-server	51	DNS Name (see RFC 5223)
lq-client-links	48	IPv6 address(es) (see RFC 5007)
lq-query	44	binary structured (see RFC 5007)
lq-relay-data	47	binary (DHCPv6 message) (see RFC 5007)
new-posix-timezone	41	string (RFC 4833)
new-tzdb-timezone	42	string (RFC 4833)
nis-domain-name*	29	DNS name
nis-servers*	27	IP address
nisp-domain-name*	30	DNS name
nisp-servers*	28	IP address
oro*	6	unsigned 16-bit
pana agent	40	IPv6 address(es) (see RFC 5192)
preference	7	unsigned 8-bit
rapid-commit	14	zero size
reconfigure-accept	20	zero size
reconfigure-message	19	unsigned 8-bit
relay-agent-subscriber-id	38	binary
relay-message	9	binary
remote-id	37	binary; 2 fields: enterprise-id and remote-id
server-identifier	2	binary (AT_BLOB)
server-unicast	12	IPv6 address
sip-servers-address*	22	IPv6 address

Name (*=multivalue)	Number	Value
sip-servers-name*	21	DNS name
sntp-servers*	31	IP address
status-code	13	binary; 2 fields: status-code and status-message
user-class*	15	counted-type; 2 fields: typecnt-size and user-data
vendor-class	16	vendor-class
vendor-opts	17	vendor-opts (see also cablelabs-17)

Request Dictionary

The table below lists the data items that you can set in the request dictionary at any time. The DHCP server reads them at various times. Unless indicated otherwise, all operations are read/write.

Table 72: Request Dictionary Specific Data Items

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>active-leasequery-control</i>	int (v4,v6)
Controls the sending of a lease (such as only on specific state changes). Values are: 0—unspecified (the server determines whether to send the notification), 1—send (the server will send the notification), and 2—do not send (the server will not send the notification). The <i>active-leasequery-control</i> is initialized as 0, that is, unspecified.	
<i>allow-bootp</i>	int (v4)
If set to 1, allows BOOTP for any scope for this request. Read during scope selection and while checking for lease acceptability.	
<i>allow-dhcp</i>	int (v4)
If set to a 1, allows DHCP for any scope for this request. Read during scope selection and while checking for lease acceptability.	
<i>allow-dynamic-bootp</i>	int (v4)
If set to a 1, allows dynamic BOOTP for any scope for this request. Read during scope selection and while checking for lease acceptability.	
<i>bootp-reply-options</i>	blob (v4)
Overrides any <i>v4-bootp-reply-options</i> in any policy; read when gathering data for the output packet. (There are no IPv6 bootp-reply-options.)	
<i>client-class-name</i>	string (v4, v6)
Name of the client-class used to complete the client information (if any). Read-only.	

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>client-class-policy</i>	string (v4, v6)
Name of the policy that is associated with the client-class. If set, it must be with the name of a policy that was already configured in the server.	
<i>client-domain-name</i>	string (v4, v6)
Domain name that the client wants to use. If it does not exist, in which case the DHCP server uses the domain name specified in the scope. Read when queuing the request for DNS update just prior to the update of stable storage. For DHCPv6, overrides the <i>client-fqdn</i> value and used for DNS updates.	
<i>client-host-name</i>	string (v4, v6)
Hostname for the client in DNS; read when queuing in the request for a DNS update just before updating stable storage. Places the actual name in DNS when that operation finishes. For DHCPv6, overrides the <i>client-fqdn</i> value and used for DNS updates.	
<i>client-id</i>	blob (v4, v6)
Client identification that the server uses to track the client. Can be the <i>client-id</i> sent with a request or internally generated from the MAC address. See <i>client-id-created-from-mac-address</i> . For DHCPv6, the Client Identifier Option value (the client's DUID).	
<i>client-id-created-from-mac-address</i>	int (v4)
If set to 1, the <i>client-id</i> must be created for internal use from the client-supplied MAC address and should not be used in reporting.	
<i>client-ipaddress</i>	IP address (v4)
IP address from which the client sent its packet. Note that it could be zero if the client does not yet have an IP address.	
<i>client-limitation-id</i>	blob (v4, v6)
Limitation ID for the client.	
<i>client-lookup-id</i>	blob (v4, v6)
Client lookup ID calculated by the <i>client-lookup-id</i> expression of the client-class.	
<i>client-mac-address</i>	blob (v4)
MAC address stored in the client object associated with the request dictionary. Has the same format (and was created from) the <i>mac-address</i> .	
<i>client-os-type</i>	int (v4)
Change the client entry of the request packet by setting this at the pre-client-lookup or post-client-lookup extension points. Can also be read at check-lease-acceptable , but cannot be set there. To set the value, you must first set the <i>os-type</i> in the post-packet-decode request dictionary.	
<i>client-packet</i>	blob (v4, v6, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
	The client portion of the received packet. For DHCPv4, this is the complete packet. For DHCPv6, this is the client message. (See packet to obtain the full packet.)
<i>client-policy</i>	string (v4, v6)
	Name of the policy that is associated with the client entry. If set, must be the name of a preconfigured policy in the DHCP server.
<i>client-port</i>	int (v4, v6)
	Port from which the client sent its request.
<i>client-requested-host-name</i>	string (v4)
	Hostname that the client requested be used for the DNS update. The DHCP server saves this information so that a change can be detected.
<i>client-unicast</i>	boolean (v6, read-only)
	True if the received packet was unicast by the client to the server.
<i>client-wants-nulls-in-strings</i>	int (v4)
	Determines whether the DHCP server returns strings to the client terminated with a null. If set to 1, the server terminates strings with a null. If set to 0, it does not terminate strings with a null. Set before post-packet-decode and read when encoding the response packet after pre-packet-encode .
<i>derived-vpn-id</i>	int (v4, v6, read-only)
	VPN identifier. See <i>vpn-name</i> for details.
<i>destination-ipaddress</i>	IP address (v6, read-only)
	Destination IPv6 address of the packet.
<i>dhcp-reply-options</i>	blob (v4, v6)
	Overrides any <i>v4-reply-options</i> or <i>v6-reply-options</i> specified in a policy; read when gathering data for the output packet.
<i>dump-packet</i>	int (v4, v6, write-only)
	When set to 1, Cisco Prime Network Registrar dumps the current decoded DHCP/BOOTP packet to the log file. An extension can put the value 1 into this data item at multiple points in its execution. This might be useful when debugging extensions.
<i>failover-role</i>	int (v4, v6, read only)
	Determines the failover server role. The failover server role can be one of three values: <ul style="list-style-type: none"> • None—Failover is not configured. • Main/Backup—Failover is configured and the role of the failover server

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>failover-state</i>	int (v4, v6, read only)
Determines failover server state. The failover state can be normal, partner-down, communications-interrupted, recover, potential-conflict, recover-done, startup, shutdown, or paused. If failover is not configured the value is none.	
<i>import-packet</i>	int (v4)
Determines whether the server treats the packet as if it came from an import client. If set to 1, the server treats the client as an import client and performs all DNS operations on it before sending an ACK. Read when checking the server import mode (right after post-packet-decode), getting ready for DNS processing, and when setting the reply address.	
<i>limitation-count</i>	int (v4)
Number of simultaneous users allowed with the same <i>limitation-id</i>	
<i>limitation-id</i>	blob (v4)
Calculated by the <i>limitation-id</i> expression (if any) for the client-class in which this request falls	
<i>limitation-id-null</i>	int (v4, v6)
Set to 1(TRUE) if the <i>limitation-id</i> is null, 0 (FALSE) if another value.	
<i>log-client-criteria-processing</i>	int (v4, v6)
If set to a 1, logs the criteria processing for the client for this request. Read when trying to acquire a new lease for a client that does not have one, and when checking for lease acceptability.	
<i>log-client-detail</i>	int (v4, v6)
If set to a 1, logs the client-class processing for this request. Read at the end of client-class processing, after post-client-lookup .	
<i>log-dns-update-detail</i>	int (v4, v6)
If set to a 1, logs DNS update details for this request.	
<i>log-dropped-bootp-packets</i>	int (v4)
If set to a 1, logs dropped BOOTP packets for this request.	
<i>log-dropped-dhcp-packets</i>	int (v4, v6)
If set to a 1, logs dropped DHCP packets for this request.	
<i>log-dropped-waiting-packets</i>	int (v4, v6)
If set to a 1, logs dropped waiting packets for this request.	
<i>log-failover-detail</i>	int (v4)
If set to a 1, logs a more detailed level of failover activity, such as all failover state changes.	

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>log-incoming-packet-detail</i>	int (v4, v6)
If set to a 1, checks whether detailed incoming packet tracing occurred for this request, so that you do not need to put a separate trace on it. Read before packet decoding and the first extension point.	
<i>log-incoming-packets</i>	int (v4, v6)
If set to a 1, logs the incoming packets for this request. Read after post-decode-packet	
<i>log-ldap-create-detail</i>	int (v4)
If set to a 1, logs messages whenever the DHCP server initiates a lease state entry creation to, receives a response from, or retrieves a result or error message from an LDAP server.	
<i>log-ldap-query-detail</i>	int (v4, v6)
If set to a 1, logs messages whenever the DHCP server initiates a query to, receives a response from, or retrieves a query result or an error message from an LDAP server.	
<i>log-ldap-update-detail</i>	int (v4)
If set to a 1, logs messages whenever the DHCP server initiates an update lease state to, receives a response from, or a retrieves a result or error message from an LDAP server.	
<i>log-leasequery</i>	int (v4, v6)
If set to a 1, logs messages when leasequery packets are processed without internal errors and result in an ACK or a NAK.	
<i>log-missing-options</i>	int (v4, v6)
If set to a 1, logs missing options (those a client requests but the DHCP server cannot return). Read while gathering data for the response.	
<i>log-outgoing-packet-detail</i>	int (v4, v6)
If set to a 1, logs a detailed dump of the outgoing packet for this request. Read after pre-packet-encode and just before sending the packet to the DHCP client.	
<i>log-success-messages</i>	int (v4, v6)
If set to a 1, logs the success messages.	
<i>log-unknown-criteria</i>	int (v4, v6)
If set to a 1, logs any unknown criteria specified in the client inclusion or exclusion criteria for this request. Read when acquiring a new client lease or checking lease acceptability for an existing client.	
<i>log-v6-lease-detail</i>	int (v6)
If set to 1, logs individual messages about DHCPv6 leasing activity.	
<i>mac-address</i>	blob (v4)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
	MAC address that came in the client packet. The first byte is the hardware type, the second is the hardware length, and the remaining (up to 16) is the information from the <i>chaddr</i> read just after post-packet-decode . This is a useful aggregation of the <i>htype</i> , <i>hlen</i> , and <i>chaddr</i> fields of the DHCP packet. When read it is constructed from these fields; when written it is placed into these fields.
<i>max-client-lookups</i>	integer (v4, v6)
	Maximum number of client database lookups allowed. Usually a small integer such as 2; the preset value is 1.
<i>override-client-id</i>	blob (v4, v6)
	Blob used for the current client-id value. Replaces any client-id from the incoming packet (although both values are kept in the lease state database).
<i>override-client-id-data-type</i>	string (v4, v6, read-only)
	Returns the data type of the <i>override-client-id</i> , either “nstr” for string or “blob” for blob.
<i>override-client-id-string</i>	string (v4, v6)
	Current client-id value in string format that replaces any client-id from the incoming packet (although both values are kept in the lease state database). For a <i>get</i> , if the <i>override-client-id</i> is not a string, the binary data is formatted as blob data, which is then returned as the “string.”
<i>packet</i>	blob (v4, v6)
	The received packet. For DHCPv4, this is the same as <i>client-packet</i> . For DHCPv6, this is the full packet if relayed or the same as <i>client-packet</i> if not relayed. It should only be written from the pre-packet-decode extension point; the server then decodes this new packet instead of the packet received from the client.
<i>ping-clients</i>	int (v4)
	If set to a 1, performs a ping before offering a lease for this request. Read just before determining if a lease is acceptable for a client.
<i>relay-agent-circuit-id</i>	blob (v4, v6)
	Contents of the circuit-id suboption of option 82.
<i>relay-agent-circuit-id-data</i>	blob (v4, v6)
	Contents of just the data part of the circuit-id suboption of option 82.
<i>relay-agent-device-class-data</i>	blob (v4, v6)
	Contents of the device-class suboption of option 82.
<i>relay-agent-radius-attributes</i>	blob (v4)
	Contents of the radius suboption of option 82.
<i>relay-agent-radius-class</i>	string (v4)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
	Encapsulated class attribute of the radius suboption of option 82.
<i>relay-agent-radius-pool-name</i>	string (v4)
	Encapsulated framed-pool attribute of the radius suboption of option 82.
<i>relay-agent-radius-user</i>	string (v4)
	Encapsulated user attribute of the radius suboption of option 82.
<i>relay-agent-remote-id</i>	blob (v4, v6)
	Contents of the remote-id suboption of option 82.
<i>relay-agent-remote-id</i>	blob (v4, v6)
	Contents of just the data part of the remote-id suboption of option 82.
<i>relay-agent-server-id-override-data</i>	IPv6 address (v4, v6)
	Contents of the server-id suboption of option 82. If the IANA suboption 182 is in the packet, that value appears; otherwise, the Cisco suboption 152 value appears.
<i>relay-agent-subscriber-id</i>	string (v4)
	Contents of the subscriber-id suboption of option 82
<i>relay-count</i>	int (v6, read-only)
	Number of DHCPv6 relay hops
<i>reply-options</i>	blob
	Overrides any DHCPv4 reply options specified in any policy. Read when gathering data for the output packet.
<i>reply-to-client-address</i>	int (v4, v6)
	For v4, if set to 1, the server sends the response packet to the client-ipaddress and the client-port. For v6, if set to 1, the server sends the response packet back to the address and port of the sender (client or relay agent). If 0, the server sends the response using the RFC mandated algorithm.
<i>reserved-addresses</i>	IP address (v4, read/write)
	List of addresses reserved for the client. The first available address to match a usable Scope (which must have restrict-to-reservations enabled) will be assigned to the client.
<i>reserved-ip6addresses</i>	IP address (v6, read/write)
	List of addresses reserved for the client. All available addresses to match a usable Prefix (which must have restrict-to-reservations enabled) will be assigned to the client.
<i>reserved-prefixes</i>	IP address (v6, read/write)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
	List of prefixes reserved for the client. All available prefixes to match a usable Prefix (which must have restrict-to-reservations enabled) will be assigned to the client.
<i>selection-criteria</i>	string (v4, v6)
	Comma-separated string that contains the scope selection criteria.
<i>selection-criteria-excluded</i>	string (v4, v6)
	Comma-separated string that contains the scope exclusion criteria.
<i>send-ack-first</i>	int (v4, v6)
	If set to a 1, updates DNS after the ACK for DHCP requests. Read just before initiating the DNS operation.
<i>source-ipaddress</i>	IPv6 address (v6, read-only)
	IPv6 source address of the packet.
<i>trace-id</i>	string (v4, v6, read-only)
	ID used by the system to trace the packet.
<i>transaction-time</i>	int (v4, v6)
	Time, in seconds since 1970, that the input packet was decoded.
<i>update-dns</i>	string (v4, v6)
	Requests partial, full, or no dynamic DNS updates on a per-request packet basis. Input and output values are: 1=update-all, 2=update-fwd-only, 3=update-rev-only, and 0=update-none.
<i>update-dns-for-bootp</i>	int (v4)
	If set to a 1, updates DNS for BOOTP requests for this request. Read just before initializing the DNS operation for BOOTP.
<i>verbose-logging</i>	int (v4, v6)
	If set to a 1, logs verbose messages for this request. Read at various times during processing.
<i>vpn-description</i>	string (v4, v6, read-only)
	Description for the VPN. See <i>vpn-name</i> for details.
<i>vpn-name</i>	string (v4, v6, read-only)
	Name of the VPN. The request dictionary does not have valid values for these items at post-packet-decode , but does at all other extension points, because the VPN has not yet been determined. This is so that a script can change the <i>derived-vpn-id</i> option or suboption at post-packet-decode and thereby affect the VPN used for a lease
<i>vpn-vpn-id</i>	blob, typically 7 bytes (v4, v6, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
Virtual private network identifier. See <i>vpn-name</i> for details.	
<i>vpn-vrf-name</i>	string (v4, v6, read-only)
Virtual routing and forwarding table identifier for the VPN. See <i>vpn-name</i> for details.	

Response Dictionary

The table below lists the data items you can set in the response dictionary at any time. The DHCP server reads them at various times. Unless indicated otherwise, the operation is read/write.

Table 73: Response Dictionary Specific Data Items

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>active-leasequery-control</i>	int (v4,v6)
Controls the sending of a lease (such as only on specific state changes). Values are: 0—unspecified (the server determines whether to send the notification), 1—send (the server will send the notification), and 2—do not send (the server will not send the notification). The <i>active-leasequery-control</i> is initialized as 0, that is, unspecified.	
<i>client-active-lease-count</i>	int (v6, read-only)
Number of active leases on the DHCPv6 client.	
<i>client-creation-time</i>	int (v6, read-only)
Creation time of the IPv6 client.	
<i>client-domain-name</i>	string (v4, read-only)
From the client information in the lease, the domain name that the client wants to use. It might not exist, in which case the DHCP server uses the domain name specified in the scope. Read when queuing the request for DNS update just prior to the update of stable storage.	
<i>client-expiration-time</i>	int (v4, v6, read-only)
The highest lease expiration time given to the client by this server (in seconds, since 1970).	
<i>client-host-name</i>	string (v4, read-only)
From the client information in the lease, the hostname that the DHCP server puts into DNS. Read when queueing the request for a DNS update just before updating stable storage.	
<i>client-id</i>	blob (v4, v6, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
From the client information in the lease, the client identification that the server used to keep track of the client. This might be the <i>client-id</i> sent with a request or internally generated from the MAC address. For DHCPv6, the Client Identifier Option value (the client's DUID).	
<i>client-id-created-from-mac-address</i>	int (v4, read-only)
From the client information in the lease. If set to 1, the <i>client-id</i> must be created from the MAC address and should not be used in reporting.	
<i>client-last-transaction-time</i>	int (v4, v6, read-only)
Time, in seconds, since 1970, that the DHCP server last heard from this client.	
<i>client-limitation-id</i>	blob (v4, read-only)
Limitation identifier of the client associated with the current lease.	
<i>client-mac-address</i>	blob (v4, read-only)
From the client information in the lease, the MAC address stored in the client object associated with the request dictionary. Has the same format as (and was created from) the <i>mac-address</i> .	
<i>client-os-type</i>	int (v4)
Change the client entry of the request packet by setting this at the pre-client-lookup or post-client-lookup extension points. Can also be read at check-lease-acceptable , but cannot be set there. To set the value, you must first set the <i>os-type</i> in the post-packet-decode request dictionary.	
<i>client-override-client-id</i>	blob (v4, v6, read-only)
Blob used for the current <i>client-id</i> value. Replaces any <i>client-id</i> from the incoming packet (although both values are kept in the lease state database).	
<i>client-override-client-id-data-type</i>	string (v4, v6, read-only)
Returns the data type of the <i>client-override-client-id</i> , either nstr for string or <i>blob</i> for blob.	
<i>client-override-client-id-string</i>	string (v4, v6, read-only)
Current <i>client-id</i> value in string format that replaces any <i>client-id</i> from the incoming packet (although both values are kept in the lease state database). For a <i>get</i> , if the <i>client-override-client-id</i> is not a string, the binary data is formatted as blob data, which is then returned as the “string.”	
<i>client-packet</i>	blob (v4, v6, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
The client portion of the response packet. For DHCPv4, this is the complete packet. For DHCPv6, this is the client message. (See packet to obtain the full packet.) Only available from the post-packet-encode extension point.	
<i>client-reconfigure-key</i>	string (v6)
Returns the <i>client-reconfigure-key</i> attribute value of the DHCPv6 lease.	
<i>client-reconfigure-key-generation-time</i>	string (v6)
Returns the <i>client-reconfigure-key-generation-time</i> attribute value of the DHCPv6 lease.	
<i>client-relay-address</i>	IPv6 address (v6, read-only)
Source IPv6 address for the (last) relay.	
<i>client-relay-message</i>	string (v6, read-only)
Last relayed DHCPv6 message, excluding the client message.	
<i>client-requested-host-name</i>	string (v4)
From the client information in the lease, the hostname that the client requested for the DNS update.	
<i>client-user-defined-data</i>	string (v4, v6, read-only)
Returns the value previously or currently associated with the client, as derived from the <i>user-defined-data</i> environment dictionary data item. It returns the previously associated value if requested in a check-lease-acceptable or lease-state-change extension point. It returns the current value if requested in a pre-packet-encode or post-send-packet extension point.	
<i>client-vendor-class</i>	string (v4, v6)
Returns the <i>client-vendor-class</i> attribute value of the DHCPv4 or DHCPv6 lease.	
<i>client-vendor-info</i>	string (v4, v6)
Returns the <i>client-vendor-info</i> attribute value of the DHCPv4 or DHCPv6 lease.	
<i>client-write-sequence</i>	int (v6, read-only)
Write sequence of the client IPv6 request.	
<i>client-write-time</i>	int (v6, read-only)
Time of the client IPv6 write request.	

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>derived-vpn-id</i>	int (v4, v6, read-only)
VPN identifier.	
<i>domain-name-changed</i>	int (v4)
If set to 1, the domain name in the current packet differs from the domain name used in the DNS update. Read after check-lease-acceptable and before pre-packet-encode .	
<i>dump-packet</i>	int (v4, v6, write-only)
When set to 1, Cisco Prime Network Registrar dumps the current decoded DHCP/BOOTP packet to the log file. An extension can put the value 1 into this data item at multiple points in its execution. This might be useful when debugging extensions.	
<i>failover-role</i>	int (v4, v6, read only)
Determines the failover server role. The failover server role can be one of three values: <ul style="list-style-type: none"> • None—Failover is not configured. • Main/Backup—Failover is configured and the role of the failover server 	
<i>failover-state</i>	int (v4, v6, read only)
Determines failover server state. The failover state can be normal, partner-down, communications-interrupted, recover, potential-conflict, recover-done, startup, shutdown, or paused. If failover is not configured the value is none.	
<i>host-name-changed</i>	int (v4)
If set to 1, the hostname in the current packet differs from that used in the DNS update. Read after check-lease-acceptable and before pre-packet-encode .	
<i>host-name-in-dns</i>	int (v4, v6)
If set to 1, the hostname is in DNS. Read after check-lease-acceptable and before pre-packet-encode . Written after the hostname goes into DNS.	
<i>lease-binding-iaid</i>	int (v6, read-only)
IPv6 lease binding IAID.	
<i>lease-binding-rebinding-time</i>	int (v6, read-only)
IPv6 lease binding rebinding time.	

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>lease-binding-renewal-time</i>	int (v6, read-only)
IPv6 lease binding renewal time.	
<i>lease-binding-type</i>	string (v6, read-only)
IPv6 lease binding type: "IA_NA", "IA_TA", or "IA_PD".	
<i>lease-client-reserved</i>	int (v4, v6, read-only)
Returns 1 if the lease is client reserved and 0 if not.	
<i>lease-creation-time</i>	string (v6, read-only)
IPv6 lease creation time.	
<i>lease-deactivated</i>	int (v4, v6, read-only)
If set to 1, reports that the lease is deactivated.	
<i>lease-dns-forward-backup-server-address</i>	IP address (v4, v6, read-only)
Address of the backup DNS server that receives DNS updates for the DHCPv4 and DHCPv6 lease, if the server specified in <i>lease-dns-forward-server-address</i> is down.	
<i>lease-dns-forward-server-address</i>	IP address (v4, v6, read-only)
Address of the DNS server that receives dynamic DNS updates for the DHCPv4 and DHCPv6 lease.	
<i>lease-dns-forward-update</i>	string (v4, v6, read-only)
Name of the update configuration that determines the forward zones to be included in DNS updates for the DHCPv4 and DHCPv6 lease. Returns TRUE if <i>update-all</i> or <i>update-fwd-only</i> is set.	
<i>lease-dns-forward-zone-name</i>	string (v4, v6, read-only)
Name of an optional forward zone for DNS updates.	
<i>lease-dns-reverse-backup-server-address</i>	IP address (v4, v6, read-only)
Address of the backup DNS server that receives DNS updates for a DHCPv4 and DHCPv6 lease, if the server specified in <i>lease-dns-reverse-server-address</i> is down.	
<i>lease-dns-reverse-host-bytes</i>	int (v4, read-only)
The number of bytes in a lease IP address to use for a reverse zone.	
<i>lease-dns-reverse-prefix-length</i>	int (v6, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
Prefix length of the reverse zone for ip6.arpa updates.	
<i>lease-dns-reverse-server-address</i>	IP address (v4, v6, read-only)
Address of the DNS server address that receives dynamic DNS updates for the DHCPv4 and DHCPv6 lease.	
<i>lease-dns-reverse-update</i>	string (v4, v6, read-only)
Name of the update configuration that determines which reverse zones to include in a DNS update for the DHCPv4 and DHCPv6 lease. Returns TRUE if <i>update-all</i> or <i>update-fwd-only</i> is set.	
<i>lease-dns-reverse-zone-name</i>	string (v4, v6, read-only)
DNS reverse (in-addr.arpa and ip6.arpa) zone that is updated with PTR records.	
<i>lease-fqdn</i>	string (v6, read-only)
Fully qualified domain name assigned to the DHCPv6 lease by the server (and possibly successfully entered into DNS). The <i>lease-fqdn</i> may be the name that is expected to be added to DNS for the lease or the actual name added. If <i>host-name-in-dns</i> is equal to true, the actual <i>lease-fqdn</i> is in DNS.	
<i>lease-requested-fqdn</i>	string (v6, read-only)
Partial or fully qualified domain name most recently requested by the client for the DHCPv6 lease.	
<i>lease-giaddr</i>	IP address (v4, read-only)
Lease <i>giaddr</i> .	
<i>lease-ipaddress</i>	IPv4 or IPv6 address or prefix (v4, v6, read-only)
For DHCPv4, the address of the lease associated with the client. For DHCPv6, the IPv6 address or IPv6 prefix (address and prefix-length) of the lease for the current context (See setObject method).	
<i>lease-preferred-lifetime</i>	int (v6, read-only)
Preferred lifetime of the IPv6 lease.	
<i>lease-prefix-name</i>	string (v6, read-only)
Prefix name of the IPv6 lease.	
<i>lease-relay-agent-info</i>	blob (v4)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
Entire contents of option 82.	
<i>lease-relay-agent-circuit-id</i>	blob (v4)
Accesses and manipulates the relay agent circuit ID as stored with the lease of a response. Requires the suboption number 1 as the first byte. Deprecated in favor of the <i>lease-relay-agent-circuit-id-data</i> item.	
<i>lease-relay-agent-circuit-id-data</i>	blob (v4, use instead of deprecated <i>lease-relay-agent-circuit-id</i>)
Accesses and manipulates the <i>relay-agent-circuit-id-data</i> as stored with the lease of a response.	
<i>lease-relay-agent-device-class-data</i>	blob (v4)
Contents of the device-class suboption of option 82.	
<i>lease-relay-agent-radius-attributes</i>	blob (v4)
Contents of the radius suboption of option 82.	
<i>lease-relay-agent-radius-class</i>	string (v4)
Encapsulated class attribute of the radius suboption of option 82.	
<i>lease-relay-agent-radius-pool-name</i>	string (v4)
Encapsulated framed-pool attribute of the radius suboption of option 82.	
<i>lease-relay-agent-radius-user</i>	string (v4)
Encapsulated user attribute of the radius suboption of option 82.	
<i>lease-relay-agent-remote-id</i>	blob (v4)
Accesses and manipulates the <i>relay-agent-remote-id</i> data as stored with the lease of a response. Requires suboption number 2 as the first byte. Deprecated in favor of the <i>lease-relay-agent-remote-id-data</i> item.	
<i>lease-relay-agent-remote-id-data</i>	blob (v4, use instead of <i>lease-relay-agent-remote-id</i> item)
Accesses and manipulates the <i>relay-agent-remote-id-data</i> as stored with the lease of a response.	
<i>lease-relay-agent-server-id-override-data</i>	IP address (v4)
Accesses and manipulates the <i>relay-agent-server-id-override-data</i> as stored with the lease of a response.	

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>lease-relay-agent-subnet-selection-data</i>	IP address (v4)
Accesses and manipulates the <i>relay-agent-subnet-selection-data</i> as stored with the lease of a response.	
<i>lease-relay-agent-subscriber-id</i>	string (v4)
Contents of the subscriber-id suboption of option 82.	
<i>lease-relay-agent-vpn-id-data</i>	blob (v4)
Accesses and manipulates the <i>relay-agent-vpn-id</i> data as stored with the lease of a response.	
<i>lease-reserved</i>	int (v4, v6, read-only)
Returns 1 if the lease is lease reserved and 0 if not.	
<i>lease-start-time-of-state</i>	int (v4, v6, read-only)
Time, in seconds since 1970, that this lease was first placed into its current state.	
<i>lease-state</i>	string (v4, v6, read-only)
State of the lease, which can be available, offered, leased, expired, unavailable, released, other-available (DHCPv4 only), pending-available (DHCPv4 only), or revoked (DHCPv6 only).	
<i>lease-state-expiration-time</i>	int (v6, read-only)
Expiration time of the IPv6 lease state.	
<i>lease-status</i>	string (v4, v6, read-only)
Returns “nonexistent,” “owned-by-client,” or “exists.” Used to determine if a lease exists and if the current client owns it. If “exists” is returned, the lease exists but the current owner does not own it (limited information on the lease is available).	
<i>lease-valid-lifetime</i>	int (v6, read-only)
Valid lifetime of the IPv6 lease.	
<i>lease-vpn-description</i>	string (v4, v6, read-only)
Description for the VPN stored with the lease of a response.	
<i>lease-vpn-id</i>	int (v4, v6, read-only)
Identifier for the VPN stored with the lease of a response.	
<i>lease-vpn-name</i>	string (v4, v6, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
Name of the VPN stored with the lease of a response.	
<i>lease-vpn-vpn-id</i>	blob, typically 7 bytes (v4, v6, read-only)
Virtual private network (VPN) identifier stored with the lease of a response.	
<i>lease-vpn-vrf-name</i>	string (v4, v6, read-only)
Virtual routing and forwarding table identifier for the VPN stored with the lease of a response.	
<i>mac-address</i>	blob (v4)
MAC address that came in the client packet. The first byte is the hardware type, the second is the hardware length, and the remaining (up to 16) is the information from the <i>chaddr</i> . This is a useful aggregation of the <i>htype</i> , <i>hlen</i> , and <i>chaddr</i> fields of the DHCP packet. When read it is constructed from these fields; when written it is placed into these fields.	
<i>override-client-id</i>	blob (v4, v6, read-only)
Blob used for the current client-id value. Replaces any client-id from the incoming packet (although both values are kept in the lease state database).	
<i>override-client-id-data-type</i>	string (v4, v6, read-only)
Returns the data type of the <i>override-client-id</i> , either “nstr” for string or “blob” for blob.	
<i>override-client-id-string</i>	string (v4, v6, read-only)
Current client-ID value in string format that replaces any client-id from the incoming packet (although both values are kept in the lease state database). For a <i>get</i> , if the <i>override-client-id</i> is not a string, the binary data is formatted as blob data, which is then returned as the “string.”	
<i>packet</i>	blob (v4, v6, use only at post-packet-decode)
The response packet. For DHCPv4, this is the same as client-packet. For DHCPv6, this is the full packet if relayed or the same as client-packet if not relayed. It should only be read or written from the post-packet-encode extension point; if written, the server will then send the new packet to the client.	
<i>ping-clients</i>	int (v4)
If set to 1, performs a ping before offering a lease for this request. Read just before determining a client lease acceptability.	

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>prefix-address</i>	IPv6 prefix (v6, read-only)
Prefix address (17 bytes—IPv6 address and prefix length).	
<i>prefix-allocate-random</i>	int (v6, read-only)
Prefix randomly allocated.	
<i>prefix-allocate-via-best-fit</i>	int (v6, read-only)
Prefix allocated via the best fit.	
<i>prefix-allocate-via-client-request</i>	int (v6, read-only)
Prefix allocated via client request.	
<i>prefix-allocate-via-extension</i>	int (v6, read-only)
Prefix allocated via an extension.	
<i>prefix-allocate-via-reservation</i>	int (v6, read-only)
Prefix allocated via a reservation.	
<i>prefix-allocate-via-interface-identifier</i>	int (v6, read-only)
Prefix allocated via an interface identifier.	
<i>prefix-allocation-group</i>	string (v6, read-only)
Allocation group name for the prefix.	
<i>prefix-allocation-group-priority</i>	int (v6, read-only)
Allocation group priority for the prefix.	
<i>prefix-deactivated</i>	int (v6, read-only)
Indicates if the prefix is deactivated.	
<i>prefix-dhcp-type</i>	string (v6, read-only)
Prefix DHCP type.	
<i>prefix-expiration-time</i>	string (v6, read-only)
Expiration time of the prefix.	
<i>prefix-link-group-name</i>	string (v6, read-only)
Link group name for the link.	
<i>prefix-link-name</i>	string (v6, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
Link of the prefix.	
<i>prefix-link-type</i>	string (v6, read-only)
Link type (topological, location-independent, or universal).	
<i>prefix-name</i>	string (v6, read-only)
Name of the prefix.	
<i>prefix-range</i>	IPv6 address (v6, read-only)
IPv6 address range of the prefix.	
<i>prefix-restrict-to-reservations</i>	int (v6, read-only)
If set to 1, the prefix has restrict-to-reservations enabled.	
<i>prefix-selection-tags</i>	string (v6, read-only)
Selection tags of the prefix.	
<i>relay-count</i>	int (v6, read-only)
Number of DHCPv6 relay hops.	
<i>reply-ipaddress</i>	IPv4 or IPv6 address (v4, v6)
IP address to use when replying to the DHCP client. Read just after pre-packet-encode . If you change its value in a pre-packet-encode , the IP address you place in it should be for a system that can respond to ARP queries (unless it is a broadcast address). Even if unicast is enabled and the broadcast flag is not set in the DHCP request, the local ARP cache is not set with a mapping from a new <i>reply-ipaddress</i> in the pre-packet-encode to the MAC address in the DHCP request.	
<i>reply-port</i>	int (v4, v6)
Port to use when replying to the DHCP client. Read just after pre-packet-encode .	
<i>response-source</i>	string (v4, v6, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<p>The source of the response (the major activity that invoked the extension). Output values are: client (Received client packet), failover (Received binding update from the failover partner), timeout (Lease expiration or grace period end), operator (Request from a user interface), one-lease-per-client (One lease per client removing a client from an old lease because of a new one), unknown (None of the above).</p> <p>This data item helps an extension to determine what processing it should do whether a request dictionary is present or not. (The isValid method can also be used to determine whether a dictionary is valid.)</p>	
<i>reverse-name-in-dns</i>	int (v4, v6)
If equal to 1, the reverse name is in DNS. Read before initializing a DNS operation.	
<i>scope-allow-bootp</i>	int (v4, read-only)
If set to 1, the scope allows BOOTP. Written after a DNS operation finishes.	
<i>scope-allow-dhcp</i>	int (v4, read-only)
If set to 1, the scope allows DHCP.	
<i>scope-allow-dynamic-bootp</i>	int (v4, read-only)
If set to 1, the scope allows dynamic BOOTP.	
<i>scope-available-leases</i>	int (v4, read-only)
Number of available leases on the current scope.	
<i>scope-deactivated</i>	int (v4, read-only)
If set to 1, the scope is deactivated.	
<i>scope-dns-forward-server-address</i>	IP address (v4, read-only)
DNS server to use for the DNS forward address.	
<i>scope-dns-forward-zone-name</i>	string (v4, read-only)
Forward zone name configured in the scope.	
<i>scope-dns-number-of-host-bytes</i>	int (v4, read-only)
Number of host bytes used by the DHCP server code that handles DNS updates.	
<i>scope-dns-reverse-server-address</i>	IP address (v4, read-only)
DNS server to use for the DNS reverse address.	

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
<i>scope-dns-reverse-zone-name</i>	string (v4, read-only)
Reverse zone name configured in the scope.	
<i>scope-network-number</i>	IP address (v4, read-only)
Network number of the scope that contains the lease the DHCP server is processing.	
<i>scope-ping-clients</i>	int (v4, read-only)
If set to 1, the scope associated with the current lease was configured to support a ping operation prior to offering a lease.	
<i>scope-primary-network-number</i>	IP address (v4, read-only)
Network number of this primary scope.	
<i>scope-primary-subnet-mask</i>	IP address (v4, read-only)
Subnet mask of this primary scope.	
<i>scope-renew-only</i>	int (v4, read-only)
If set to 1, the scope is renew-only.	
<i>scope-renew-only-expire-time</i>	int (v4, read-only)
Absolute time, in seconds since January 1, 1970, at which a renew-only scope should cease to be renew-only.	
<i>scope-restrict-to-reservations</i>	int (v4, read-only)
If set to 1, the scope has restrict-to-reservations enabled.	
<i>scope-selection-tags</i>	string (v4, read-only)
Comma-separated string that contains the scope selection criteria. Use this data item for decisions based on scopes.	
<i>scope-send-ack-first</i>	int (v4, read-only)
If set to 1, the scope sends an ACK before performing the rest of the processing.	
<i>scope-subnet-mask</i>	IP address (v4, read-only)
Subnet mask of the scope that contains the lease the DHCP server is processing.	
<i>scope-update-dns</i>	string (v4, read-only)

Data Item	Value (Protocol: v4=DHCPv4, v6=DHCPv6)
DNS updates for forward or reverse zones. Output values are: 1=update-all, 2=update-fwd-only, 3=update-rev-only, and 0=update-none.	
<i>scope-update-dns-enabled</i>	boolean (v4, read-only)
If set to 1, the scope has update DNS enabled for forward and reverse zones. Deprecated in favor of <i>scope-update-dns</i> .	
<i>scope-update-dns-for-bootp</i>	int (v4, read-only)
If set to 1, the scope has update DNS enabled for BOOTP.	
<i>trace-id</i>	string (v4, v6, read-only)
ID used by the system to trace the packet.	
<i>transaction-time</i>	int (v4, v6, read-only)
Time, in seconds since 1970, that the request was decoded.	
<i>vpn-description</i>	string (v4, v6, read-only)
Description for the VPN.	
<i>vpn-name</i>	string (v4, v6, read-only)
Name of the VPN.	
<i>vpn-vpn-id</i>	blob, typically 7 bytes (v4, v6, read-only)
Virtual private network (VPN) identifier.	
<i>vpn-vrf-name</i>	string (v4, v6, read-only)
Virtual routing and forwarding table (VRF) identifier for the VPN.	

Extension Dictionary API

This section contains the dictionary method calls to use when accessing dictionaries from Tcl extensions and shared libraries.

Tcl Attribute Dictionary API

In an attribute dictionary, the keys are constrained to be the names of attributes as defined in the Cisco Network Registrar DHCP server configuration. The values are the string representation of the legal values for that particular attribute. For example, IP addresses are specified by the dotted-decimal string representation of the address, and enumerated values are specified by the name of the enumeration. This means that numbers are specified by the string representation of the number.

Attribute dictionaries are unusual in that they can contain more than one instance of a key. These instances are ordered, with the first instance at index zero. Some of the attribute dictionary methods allow an index to indicate a particular instance or position in the list of instances to be referenced.

Tcl Request and Response Dictionary Methods

Attribute dictionaries use commands with which you can change and access the values in the dictionaries. The table below lists the commands to use with the request and response dictionaries. In this case, you can define the *dict* variable as **request** or **response**.

See the *install-path* /examples/dhcp/tcl/tclexension.tcl file for examples.

Table 74: Tcl Request and Response Dictionary Methods

Method	Syntax
get	\$dict get attribute [index [bMore]]
Returns the value of the attribute from the dictionary, represented as a string. If the dictionary does not contain the attribute, the empty string is returned instead. If you include the index value, this returns the <i>index</i> th instance of the attribute. Some attributes can appear more than once in the request or response packet. The index selects which instance to return. If you include the <i>bMore</i> , the get method sets <i>bMore</i> to TRUE if there are more attributes after the one returned, otherwise to FALSE. Use this to determine whether to make another call to get to retrieve other instances of the attribute.	
getOption	\$dict getOption arg-type [arg-data]
Gets the data for an option as a string. See Table 75: Tcl arg-type and obj-type Values , on page 485 for the <i>arg-type</i> values. If the next argument is a numeric value, it is assumed to be a number, otherwise a name. Note that this function always returns a pointer to a string, which can be zero length if the option does not exist or has length zero. For sample usage, see the Handling Vendor Class Option Data, on page 501 .	
isValid isV4 isV6	\$dict isValid \$dict isV4 \$dict isV6
The isValid method returns TRUE if there is a request or response (depending on the dictionary passed in); FALSE otherwise. Extensions such as lease-state-change can use this method to determine whether a dictionary is available. The isV4 method returns TRUE if this extension is being called for a DHCPv4 packet; FALSE otherwise. Calling this method from an init-entry routine returns FALSE. The isV6 method returns TRUE if this extension is being called for a DHCPv6 packet; FALSE otherwise. Calling this method from an init-entry routine returns FALSE.	

Method	Syntax
log	<i>\$dict log level message</i> ...
<p>Puts a message into the DHCP server logging system. The level should be LOG_ERROR, LOG_WARNING, or LOG_INFO. The remaining arguments are concatenated and sent to the logging system at the specified level.</p> <p>Note Use the LOG_ERROR and LOG_WARNING levels sparingly, because the server flushes its log file with messages logged at these levels. Using these levels for messages that are likely to occur frequently (such as client requests) can have severe impact on disk I/O performance.</p>	
moveToOption	<i>\$dict moveToOption arg-type [arg-data] ...</i>
<p>Sets the context for subsequent get, put, and remove option operations. See Table 75: Tcl arg-type and obj-type Values , on page 485 for the <i>arg-type</i> values. Note that the context can become invalid if the option is removed (such as by removeOption).</p>	
put	<i>\$dict put attribute value [index]</i>
<p>Associates a value with the attribute in the dictionary. If you omit the index or set it to the special value REPLACE, this replaces any existing instances of the attribute with the single value. If you include the index value as the special value APPEND, this appends a new instance of the attribute to the end of the list of instances of the attribute. If you include the index value as a number, this inserts a new instance of the attribute at the position indicated. If you set the index value to the special value AUGMENT, this puts the attribute only if there is not one already.</p>	
putOption	<i>\$dict putOption data arg-type [arg-data] ...</i>
<p>Adds an option and its data or modifies the data for an option. See Table 75: Tcl arg-type and obj-type Values , on page 485 for the <i>arg-type</i> values. For sample usage, see the Handling Vendor Class Option Data, on page 501.</p>	
remove	<i>\$dict remove attribute [index]</i>
<p>Removes the attribute from the dictionary. If you omit the index or set it to the special value REMOVE_ALL, this removes any existing instances of the attribute. If you include the index as a number, this removes the instance of the attribute at the position indicated. This method always returns 1, even if the dictionary does not contain that attribute at that index.</p>	
removeOption	<i>\$dict removeOption arg-type [arg-data] ...</i>
<p>Removes an option. See Table 75: Tcl arg-type and obj-type Values , on page 485 for the <i>arg-type</i> values. For sample usage, see the Handling Vendor Class Option Data, on page 501.</p>	

Method	Syntax
setObject	\$dict setObject <i>obj-type [data]</i>
(DHCPv6 only.) Sets the object for get , put , and remove methods, and alters the message on which the new option methods operate. See Table 75: Tcl arg-type and obj-type Values , on page 485 for the <i>obj-type</i> values. DHCPv6 extensions primarily use this method to access the leases and prefixes available for the client and link, or to get message header fields or options from relay packets. Unlike in DHCPv4, where one lease and scope are associated with a response, a DHCPv6 response can involve several leases and prefixes. Returns TRUE if the object exists; FALSE otherwise. For sample usage, see the Handling Object Data , on page 501. Note For leases not associated with the current client, only minimal information is available.	
trace	\$dict trace level <i>message ...</i>
Returns a message in the DHCP server packet tracing system. At level 0, no tracing occurs. At level 1, it traces only that the server received the packet and sent a reply. At level 4, it traces everything. The remaining arguments are concatenated and sent to the tracing system at the specified level. The default tracing is set using the DHCP server <i>extension-trace-level</i> attribute.	

Table 75: Tcl arg-type and obj-type Values

arg-type or obj-type	Description
enterprise <i>number/name</i>	Enterprise-id number or name for the option definition set for the option or suboption.
home	Requests that the context is reset to the “top” of the current client or relay message.
index <i>number/keyword</i>	Number or keyword (replace, append, augment, raw, or remove_all) for the array index on which to operate.
index-count	Returns the number of array index entries in the option.
instance <i>number</i>	Instance number of the option (primarily used for DHCPv6).
instance-count	Returns the number of times the option appears (if 0, the option is not present).
more <i>tcl-variable-name</i>	Name of a Tcl variable that is set to TRUE or FALSE, depending on whether more array index entries exist in the option data.
move-to	Requests that the context be set to the option.
option <i>number/name</i>	Option number or name to operate on.
parent	Requests that the context is moved up one option.

arg-type or obj-type	Description
suboption <i>number/name</i>	Suboption number or name to operate on.
vendor <i>name</i>	Vendor name for the option definition set for the option or suboption.
lease initial <i>index</i> <i>address</i> <i>prefix</i>	Used with setObject , sets the context for the lease, binding, and prefix data items in the response dictionary to the indicated lease. The initial keyword requests that the original context for when the extension was called is restored. The <i>index</i> requests that the numbered lease (starting at 0) is set and can be used to iterate through all of the leases for the client. The <i>address</i> or <i>prefix</i> requests that the lease with that address or prefix is set (if it exists).
message initial <i>number</i>	Used with setObject , sets the context for the message data items and options in the request or response dictionary to the indicated message. The initial keyword sets the context to the client message. The <i>number</i> sets the context to the relay message, with 0 specifying the relay closest to the client.
prefix initial <i>index</i> <i>address</i> <i>prefix</i> <i>name</i>	Used with setObject , sets the context for the prefix data items in the response dictionary to the indicated prefix. The initial keyword requests that the original context for when the extension was called is restored. The <i>index</i> requests the numbered prefix (starting at 0) is set and can be used to iterate through all of the prefixes for the client on the link. The <i>address</i> or <i>prefix</i> requests that the prefix for the address or prefix is set (if found). The <i>name</i> requests that the named prefix is found. Note that only prefixes on the current link can be used.

Tcl Environment Dictionary Methods

The table below describes the commands to use with the environment dictionary. In this case, you can define the *dict* variable as **environ**, as in the following procedure example:

```
proc tclhelloworld2 { request response environ } {
  $environ put trace-level 4
  $environ log LOG_INFO "Environment hello world"
}
```

Table 76: Tcl Environment Dictionary Methods

Method	Syntax
clear	<i>\$dict clear</i>
Removes all entries from the dictionary.	
containsKey	<i>\$dict containsKey key</i>
Returns 1 if the dictionary contains the key, otherwise 0.	
firstKey	<i>\$dict firstKey</i>
Returns the name of the first key in the dictionary. Note that the keys are not stored sorted by name. If a key does not exist, returns the empty string.	

Method	Syntax
get	<i>\$dict get key</i>
Returns the value of the key from the dictionary. If a key does not exist, returns the empty string.	
isEmpty	<i>\$dict isEmpty</i>
Returns 1 if the dictionary has no entries, otherwise 0.	
log	<i>\$dict log level message ...</i>
Returns a message in the DHCP server logging system. The <i>level</i> should be one of LOG_ERROR, LOG_WARNING, or LOG_INFO. The remaining arguments are concatenated and sent to the logging system at the specified level.	
Note	Use the LOG_ERROR and LOG_WARNING levels sparingly, because the server flushes its log file with messages logged at these levels. Using these levels for messages that are likely to occur frequently (such as client requests) can have severe impact on disk I/O performance.
nextKey	<i>\$dict nextKey</i>
Returns the name of the next key in the dictionary that follows the key returned in the last call to firstKey or nextKey . If a key does not exist, returns the empty string.	
put	<i>\$dict put key value</i>
Associates a value with the key, replacing an existing instance of the key with the new value.	
remove	<i>\$dict remove key</i>
Removes the key from the dictionary. Always returns 1, even if the dictionary did not contain the key.	
size	<i>\$dict size</i>
Returns the number of entries in the dictionary.	
trace	<i>\$dict trace level message ...</i>
Returns a message in the DHCP server packet tracing system. At level 0, no tracing occurs. At level 1, it traces only that the server received the packet and sent a reply. At level 4, it traces everything. The remaining arguments are concatenated and sent to the tracing system at the specified level. The default tracing is set using the DHCP server <i>extension-trace-level</i> attribute.	

DEX Attribute Dictionary API

When writing DEX extensions for C/C++, you can specify keys as the attribute name string representation or by type (a byte sequence defining the attribute). This means that some of these access methods have four different variations that are the combinations of string or type for the key or value.

A basic DEX extension example might be:

```
int DEXAPI dexhelloworld( int iExtensionPoint,
dex_AttributeDictionary_t *pRequest,
```

```

dex_AttributeDictionary_t *pResponse,
dex_EnvironmentDictionary_t *pEnviron )
{
pEnviron->log( pEnviron, DEX_LOG_INFO, "hello world" );
return DEX_OK;
}

```

See the *install-path* /examples/dhcp/dex/dexextension.c file or other files in that directory for examples.

DEX Request and Response Dictionary Methods

DEX attribute dictionaries use active commands, called methods, with which you can change and access values. The table below lists the methods to use with the request and response dictionaries. In this case, you can define the *pDict* variable as **pRequest** or **pResponse**, as in:

```
pRequest->get( pRequest, "host-name", 0, 0 );
```

The *pszAttribute* is the **const char *** pointer to the attribute name that the application wants to access. The *pszValue* is the pointer to the **const char *** string that represents the data (returned for a **get** method, and stored in a **put** method). See the corresponding *iObjectType*, *iObjArgType*, and *iArgType* tables, respectively.



Tip See also the [Differences in get, put, Option, Bytes, and OptionBytes Methods, on page 494](#) and the [Differences in get, put, remove, and ByType Methods, on page 494](#).

Table 77: DEX Request and Response Dictionary Methods

Method	Syntax
allocateMemory	void *pDict->allocateMemory(dex_AttributeDictionary_t *pDict, unsigned int iSize)
Allocates memory in extensions that persists only for the lifetime of this request.	
get	const char *pDict->get(dex_AttributeDictionary_t *pDict, const char *pszAttribute, int iIndex, abool_t *pbMore)
Returns the value of the <i>iIndex</i> ed instance of the attribute from the dictionary, represented as a string. If the dictionary does not contain the attribute (or that many instances of it), the empty string is returned instead. If <i>pbMore</i> is nonzero, the get method sets <i>pbMore</i> to TRUE if there are more instances of the attribute after the one returned, otherwise to FALSE. Use this to determine whether to make another call to get to retrieve other instances of the attribute.	
getBytes	const abytes_t *pDict->getBytes(dex_AttributeDictionary_t *pDict, const char *pszAttribute, int iIndex, abool_t *pbMore)
Returns the value of the <i>iIndex</i> ed instance of the attribute from the dictionary as a sequence of bytes. If the dictionary does not contain the attribute (or that many instances of it), returns 0 instead. If <i>pbMore</i> is nonzero, the getBytes method sets it to TRUE if there are more instances of the attribute after the one returned, otherwise to FALSE. Use this to determine whether to make another call to getBytes to retrieve other instances of the attribute.	

Method	Syntax
getBytesByType	const abytes_t *pDict->getBytesByType(dex_AttributeDictionary_t *pDict, const abytes_t *pszAttribute, int iIndex, abool_t *pbMore)
Returns the value of the <i>iIndex</i> ed instance of the attribute from the dictionary as a sequence of bytes. If the dictionary does not contain the attribute (or that many instances of it), 0 is returned instead. If <i>pbMore</i> is nonzero, sets the variable pointed to TRUE if there are more instances of the attribute after the one returned, otherwise to FALSE. Use this to determine whether to make another call to get to retrieve other instances of the attribute.	
getByType	const char *pDict->getByType(dex_AttributeDictionary_t *pDict, const abytes_t *pszAttribute, int iIndex, abool_t *pbMore)
Returns the value of the <i>iIndex</i> ed instance of the attribute from the dictionary, represented as a string. If the dictionary does not contain the attribute (or that many instances of it), returns the empty string instead. If <i>pbMore</i> is nonzero, the getByType method sets <i>pbMore</i> to TRUE if there are more instances of the attribute after the one returned, otherwise to FALSE. Use this to determine whether to make another call to getByType to retrieve other instances.	
getOption	const char *getOption(dex_AttributeDictionary_t *pDict, int iArgType, ...)
Gets the data for an option as a string. Note that this function always returns a pointer to a string, which can be zero length if the option does not exist or has length zero. To find out if the option exists, use getOptionBytes or specify DEX_INSTANCE_COUNT.	
getOptionBytes	const abytes_t *getOptionBytes(dex_AttributeDictionary_t *pDict, int iArgType, ...)
Gets the data for an option as a sequence of bytes. Note that this function returns a null pointer if the option does not exist, and an abytes_t with a zero-length buffer if the option exists but is zero bytes long.	
getType	const abytes_t * pDict->getType(dex_AttributeDictionary_t * pDict, const char* pszAttribute)
Returns a pointer to the byte sequence defining the attribute, if the attribute name matches a configured attribute, otherwise 0.	
isValidisV4isV6	abool_t isValid(dex_AttributeDictionary_t *pDict) abool_t isV4(dex_AttributeDictionary_t *pDict) abool_t isV6(dex_AttributeDictionary_t *pDict)

Method	Syntax
<p>The isValid method returns TRUE if there is a request or response (depending on the dictionary passed in); FALSE otherwise. Extensions such as lease-state-change can use this method to determine whether a dictionary is available.</p> <p>The isV4 method returns TRUE if this extension is being called for a DHCPv4 packet; FALSE otherwise. Calling this method from an init-entry routine returns FALSE.</p> <p>The isV6 method returns TRUE if this extension is being called for a DHCPv6 packet; FALSE otherwise. Calling this method from an init-entry routine returns FALSE.</p>	
log	abool_t <i>pDict</i> -> log (dex_AttributeDictionary_t * <i>pDict</i> , int <i>eLevel</i> , const char * <i>pszFormat</i> , ...)
<p>Returns a message in the DHCP server logging system. The <i>eLevel</i> should be one of DEX_LOG_ERROR, DEX_LOG_WARNING, or DEX_LOG_INFO. The <i>pszFormat</i> is treated as a printf style format string, and it, along with the remaining arguments, are formatted and sent to the logging system at the specified level.</p> <p>Note Use the DEX_LOG_ERROR and DEX_LOG_WARNING levels sparingly, because the server flushes its log file with messages logged at these levels. Using these levels for messages that are likely to occur frequently (such as client requests) can have severe impact on disk I/O performance.</p>	
moveToOption	abool_t moveToOption (dex_AttributeDictionary_t * <i>pDict</i> , int <i>iArgType</i> , ...)
<p>Sets the context for subsequent get, put, and remove option operations. Note that the context can become invalid if the option is removed (such as with removeOption).</p>	
put	abool_t <i>pDict</i> -> put (dex_AttributeDictionary_t * <i>pDict</i> , const char * <i>pszAttribute</i> , const char * <i>pszValue</i> , int <i>iIndex</i>)
<p>Converts <i>pszValue</i> to a sequence of bytes, according to the definition of <i>pszAttribute</i> in the server configuration. Associates that sequence of bytes with the attribute in the dictionary. If <i>iIndex</i> is the special value DEX_REPLACE, replaces any existing instances of the attribute with a single value. If the special value DEX_APPEND, it appends a new instance of the attribute to its list. If the special value DEX_AUGMENT, puts the attribute only if there is not one already. Otherwise, inserts a new instance at the position indicated. Returns TRUE unless the attribute name does not match any configured attributes or the value could not be converted to a legal value.</p>	
putBytes	abool_t <i>pDict</i> -> putBytes (dex_AttributeDictionary_t * <i>pDict</i> , const char * <i>pszAttribute</i> , const abytes_t * <i>pszValue</i> , int <i>iIndex</i>)
<p>Associates <i>pszValue</i> with the <i>pszAttribute</i> in the dictionary. If <i>iIndex</i> is the special value DEX_REPLACE, replaces any existing instances of the attribute with a single new value. If the special value DEX_APPEND, appends a new instance of the attribute to its list. If the special value DEX_AUGMENT, puts the attribute only if there is not one already. Otherwise, inserts a new instance at the position indicated. Returns TRUE unless the attribute name does not match a configured one.</p>	

Method	Syntax
putBytesByType	abool_t <i>pDict</i> -> putBytesByType (dex_AttributeDictionary_t * <i>pDict</i> , const abytes_t * <i>pszAttribute</i> , const abytes_t * <i>pszValue</i> , int <i>iIndex</i>)
Associates <i>pszValue</i> with the <i>pszAttribute</i> in the dictionary. If <i>iIndex</i> is the special value DEX_REPLACE, replaces any existing instances of the attribute with the new value. If the special value DEX_APPEND, appends a new instance of the attribute to its list. If the special value DEX_AUGMENT, puts the attribute only if there is not one already. Otherwise, inserts a new instance of the attribute at the position indicated.	
putByType	abool_t <i>pDict</i> -> putByType (dex_AttributeDictionary_t * <i>pDict</i> , const abytes_t * <i>pszAttribute</i> , const char * <i>pszValue</i> , int <i>iIndex</i>)
Converts <i>pszValue</i> to a sequence of bytes, according to the definition of <i>pszAttribute</i> in the server configuration. Associates that sequence of bytes with the attribute in the dictionary. If <i>iIndex</i> is the special value DEX_REPLACE, replaces any existing instances of the attribute with a single new value. If the special value DEX_APPEND, appends a new instance of the attribute to its list. If the special value DEX_AUGMENT, puts the attribute only if there is not one already. Otherwise, inserts a new instance at the position indicated.	
putOption	abool_t putOption (dex_AttributeDictionary_t * <i>pDict</i> , const char * <i>pszValue</i> , int <i>iArgType</i> , ...)
Adds an option and its data or modifies the data for an option	
putOptionBytes	abool_t putOptionBytes (dex_AttributeDictionary_t * <i>pDict</i> , const abytes_t * <i>pValue</i> , int <i>iArgType</i> , ...)
Adds an option and its data or modifies the data for an option.	
remove	abool_t <i>pDict</i> -> remove (dex_AttributeDictionary_t * <i>pDict</i> , const char * <i>pszAttribute</i> , int <i>iIndex</i>)
Removes the attribute from the dictionary. If <i>iIndex</i> is the special value DEX_REMOVE_ALL, removes any existing instances of the attribute. Otherwise, removes the instance at the position indicated. Returns TRUE, even if the dictionary did not contain that attribute at that index, unless the attribute name does not match any configured one.	
removeByType	abool_t <i>pDict</i> -> removeByType (dex_AttributeDictionary_t * <i>pDict</i> , const abytes_t * <i>pszAttribute</i> , int <i>iIndex</i>)
Removes the attribute from the dictionary. If <i>iIndex</i> is the value DEX_REMOVE_ALL, removes any existing instances of the attribute. Otherwise, removes the instance at the position indicated. Always returns TRUE, even if the dictionary does not contain that attribute at that index.	
removeOption	abool_t removeOption (dex_AttributeDictionary_t * <i>pDict</i> , int <i>iArgType</i> , ...)
Removes an option. Note that if you omit DEX_INDEX, a DEX_INDEX of DEX_REMOVE_ALL is assumed (this removes the entire option).	

Method	Syntax
setObject	abool_t setObject(dex_AttributeDictionary_t *pDict, int iObjectType, int iObjArgType, ...)
Sets the object for get , put , and remove methods, and alters the message on which the new option methods operate. DHCPv6 extensions primarily use this method to access the leases and prefixes available for the client and link, or to get message header fields or options from relay packets. Unlike in DHCPv4, where one lease and scope are associated with a response, a DHCPv6 response can involve several leases and prefixes. Returns TRUE if the object exists; FALSE otherwise. For sample usage, see the Handling Object Data, on page 501 .	
Note	For leases not associated with the current client, only minimal information is available.
trace	abool_t pDict->trace(dex_AttributeDictionary_t *pDict, int iLevel, const char *pszFormat, ...)
Returns a message in the DHCP server packet tracing system. At level 0, no tracing occurs. At level 1, it traces only that the server received the packet and sent a reply. At level 4, it traces everything. The remaining arguments are concatenated and sent to the tracing system at the specified level. The default tracing is set using the DHCP server <i>extension-trace-level</i> attribute.	

DEX Environment Dictionary Methods

The environment dictionary uses active commands, called *methods*, with which you can change and access the dictionary values. The table below lists the methods to use with the environment dictionary. In this case, you can define the *pDict* variable as **pEnviron**, as in:

```
pEnviron->log( pEnviron, DEX_LOG_INFO, "Environment hello world");
```

Table 78: DEX Environment Dictionary Methods

Method	Syntax
allocateMemory	void *pDict->allocateMemory(dex_EnvironmentDictionary_t *pDict, unsigned int iSize)
Allocates memory for extensions that persists only for the lifetime of this request.	
clear	void pDict->clear(dex_EnvironmentDictionary_t *pDict)
Removes all entries from the dictionary.	
containsKey	abool_t pDict->containsKey(dex_EnvironmentDictionary_t *pDict, const char *pszKey)
Returns TRUE if the dictionary contains the key, otherwise FALSE.	
firstKey	const char *pDict->firstKey(dex_EnvironmentDictionary_t *pDict)

Method	Syntax
Returns the name of the first key in the dictionary. Note that the keys are not stored sorted by name. If a key does not exist, returns zero.	
get	const char *pDict->get(dex_EnvironmentDictionary_t *pDict, const char *pszKey)
Returns the value of the key from the dictionary. If a key does not exist, returns the empty string.	
isEmpty	bool_t pDict->isEmpty(dex_EnvironmentDictionary_t *pDict)
Returns TRUE if the dictionary has 0 entries, otherwise FALSE.	
log	bool_t pDict->log(dex_EnvironmentDictionary_t *pDict, int eLevel, const char *pszFormat, ...)
Returns a message in the DHCP server logging system. The <i>eLevel</i> should be one of DEX_LOG_ERROR, DEX_LOG_WARNING, or DEX_LOG_INFO. The <i>pszFormat</i> is treated as a printf style format string, and it, along with the remaining arguments, are formatted and sent to the logging system at the specified level. Note Use the DEX_LOG_ERROR and DEX_LOG_WARNING levels sparingly, because the server flushes its log file with messages logged at these levels. Using these levels for messages that are likely to occur frequently (such as client requests) can have severe impact on disk I/O performance.	
nextKey	const char *pDict->nextKey(dex_EnvironmentDictionary_t *pDict)
Returns the name of the next key in the dictionary that follows the key returned in the last call to firstKey or nextKey . If a key does not exist, returns zero.	
put	bool_t pDict->put(dex_EnvironmentDictionary_t *pDict, const char *pszKey, const char* pszValue)
Associates a value with the key, replacing an existing instance of the key with the new value.	
remove	bool_t pDict->remove(dex_EnvironmentDictionary_t *pDict, const char *pszKey)
Removes the key and the associated value from the dictionary. Always returns TRUE, even if the dictionary did not contain the key.	
size	int pDict->size(dex_EnvironmentDictionary_t *pDict)
Returns the number of entries in the dictionary.	
trace	bool_t pDict->trace(dex_EnvironmentDictionary_t *pDict, int iLevel, const char *pszFormat, ...)

Method	Syntax
Returns a message in the DHCP server packet tracing system. At level 0, no tracing occurs. At level 1, it traces only that the server received the packet and sent a reply. At level 4, it traces everything. The remaining arguments are concatenated and sent to the tracing system at the specified level. The default tracing is set using the DHCP server <i>extension-trace-level</i> attribute.	

Differences in get, put, Option, Bytes, and OptionBytes Methods

There are differences among the following DEX extension methods:

- **get** and **put**
- **getOption** and **putOption**
- **getBytes** and **putBytes**
- **getOptionBytes** and **putOptionBytes**

The **get** and **getOption** methods return the requested information formatted as a string. The server converts the data to the string depending on the expected data type for the dictionary item. If the data type is unknown, the server returns the data in blob string format.

The **getBytes** and **getOptionBytes** methods return the requested information as the raw bytes (a pointer to a buffer and the size of that buffer). The server should have to read this buffer only, and it contains only the data from the option (no null terminator has been added, for example).

The **put** and **putOption** methods expect the data to be written as a formatted string. The server converts the data from the string depending on the expected data type for the dictionary item. If the data type is unknown, it is expected to be in blob string format.

The server passes raw bytes to the **putBytes** and **putOptionBytes** methods (a pointer to a buffer and the size of that buffer). The server only reads these bytes.

Differences in get, put, remove, and ByType Methods

There are differences among the following DEX extension methods:

- **get**, **put**, and **remove**
- **getByType**, **putByType**, and **removeByType**

The server passes the **get**, **put**, and **remove** methods the name of the desired data item as a string. This requires that the server map the string to its internal data tables.

The server passes the **getByType**, **putByType**, and **removeByType** methods an internal data table reference, which the server must have previously obtained (such as in the extension init-entry) by calling the **getType** method on the string. This speeds processing for extensions, which can be important in applications requiring high performance.



Note

The internal data table that the **getType** method references is the same whether requested for the Request or Response dictionary. There is no need for separate **getType** calls on each dictionary for the same data item name.

Table 79: DEX *iObjectType* Values

i Object Type	Description
General definition: Object for which the context is to be changed.	
DEX_LEASE	Changes the lease (and prefix) context. Response dictionary only. Allows <i>iObjTypeArg</i> : DEX_BY_IPV6ADDRESS DEX_BY_IPV6PREFIX DEX_BY_INSTANCE DEX_INITIAL
DEX_MESSAGE	Changes the message context to a relay message or the client message. Request and response dictionaries. Allows <i>iObjArgType</i> : DEX_INITIAL DEX_RELAY DEX_BY_NUMBER
DEX_PREFIX	Changes the prefix context, but does not change the lease context. Response dictionary only. Allows <i>iObjTypeArg</i> : DEX_BY_IPV6ADDRESS DEX_BY_IPV6PREFIX DEX_BY_INSTANCE DEX_BY_NAME DEX_INITIAL

Table 80: DEX *iObjArgType* Values

iObjArgType	Description
General definition: By what means the context is to be changed.	
DEX_BY_INSTANCE	Used with DEX_LEASE or DEX_PREFIX <i>iObjectType</i> . Requires that int follows to specify the instance number (starting with 0). Used to walk through the list of all available objects, but only through the list of objects applicable to the current request or response: for DEX_LEASE, the leases for that client (if any); for DEX_PREFIX, the prefixes on the current link (if any). Used with DEX_MESSAGE, a synonym for DEX_RELAY.

iObjArgType	Description
DEX_BY_IPV6ADDRESS	Used with DEX_LEASE and DEX_PREFIX <i>iObjectType</i> only. Requires that const unsigned char * follows to specify the 16-byte address.
DEX_BY_IPV6PREFIX	Used with DEX_LEASE or DEX_PREFIX <i>iObjectType</i> . Requires that const unsigned char * follows to specify a 17-byte prefix buffer (16-byte address followed by a 1-byte prefix length).
DEX_BY_NAME	Used with the DEX_PREFIX <i>iObjectType</i> only. Requires that a const char * follows to specify the name of the desired object.
DEX_INITIAL	Resets the context back to the original for the request or response, and has no additional argument. Sets the lease and prefix (DEX_LEASE), prefix (DEX_PREFIX), or message (DEX_MESSAGE) to what it was when the extension was originally called (which can be none).
DEX_RELAY	Used with DEX_MESSAGE <i>iObjectType</i> only. Requires that int follows to specify the relay (0 specifies the relay closest to the client). To set the message context back to the client, use setObject(pDict, DEX_MESSAGE, DEX_INITIAL) .

iArg Type	Description
General definition: Action and argument that follows the context. There can be any number of <i>iArgType</i> instances in the calls.	
DEX_ARG_ARRAY	<p>Requires that a pointer to an array of dex_OptionsArgs_t follow, and is an alternative to specifying the argument list. Each dex_OptionsArgs_t structure has two fields:</p> <ul style="list-style-type: none"> • <i>iArgType</i> —One of the <i>iArgType</i> DEX values in this table. • <i>pData</i> —Data (integer), pointer to the data (for strings and other data types), or ignored (if the <i>iArgType</i> takes no arguments). <p>Note that once the server encounters the DEX_ARG_ARRAY (in an argument list or in an array of dex_OptionsArgs_t), it ignores any subsequent arguments in the original list.</p>
DEX_END	Note Required, has no additional argument, and marks the end of the argument list.

iArg Type	Description
DEX_ENTERPRISE_NAME	Requires that const char * follow to specify the option definition set name, from which the server extracts the enterprise-id to get the vendor option data. Valid only for vendor-identifying options. Requires that the vendor option definition set exists.
DEX_ENTERPRISE_ID	Requires that int follow to specify the enterprise-id for the vendor.
DEX_HOME	Moves the context back to the client or relay message options. Has no additional argument. Always returns success. If used, must be the first <i>iArgType</i> . Valid only for getOption , getOptionBytes , and moveToOption methods.
DEX_INDEX	<p>Requires that int follow with the index of the option data (if any array of data is to be acted on). If omitted, index 0 is assumed, except for removeOption, in which case DEX_REMOVE_ALL is assumed. Use the special value DEX_RAW to get, put, or remove the entire option data. However, for the DHCPv4 Vendor-Identifying Vendor Options (RFC 3925 and RFC 4243), DEX_RAW returns the data for only one vendor (based on the instance or enterprise-id) and not that for the entire option.</p> <p>The DEX_RAW special value accesses the entire option (or suboption) data. It provides consistent access to the data, regardless of what the option definitions might specify in terms of the data type and repeat counts of the data type. It is recommended for general-purpose extensions that decode the data.</p> <p>Use the special values DEX_REPLACE (replace a value), DEX_APPEND (add to end), and DEX_AUGMENT (add if no value currently exists) with putOption and putOptionBytes methods, which operate the same as the put, putByType, putBytes, and putBytesByType methods. Use DEX_REMOVE_ALL for removeOption to remove the option completely.</p>
DEX_INDEX_COUNT	Results in an int value returned with the count of the number of indexed entries of the option, rather than the option data. Has no additional argument, and cannot be used with DEX_INDEX or DEX_INSTANCE_COUNT. DEX_END must follow. Valid only for getOption and getOptionBytes .

iArg Type	Description
DEX_INSTANCE	Requires that int follow to specify the instance of the option (valid only for DHCPv6 options, which can have more than one instance). 0 specifies the first instance.
DEX_INSTANCE_COUNT	Results in an int value returned with the count of the number of instances of the option, rather than the option data. Has no additional argument and cannot be used with DEX_INSTANCE. DEX_END must follow. Valid only for getOption and getOptionBytes .
DEX_MORE	Requires that bool_t * follow to specify the location at which a more flag is to be written. This location is set to TRUE if more array items exist beyond the index that DEX_INDEX specified. Valid only for getOption and getOptionBytes methods.
DEX_MOVE_TO	Leaves the context at the option or suboption immediately preceding DEX_MOVE_TO. Has no additional argument. If omitted, the context does not change. Use moveToOption to move the context without getting any data. Valid only for getOption and getOptionBytes methods. Note An attempt to move to an option or suboption that does not exist logs an error. Use moveToOption if your extension did not previously confirm that the option exists.
DEX_OPTION_NAME	Requires that const char * follow to specify the desired option name. Option names should be in the dhcpv4-config or dhcpv6-config option definition set.
DEX_OPTION_NUMBER	Requires that const char * follow to specify the desired option name. Option names should be in the dhcpv4-config or dhcpv6-config option definition set.
DEX_PARENT	Moves the context to the parent option. Has no additional argument. It does not move beyond the client or relay message and returns FALSE if the context does not change. If used, must be the first <i>iArgType</i> . Valid only for getOption , getOptionBytes , and moveToOption methods.
DEX_SUBOPTION_NAME	Requires that const char * follow to specify the name of the desired suboption. Suboptions must be in the current option definition.

iArg Type	Description
DEX_SUBOPTION_NUMBER	Requires that int follow to specify the desired suboption number. Suboption numbers should be in the current option definition, although there is no requirement that a definition exists. However, if the suboption does not exist, it is assumed to be a byte blob of data.
DEX_VENDOR_NAME	Requires that const char * follow to specify the vendor string. The string serves only to find the appropriate option definition set.

Handling Objects and Options

The following sections describe specialized ways of handling DHCP objects and options in extensions.

Using Object and Option Handling Methods

Extensions can call methods to set DHCP objects, and get, move to, put, and remove DHCP options. The methods are **setObject**, **getOption**, **moveToOption**, **putOption**, and **removeOption** methods in Tcl and C/C++.

These new callback methods were introduced primarily to provide support for DHCPv6. However, you can use the option-related functions for DHCPv4. In fact, it is recommended to use these methods for DHCPv4, because they provide richer access to options than the original **get[Bytes]**, **get[Bytes]ByType**, **put[Bytes]**, **put[Bytes]ByType**, and **remove[ByType]** methods.



Tip See [DEX Request and Response Dictionary Methods, on page 488](#) for the different usages of some of these methods in C/C++.

For DHCPv6, you must use the **setObject**, **getOption**, **moveToOption**, **putOption**, and **removeOption** methods to access options. The **setObject** method was introduced for DHCPv6, because there can be many leases, prefixes, and messages (client or multiple relay) that an extension might want to access. So, **setObject** serves to set the context for subsequent calls to get request and response dictionary data items and options. When the server calls an extension, the context is set to the current lease (if applicable), prefix (if applicable), and client message. For example, when the server calls the **pre-packet-encode** extension point, only the request and response dictionary message context is valid, and set to the corresponding client message, because there is no lease or prefix associated with this extension point. However, when the server calls the **lease-state-change** extension point, it sets the response dictionary lease context to the lease on which the state has changed, sets the response dictionary prefix context to the prefix for the lease, and sets the request and response dictionary message context to the corresponding client message.

Options and Suboptions in C/C++

Some C/C++ extensions provide specialized argument type values to handle DHCP options and suboptions. The **DEX_OPTION_*** argument type specifies to use the standard DHCPv4 or DHCPv6 option definition set

and not the definitions under an option (or suboption). So, `DEX_OPTION_*` means that the server looks up the option name or number in the standard DHCPv4 or DHCPv6 option definition set, whereas `DEX_SUBOPTION_*` means that the server looks up the suboption name or number of the current option definition (if any).

Thus, when you access options in DHCPv6, you often use `DEX_OPTION_*` followed by `DEX_OPTION_*` when options are encapsulated. You would use `DEX_SUBOPTION` when looking at vendor options. For DHCPv4, you would use `DEX_OPTION` at the client packet level, and then `DEX_SUBOPTION` perhaps one or more times, depending on the nesting level. Generally, only options have enterprise numbers or vendor names, but there is no prohibition on this. The option definition sets determine what is valid (although one can walk off definitions, at which point everything is treated as binary bytes and thus it limits what is possible, and you cannot use the option or suboption names, but must use numbers).

The option ordering rules for the `getOption`, `moveToOption`, `putOption`, and `removeOption` methods are similar to the `request` expression syntax. The ordering generally consists of:

- Preamble clause (`[parent | home]`)
- Option clause (`option [vendor | enterprise] [instance]`)
- Suboption clause (`suboption [vendor | enterprise] [instance]`)
- End clause (`[instance-count | index-count | [index] [more] end]`)

You can construct calls by using a preamble clause, followed by zero or more option clauses, followed by zero or more suboption clauses (which may themselves be followed by option and suboption clauses), followed by an end clause. Note that some things are possible only through a `get` method (such as `instance-count`, `index-count`, and `more`), and `move-to` can appear anywhere to move the context to the current option or suboption.

The option definition determines its data format, which can differ from what the older functions return for a specific option. To handle specific options:

- For the vendor class options (`v-i-vendor-class` [124] for DHCPv4 and `vendor-class` [16] for DHCPv6), if you ask for a specific instance of the option (instead of by enterprise-id or name), the only way to get the enterprise-id is to ask for the raw data (`DEX_INDEX` with `DEX_RAW`).
- For the DHCPv4 vendor options (`v-i-vendor-class` [124] and `v-i-vendor-opts` [125]), operating on the raw data (`DEX_INDEX` with `DEX_RAW`) only applies to an instance (preset value 0) of that option, not the entire option. There is no way to get the entire data for this option, which means that you cannot use `putOption` for the entire data. This is not an issue with the DHCPv6 vendor options, because these are separate options.
- If one of the DHCPv4 vendor options (124 or 125) is not formatted properly, the entire data is returned as a blob (if you asked for instance 0 and did not specify a particular enterprise-id). However, if an extension tries to use `putOption`, depending on the operation, that data might be appended to the existing data, and the result will be formatted incorrectly.
- For the vendor options, if there is no option, `putOption(pDict, "01:02", DEX_OPTION_NUMBER, 124, DEX_END)` fails because no enterprise-id is available. However, `putOption(pDict, "00:00:00:09:04:03:65:66:67", DEX_OPTION_NUMBER, 124, DEX_END)` will work because it is assumed that 00:00:00:09 is the enterprise-id and the bytes following it starting with 04 are the length of the option data of that enterprise-id. Note that the length byte is validated in this case, and `putOption` fails if it does not have the correct length. The recommended way to add this data is to use `putOption(pDict, "65:66:67", DEX_OPTION_NUMBER, 124, DEX_ENTERPRISE_ID, 9, DEX_END)`.

Examples of Option and Object Method Calls

These sections include some examples of how to use methods to handle DHCP option and object data.

Handling Vendor Class Option Data

For DHCPv4, to include the Vendor-Identifying Vendor Class option (124) data for two enterprise-ids in the response to the client, here is some sample Tcl code that uses the **putOption** method:

```
$response putOption 65:66:67 option 124 enterprise 999998
#adds "abc" (65:66:67) under enterprise-id 999998
$response putOption 68:69:6a:6b option v-i-vendor-class enterprise 999998 index append
#appends "defg" (68:69:6a:6b) under the same enterprise-id
$response putOption 01:02:03:04 option 124 enterprise 999999
#adds 01:02:03:04 under enterprise-id 999999
```

To get the options, use the **getOption** method:

```
$response getOption option v-i-vendor-class instance-count
#returns 2 because there were two instances added (enterprise id 999998 and enterprise id
999999)
$response getOption option 124
#returns index 0 of instance 0, which is 65:66:67
$response getOption option 124 index-count
#returns 2 because there were two vendor classes added for the first enterprise id (999998)
$response getOption option 124 index raw
#returns 00:0f:42:3e:09:03:65:66:67:04:68:69:6a:6b for the complete encoding of the
enterprise-id 999998 data (see RFC 3925)
$response getOption option 124 index 1
#returns 68:69:6a:6b
$response getOption option 124 instance 1 index-count
#returns 1 because there is only one vendor class
$response getOption option 124 instance 1 index raw
#returns 00:0f:42:3f:05:04:01:02:03:04 for the complete encoding of the enterprise-id
999999 data (see RFC 3925)
$response getOption option 124 enterprise 999999
#returns 01:02:03:04
```

To remove the data, two **removeOption** calls are necessary because of the two separate enterprise-ids:

```
$response removeOption option 124
$response removeOption option 124
```

Handling Object Data

Suppose that at the **pre-packet-encode** extension point you want to extract data for all of the leases for the client. Here is sample Tcl code that uses the **setObject** method:

```
proc logleasesinit { request response environ } {
    if { [$environ get "extension-point"] == "initialize" } {
        # Set up for DHCPv6 only
        $environ put dhcp-support "v6"
        $environ put extension-extensionapi-version 2
    }
}
```

```

}
proc logleases { request response environ } {
  for { set i 0 } { 1 } { incr i } {
    # Set context to next lease
    if { ![response setObject lease $i] } {
      # Lease does not exist, so done
      break
    }
    # Log the lease address, prefix name, and prefix address
    $environ log LOG_INFO "Lease [response get lease-ipaddress], Prefix\
      [response get lease-prefix-name] - [response get prefix-address]"
  }
  # Restore the lease context to where we started
  $response setObject lease initial
  # Do other things...
}

```

The C++ equivalent code for this might be:

```

// Print the current leases for the client
for( int i=0; ; i++ ) {
  if( !pRes->setObject( pRes, DEX_LEASE, DEX_BY_INSTANCE, i ) )
    break;
  const char *pszLeaseAddress =
    pRes->get( pRes, "lease-ipaddress", 0, 0 );
  if( pszLeaseAddress == 0 )
    pszLeaseAddress = "<error>";
  const char *pszPrefixName =
    pRes->get( pRes, "prefix-name", 0, 0 );
  if( pszPrefixName == 0 )
    pszPrefixName = "<error>";
  pEnv->log( pEnv, DEX_LOG_INFO,
    "Lease %s, Prefix %s",
    pszLeaseAddress, pszPrefixName );
}

```



INDEX

A

- ablock-edit [98](#)
- ablock-list [95](#)
- acl command (CLI) [250](#)
 - create [250](#)
- acl-edit [250](#)
- acl-list [250](#)
- acl-pull-replica [250](#)
- acl-pull-replica-report [250](#)
- acl-pull-replica-run [250](#)
- acl-push-data [250](#)
- acl-push-data-report [250](#)
- ACLs [250](#)
 - See access control lists (ACLs) [250](#)
- addr-dhcp-search, dhcp-search [197](#)
- addr-dhcp-util-tree [108](#)
- addr-lease-history [225](#)
- address allocation [113, 116, 117](#)
 - attributes [116](#)
 - in scopes [117](#)
 - round-robin [113](#)
- address blocks [49, 91, 94, 95, 97](#)
 - adding [95](#)
 - administrator role [91](#)
 - delegating [97](#)
 - embedded policies [49](#)
 - when to add [94](#)
- address blocks, DHCP [49, 50, 93](#)
 - creating [49](#)
 - default subnet size [49](#)
 - orphaned leases [50](#)
 - policies, associating [49](#)
- address ranges [99, 120](#)
 - scopes [120](#)
 - subnetsubnets [99](#)
 - address ranges [99](#)
- address space [91, 100](#)
 - unified [100](#)
- address usage reports [223](#)
 - running [223](#)
- address-block-policy command (CLI) [49](#)
 - delete [49](#)
 - get [49](#)
 - getOption [49](#)
- address-block-policy command (CLI) (*continued*)
 - listOptions [49](#)
 - listVendorOptions [49](#)
 - setVendorOption [49](#)
 - show [49](#)
 - unset [49](#)
 - unsetOption [49](#)
 - unsetVendorOption [49](#)
- address-space [102](#)
- address-space-tree [102](#)
- addresses [91, 126](#)
 - dynamic [91](#)
 - IPv6 [126](#)
 - static [91](#)
- allocation priority, scopes [114](#)
 - address allocation [114](#)
 - algorithm [114](#)
 - limitation-id [114](#)
 - priority [114](#)
- and, DHCP expression [322](#)
- arithmetic functions, DHCP expressions [322](#)
- arp-cache-timeout, DHCP option [411](#)
- as-blob, DHCP expression [322](#)
- as-sint, DHCP expression [322](#)
- as-string, DHCP expression [322](#)
- as-uint, DHCP expression [322](#)
- ash, DHCP expression [322](#)
- associated-ip, DHCP option [428](#)
- Asynchronous Transfer Mode (ATM) [348](#)
- AT_BLOB, option validation [445](#)
- AT_BOOL, option validation [445](#)
- AT_CONTAINER6, option validation [445](#)
- AT_DATE, option validation [445](#)
- AT_DNSNAME, option validation [445](#)
- AT_INT, option validation [445](#)
- AT_INT8, option validation [445](#)
- AT_INTI, option validation [445](#)
- AT_IP6ADDR, option validation [445](#)
- AT_IPADDR, option validation [445](#)
- AT_MACADDR, option validation [445](#)
- AT_MESSAGE, option validation [445](#)
- AT_NOLEN, option validation [445](#)
- AT_NSTRING, option validation [445](#)
- AT_OVERLOAD, option validation [445](#)
- AT_RANGEBYTE, option validation [445](#)

AT_RANGESHORT, option validation [445](#)
 AT_RDNSNAME, option validation [445](#)
 AT_SHORT, option validation [445](#)
 AT_SHRTI, option validation [445](#)
 AT_SINT, option validation [445](#)
 AT_SINTI, option validation [445](#)
 AT_SSHORT, option validation [445](#)
 AT_SSHRTI, option validation [445](#)
 AT_STIME, option validation [445](#)
 AT_STRING, option validation [445](#)
 AT_TIME, option validation [445](#)
 AT_TYPECNT, option validation [445](#)
 AT_VENDOR_CLASS, option validation [445](#)
 AT_VENDOR_OPTS, option validation [445](#)
 AT_ZEROSIZE, option validation [445](#)
 auth, DHCPv6 option [420](#)
 authentication, DHCP option [428](#)
 auto-configure, DHCP option [428](#)

B

bar-chart [395](#)
 bcmcs-server-a, DHCPv6 option [420](#)
 bcmcs-server-d, DHCPv6 option [420](#)
 bcmcs-servers-a, DHCP option [417](#)
 bcmcs-servers-d, DHCP option [417](#)
 bit-and, DHCP expression [322](#)
 bit-andc1, DHCP expression [322](#)
 bit-andc2, DHCP expression [322](#)
 bit-eqv, DHCP expression [322](#)
 bit-not, DHCP expression [322](#)
 bit-or, DHCP expression [322](#)
 bit-orc1, DHCP expression [322](#)
 bit-orc2, DHCP expression [322](#)
 bit-xor, DHCP expression [322](#)
 boot-file, DHCP option [417](#)
 boot-size, DHCP option [407](#)
 BOOTP [51](#), [52](#), [53](#), [88](#), [122](#), [123](#)
 BOOTP Relay [53](#)
 clients, moving/decommissioning [122](#)
 configuring [51](#)
 dynamic [52](#), [123](#)
 enabling [52](#)
 scopes, scope command (CLI) [123](#)
 enable dynamic-bootp [123](#)
 enabling, disabling [52](#)
 failover, DHCP [88](#)
 BOOTP clients [88](#)
 file, DHCP packet, field [51](#)
 scopes, enabling for scopes [122](#)
 BOOTP, enabling [122](#)
 siaddr, file, sname [51](#)
 static [88](#)
 broadcast-address, DHCP option [410](#)
 byte, DHCP expression [322](#)

C

C/C++ [354](#), [359](#), [360](#)
 API [360](#)
 extensions [354](#), [359](#)
 cablelabs-125, DHCP option [449](#)
 cablelabs-17, DHCPv6 option [449](#)
 cablelabs-client-configuration, DHCP option [428](#)
 ccm-reservation-add [203](#)
 ccm-reservation-edit [203](#)
 ccm-reservation-list [203](#)
 ccm-reservation-view [202](#)
 check-lease-acceptable, DHCP [387](#)
 check-lease-acceptable, extension point, DHCP [27](#)
 children [98](#)
 address blocks [98](#)
 subnets [98](#)
 cisco-auto-configure, DHCP option [428](#)
 cisco-client-last-transaction-time, DHCP option [428](#)
 cisco-client-requested-host-name, DHCP option [428](#)
 cisco-leased-ip, DHCP option [428](#)
 cisco-vpn-id, DHCP option [428](#)
 classless-static-route, DHCP option [428](#)
 client command (CLI) [46](#), [292](#), [294](#), [295](#), [315](#)
 create [292](#)
 delete [292](#)
 listnames [292](#)
 set [46](#), [292](#), [294](#), [295](#), [315](#)
 authenticate-until [295](#)
 client-class-name [292](#)
 client-lookup-id [315](#)
 override-vpn [46](#)
 policy-name [294](#)
 selection-criteria [292](#)
 client command CLI [298](#)
 set [298](#)
 over-limit-client-class-name [298](#)
 client ID [210](#)
 overriding, client-class command (CLI) [210](#)
 set [210](#)
 override-client-id [210](#)
 client reservations [199](#)
 client-class command (CLI) [46](#), [209](#), [285](#), [287](#), [298](#)
 create [285](#)
 delete [285](#)
 list [285](#)
 listnames [285](#)
 set [46](#), [209](#), [285](#), [287](#), [298](#)
 add-to-environment-dictionary [209](#)
 default-vpn [46](#)
 host-name [287](#)
 limitation-id [298](#)
 over-limit-client-class-name [298](#)
 override-vpn [46](#)
 selection-criteria [285](#)
 show [285](#)

- client-class command (CLI) **315**
 - set **315**
 - limitation-key **315**
- client-class-policy command (CLI) **288**
 - set **288**
 - setLeaseTime **288**
 - setOption **288**
 - setV6Option **288**
 - setV6VendorOption **288**
 - setVendorOption **288**
 - show **288**
- client-classes **10, 11, 61, 210, 284, 286, 287, 288, 289, 290, 295**
 - client entries, skipping **295**
 - defining, client-class **284**
 - DHCPv6 client classes **286**
 - editing client classes **288**
 - embedded policies **288**
 - enabling **11**
 - failover synchronization effect **61**
 - host-name setting **287**
 - lookup ID, dhcp command (CLI) **210**
 - set **210**
 - client-class-lookup-id **210**
 - process **284**
 - processing order to determine **289**
 - quality of service, class of service, differentiated services **10**
 - RADIUS pool name, mapping **289**
 - troubleshooting **290**
 - user class identifier, mapping **289**
- client-data, DHCPv6 option **420**
- client-edit **293**
- client-fqdn, DHCP option **428**
- client-fqdn, DHCPv6 option **420**
- client-identifier, DHCP option **417**
- client-identifier, DHCPv6 option **420**
- client-last-transaction-time, DHCP option **428**
- client-policy command (CLI) **293**
 - set **293**
 - setLeaseTime **293**
 - setOption **293**
 - setV6Option **293**
 - setV6VendorOption **293**
 - setVendorOption **293**
 - show **293**
- clientclass-list **283**
- clients **11, 61, 291, 292, 293, 294, 295, 296, 297, 298**
 - authentication, limiting **295**
 - caching parameters **296**
 - configuring **291**
 - default **291**
 - DHCPv6 clients **293**
 - editing clients **293**
 - embedded policies **293**
 - failover synchronization effect **61**
 - lease request name **11**
- clients (*continued*)
 - listing, client command (CLI) **292**
 - list **292**
 - properties, client command (CLI) **292**
 - show **292**
 - provisioning **297**
 - subscribers, set by limitation-id **298**
 - unknown leases, provisional addresses, provisional leases,
 - one-shot **294**
 - Windows client properties **294**
- clt-time, DHCPv6 option **420**
- clusters **106**
 - subnet utilization **106**
 - poll-subnet-util-interval **106**
 - poll-subnet-util-offset **106**
 - poll-subnet-util-retry **106**
- cnr_keygen utility **252**
 - keys, generating secrets, TSIG keys **252**
- comment, DHCP expression **322**
- COMMUNICATIONS-INTERRUPTED, failover **71**
- concat, DHCP expression **322**
- configuration **231**
 - file, lease-notification **231**
- configuring client-classes **283**
 - See client-classes **283**
- configuring DHCP servers **112, 163**
 - See DHCP policies **163**
 - See scopes **112**
- configuring DNS update **256**
 - See DNS update **256**
- configuring DNS update maps **263**
 - See DNS update **263**
- configuring failover **55**
 - See failover, DHCP failover **55**
- configuring policies **166**
 - See policies **166**
- configuring virtual private networks **43**
 - See VPNs **43**
- cookie-servers, DHCP option **407**
- create-prefix-addr **151, 155**
 - link template expression **155**
 - prefix template expression **151**
- create-prefix-range **151, 155**
 - link template expression **155**
 - prefix template expression **151**
- create-prefix, link template expression **155**
 - link template expression **155**
 - prefix template expression **151**
- create-v6-option **151, 155**
 - link template expression **155**
 - prefix template expression **151**
- cron task (UNIX) **229**
- current-lease-list **224**

D

- dashboard **399, 400, 401, 402, 403, 404**
 - DHCP address utilization chart **402**

dashboard (*continued*)

- DHCP buffer capacity chart [401](#)
- DHCP DNS update activity chart [402](#)
- DHCP failover status chart [403](#)
- DHCP general indicators chart [404](#)
- DHCP response latency chart [401](#)
- DHCP server request activity [399](#)
- DHCP server response activity [400](#)

datatype, DHCP expression [322](#)default-ip-ttl, DHCP option [409](#)default-tcp-ttl, DHCP option [411](#)delegated address space, address space [92, 102](#)deployment cases [246](#)

- large enterprise network, DNS update [246](#)
- large deployments [246](#)

DHCP [1, 2, 3, 11, 15, 16, 17, 23, 25, 31, 44, 69, 114, 116, 124, 173, 214, 216, 248, 273, 297, 308, 352, 407, 428](#)administration [2](#)buffers, allocating [17](#)client-server model [1](#)clients [11, 44](#)IP address [44](#)ciaddr, DHCP field [44](#)MAC addresses [11](#)your IP address [44](#)yiaddr, DHCP field [44](#)custom options [173](#)custom options, adding [173](#)equal-priority-most-available [114, 116](#)ethernet addresses, interface cards [16](#)extension points, listing [352](#)hardware-unicast, unicast, enabling [17](#)lease-state updates to LDAP [308](#)leasequery, See leasequery [216](#)library path, dhcp command (CLI) [25](#)set [25](#)sms-library-path [25](#)log settings, dhcp command (CLI) [31](#)set [31](#)log-settings [31](#)option 82 [297](#)options [407, 428](#)policies [3](#)See policies [3](#)priority-address-allocation [116](#)request [11, 69](#)buffers, dhcp command (CLI) [69](#)set [69](#)max-dhcp-requests [69](#)processing [11](#)requests for other servers, ignoring, dhcp command (CLI) [214](#)enable [214](#)ignore-requests-for-other-servers [214](#)reverse zones, synthesizing [248](#)sample users [2](#)scopes, disabling for scopes [124](#)DHCP (*continued*)servers [15, 16, 23, 31, 273](#)configuring [15](#)forwarding [23](#)interface, removing address, dhcp-interface command (CLI) [16](#)interface, setting [16](#)logging [273](#)troubleshooting [31](#)SMS network discovery records, SMS [25](#)network discovery, dhcp command (CLI) [25](#)set [25](#)sms-network-discovery [25](#)dhcp command (CLI) [17, 21, 23, 25, 31, 46, 50, 52, 67, 73, 168, 224, 256, 268, 282, 285, 289, 295, 296, 298, 300, 303, 307, 315, 352, 403](#)attachExtension [23, 352](#)detachExtension [23, 352](#)disable [50](#)vpn-communication [50](#)enable [17, 50, 52, 168, 224, 268, 285, 295, 303, 307](#)client-class [285](#)defer-lease-extensions [17](#)delete-orphaned-leases [50](#)delete-orphaned-subnets [50](#)get-subnet-mask-from-policy [168](#)hardware-unicast [17](#)ip-history [224](#)return-client-fqdn-if-asked [268](#)save-lease-renewal-time [307](#)skip-client-lookup [295](#)update-dns-for-bootp [52](#)use-client-fqdn [268](#)use-client-fqdn-first [268](#)use-ldap-client-data [303](#)get [21](#)limitationList [300](#)set [17, 21, 25, 31, 46, 73, 256, 282, 289, 296, 298, 303, 315, 403](#)activity-summary-interval [31](#)client-cache-count [296](#)client-cache-ttl [296](#)client-class-lookup-id [298, 315](#)default-free-address-config [403](#)failover-recover [73](#)last-transaction-time-granularity [17](#)ldap-mode [303](#)log-settings [282](#)map-radius-class [289](#)max-dhcp-requests [17](#)max-dhcp-responses [17](#)max-ping-packets [17](#)max-waiting-packets [31](#)sms-lease-interval [25](#)sms-site-code [25](#)synthesize-reverse-zone [256](#)v6-default-free-address-config [403](#)vpn-communication [46](#)

- dhcp command (CLI) *(continued)*
 - setPartnerDown [67](#)
 - show [21](#)
 - unset [21](#)
 - updateSms [25](#)
- dhcp-address-block command (CLI) [46, 49](#)
 - set [46, 49](#)
 - default-subnet-size [49](#)
 - vpn [46](#)
 - vpn-id [46](#)
 - unset [49](#)
- dhcp-client-identifier, DHCP option [417](#)
- dhcp-cnclient-add [291](#)
- dhcp-cnclient-list [291](#)
- dhcp-cnembeddedpolicy-edit [170](#)
- dhcp-cnrpolicy-add [166](#)
- dhcp-cnrpolicy-edit [179](#)
- dhcp-cnrpolicy-list [166](#)
- dhcp-cnrpolicy-option-edit [179](#)
- dhcp-cnrrs-list [202](#)
- dhcp-cnrscope-edit [118](#)
- dhcp-cnrscope-lease-list [191](#)
- dhcp-cnrscope-lease-manage [191](#)
- dhcp-dns-update command (CLI) [248, 256](#)
 - enable [256](#)
 - update-dns-first [256](#)
 - update-dns-for-bootp [256](#)
 - set [248, 256](#)
 - backup-server-addr [256](#)
 - forward-zone-name [256](#)
 - reverse-zone-name [256](#)
 - reverse-zone-prefix-length [248](#)
 - server-addr [256](#)
 - synthesize-name [256](#)
 - synthetic-name-stem [256](#)
 - v6-synthetic-name-generator [248](#)
- dhcp-extension-add [352](#)
- dhcp-extension-attach [352](#)
- dhcp-extension-edit [352](#)
- dhcp-extension-list [351](#)
- dhcp-extension-point-list [352](#)
- dhcp-if-edit [16](#)
- dhcp-if-list [16](#)
- dhcp-ldap-adddhcp-ldap-edit [302](#)
- dhcp-ldap-list [301](#)
- dhcp-lease-history [225](#)
- dhcp-lease-time, DHCP option [417](#)
- dhcp-max-message-size, DHCP option [417](#)
- dhcp-message-type, DHCP option [417](#)
- dhcp-message, DHCP option [417](#)
- dhcp-network-tree [136](#)
- dhcp-ods-add [172](#)
- dhcp-ods-edit [172](#)
- dhcp-ods-list [172](#)
- dhcp-ods-pull-replica [182](#)
- dhcp-ods-pull-replica-report [182](#)
- dhcp-ods-pull-replica-run [182](#)
- dhcp-ods-push-data [182](#)
- dhcp-ods-push-data-report [182](#)
- dhcp-optdef-add [173](#)
- dhcp-optdef-edit [181](#)
- dhcp-optdef-list [173](#)
- dhcp-option-overload, DHCP option [417](#)
- dhcp-parameter-request-list, DHCP option [417](#)
- dhcp-pull-replica-report [99](#)
- dhcp-pull-replica-run [99](#)
- dhcp-pull-replica-select [99](#)
- dhcp-rebinding-time, DHCP option [417](#)
- dhcp-renewal-time, DHCP option [417](#)
- dhcp-requested-address, DHCP option [417](#)
- dhcp-server-identifier, DHCP option [417](#)
- dhcp-tcp-listener-add [237](#)
- dhcp-tcp-listener-edit [237](#)
- dhcp-tcp-listener-list [237](#)
- dhcp-user-class-id, DHCP option [428](#)
- DHCPDISCOVER packets [297](#)
- DHCLEASEQUERY packets [216](#)
 - See leasequery [216](#)
- DHCPRENEW packets [300](#)
- DHCPREQUEST packets [417](#)
- dhcserver-edit [17](#)
- DHCPv4 DNS update [266](#)
 - DHCID RR [266](#)
 - regress to TXT RR [266](#)
 - TXT RR [266](#)
- DHCPv6 [4, 24, 127, 162, 165, 179, 189, 190, 204, 246, 247, 249](#)
 - AAAA records, DNS update [246](#)
 - resource records [246](#)
 - address generation [127](#)
 - bindings [189](#)
 - client FQDN [249](#)
 - DHCID records, DNS update [246](#)
 - DNS update [246](#)
 - See DHCPv6 DNS update [246](#)
 - DNS update, upgrading [247](#)
 - lease affinity [190](#)
 - lease reservations [204](#)
 - leases [189](#)
 - links [4](#)
 - options [179](#)
 - policy hierarchy [165](#)
 - prefixes [4](#)
 - PTR records, DNS update [246](#)
 - reconfigure support [162](#)
 - server attributes [24](#)
 - max-client-leases [24](#)
 - v6-client-class-lookup-id [24](#)
- DHCPv6 DNS update [266](#)
 - DHCID RR [266](#)
- DHCPv6 failover [56](#)
- dhcpv6-lease-history [225](#)
- dhcpv6-network-tree [136](#)

- E**
- elapsed-time, DHCPv6 option [420](#)
 - end, DHCP option [407](#)
 - environment dictionary, extension point [374, 375](#)
 - data items [375](#)
 - environment-destroyer, extension point, DHCP [27, 390](#)
 - environmentdictionary, DHCP expression [322](#)
 - equal, DHCP expression [322](#)
 - equali, DHCP expression [322](#)
 - error, DHCP expression [322](#)
 - export command (CLI) [46, 181, 192](#)
 - addresses, VPN [46](#)
 - leases [46, 192](#)
 - vpn [46](#)
 - option-set [181](#)
 - expressions [146, 150, 151, 155, 315, 317, 349](#)
 - address range for scope templates [150](#)
 - creating [317](#)
 - debugging [349](#)
 - embedded policy for scope templates [151](#)
 - link templates [155](#)
 - prefix templates [151](#)
 - scope templates [146, 150](#)
 - scope name [150](#)
 - extension command (CLI) [352](#)
 - create [352](#)
 - list [352](#)
 - set [352](#)
 - init-entry [352](#)
 - extension points, DHCP [26](#)
 - extensions [26, 27, 61, 351, 352, 353, 354, 355, 356, 357, 359, 360, 362, 364, 365, 366, 367, 368, 370, 371, 373, 374, 376, 377, 378, 379, 380, 381, 382, 383, 385, 387, 388, 389, 390, 449, 461, 469, 482, 483, 486, 487, 488, 492](#)
 - API [482](#)
 - C/C++ [354, 359](#)
 - check-lease-acceptable, DHCP [387](#)
 - check-lease-acceptable, extension point, DHCP [27](#)
 - client-classes [365, 366](#)
 - modifying [365](#)
 - processing [366](#)
 - configuration errors [355](#)
 - creating [352](#)
 - deciding approaches [353](#)
 - decoded packet data items [449](#)
 - definition [351](#)
 - determining tasks [353](#)
 - DEX [492](#)
 - environment methods [492](#)
 - DEX attributes [487, 488](#)
 - dictionary [487](#)
 - methods [488](#)
 - DEX, dex [359](#)
 - dex.h file [360](#)
 - dhcp-parameter-request-list option [378](#)
 - extensions (*continued*)
 - DHCPv6, enabling [364](#)
 - dictionaries [355, 373, 449](#)
 - entries [449](#)
 - DNS requests, processing [371](#)
 - environment dictionary [355, 374](#)
 - environment-destroyer [390](#)
 - environment-destroyer, extension point, DHCP [27](#)
 - failover synchronization effect [61](#)
 - init-entry [359, 362](#)
 - C/C++, init-entry, extension point [362](#)
 - C/C++ [362](#)
 - Tcl, init-entry, extension point [359](#)
 - Tcl [359](#)
 - init-entry, extension point, DHCP [379](#)
 - initial-environment-dictionary, environment dictionary, extension point [376](#)
 - initial-environment-dictionary property [376](#)
 - lease-state-change, extension point, DHCP [388](#)
 - leases [367, 368, 371](#)
 - acceptability, determining [368](#)
 - finding [367](#)
 - requests, serializing [367](#)
 - state changes, tracing [371](#)
 - networks and links, determining [366](#)
 - packet data items, decoded [377](#)
 - packets [365](#)
 - decoding [365](#)
 - receiving [365](#)
 - post-class-lookup, extension point, DHCP [27, 382](#)
 - post-client-lookup, extension point, DHCP [27, 385](#)
 - post-packet-decode, extension point, DHCP [27, 381](#)
 - post-packet-encode, extension point, DHCP [27, 389](#)
 - post-send-packet, extension point, DHCP [27, 389](#)
 - pre-client-lookup, extension point, DHCP [27, 383](#)
 - pre-dns-add-forward, extension point, DHCP [27, 389](#)
 - pre-packet-decode, extension point, DHCP [27, 380](#)
 - pre-packet-encode, extension point, DHCP [27, 388](#)
 - recognizing [356](#)
 - request dictionary [355, 461](#)
 - request dictionary, extension points [376](#)
 - request processing [362](#)
 - response containers, building [366](#)
 - response dictionary [355, 469](#)
 - response dictionary, extension points [376](#)
 - response packets [370, 371](#)
 - encoding [371](#)
 - gathering data [370](#)
 - sending [371](#)
 - routine signature [354](#)
 - stable storage, updating [371](#)
 - Tcl [354, 357, 486](#)
 - environment methods [486](#)
 - Tcl attributes [482, 483](#)
 - dictionary [482](#)
 - methods [483](#)

extensions (*continued*)

thread-safe [360](#)

extensions-path, DHCP option [407](#)

F

failover pair-manage [61](#)

failover-pair command (CLI) [64, 67, 70, 89](#)

enable [64, 67](#)

load-balancing [64](#)

use-safe-period, failover-pair command (CLI) [67](#)

set [67](#)

safe-period [67](#)

set [64, 70, 89](#)

backup-pct [64](#)

dynamic-bootp-backup-pct [89](#)

load-balancing [70](#)

failover, DHCP [9, 55, 56, 57, 60, 61, 63, 64, 65, 67, 69, 70, 71, 73, 75, 84, 85, 86, 87, 89, 116](#)

address ranges, ensuring [63](#)

backup [64, 116](#)

allocation boundary [116](#)

failover-backup-allocation-boundary [116](#)

percentage [64](#)

backup percentage [64](#)

benefits [9](#)

BOOTP [63](#)

relay BOOTP [63](#)

changing roles [84](#)

checklist [63](#)

dynamic BOOTP, backup percentage, BOOTP [89](#)

dynamic [89](#)

PARTNER-DOWN state [89](#)

lazy updates [65](#)

lease period factor [65](#)

maximum client lead time, MCLT [65](#)

See maximum client lead time [65](#)

lease query [75](#)

load balancing [69, 70](#)

configuring [70](#)

local server synchronization [57](#)

logging [61](#)

main server, adding new [86](#)

monitoring failover, DHCP [87](#)

logging [87](#)

network failures [87](#)

operation [55, 86](#)

failover, DHCP [55](#)

types [55](#)

halting failover, DHCP [86](#)

backup server [86](#)

removing backup server [86](#)

removing failover, DHCP [86](#)

failover, DHCP (*continued*)

pairs [60, 63, 64](#)

backup percentage, scopes [64](#)

failover [64](#)

backup percentage [64](#)

creating, failover-pair command (CLI) [60](#)

create [60](#)

synchronizing, failover-pair command (CLI) [63](#)

sync [63](#)

regional cluster synchronization [61](#)

regional cluster synchronization, failover, DHCP [70](#)

confirming [70](#)

replacing servers with defective storage [85](#)

request/response buffer settings [69](#)

restrictions, communications interrupted [71](#)

safe period [67](#)

partner down state, PARTNER-DOWN state, failover [67](#)

enabling [67](#)

server pairs, creating [57](#)

simple scenarios [56](#)

state transitions [73](#)

states [71](#)

synchronize function [61](#)

troubleshooting [86](#)

finger-servers, DHCP option [411](#)

font-servers, DHCP option [411](#)

FQDN [11, 268](#)

DHCP processing [11](#)

option, DHCP [268](#)

G

gateway address, routers [12, 44](#)

gateway addresses, giaddr, DHCP field [12, 44](#)

generate-lease, extension point, DHCP [27](#)

geo-conf, DHCP option [428](#)

geoconf-civic, DHCP option [428](#)

geoconf-civic, DHCPv6 option [420](#)

grace period, leases [166](#)

H

host-name, DHCP option [407](#)

hosts [50, 86, 194, 245](#)

BOOTP [50](#)

configuring [50](#)

dynamic DNS update [245](#)

multiple interface, failover, DHCP [86](#)

pinging scopes [194](#)

pinging clients, leases [194](#)

pinging before allocating, ping hosts before offering [194](#)

ia-na, DHCPv6 option [420](#)
 ia-pd, DHCPv6 option [420](#)
 ia-ta, DHCPv6 option [420](#)
 iaaddr, DHCPv6 option [420](#)
 iaprefix, DHCPv6 option [420](#)
 ICMP [194](#)
 echo, See PING [194](#)
 ieee802.3-encapsulation, DHCP option [411](#)
 IETF [125](#)
 if, DHCP expression [322](#)
 import command (CLI) [46, 181, 192](#)
 leases [46, 192](#)
 option-set [181](#)
 impress-servers, DHCP option [407](#)
 info-refresh-time, DHCPv6 option [420](#)
 init-entry, extension point, DHCP [27](#)
 initial-url, DHCP option [428](#)
 interface cards [273](#)
 interface-id, DHCPv6 option [420](#)
 interface-mtu, DHCP option [410](#)
 Internet Control Message Protocol [194](#)
 See ICMP [194](#)
 Internet Engineering Task Force, IETF [1](#)
 Internet Operating System [43](#)
 IOS support, VPN support [43](#)
 See IOS, VPNs [43](#)
 IP helper address [53](#)
 IP history [223](#)
 See lease history reports [223](#)
 ip-forwarding, DHCP option [409](#)
 ip-helper [89](#)
 ip-string, DHCP expression [322](#)
 ip6-string, DHCP expression [322](#)
 iphist utility [226](#)
 lease history [226](#)
 IPv6 leases [186](#)
 states [186](#)
 irc-servers, DHCP option [411](#)
 is-string, DHCP expression [322](#)
 iSNS, DHCP option [417](#)

K

key command (CLI) [252](#)
 create [252](#)
 keys [251](#)
 key-edit [251](#)
 key-list [251](#)
 key-pull-replica [251](#)
 key-pull-replica-report [251](#)
 key-pull-replica-run [251](#)
 key-push-data [251](#)
 key-push-data-report [251](#)

keys (*continued*)

TSIG keys [251](#)
 creating [251](#)
 pulling [251](#)
 pushing [251](#)

L

LAN segments [11](#)
 LDAP [61, 63, 301, 302, 303, 305, 306, 308, 309, 311, 312, 313](#)
 client [302, 303](#)
 configuration [302](#)
 data use, enabling [303](#)
 configuring LDAP [301](#)
 protocol definition [301](#)
 connections [313](#)
 DHCP [303](#)
 client queries [303](#)
 mapping [303](#)
 directory support [303](#)
 distinguished name (dn) [303](#)
 embedded policies [305](#)
 entry creation [311](#)
 enabling [311](#)
 event service, failover synchronization effect [61](#)
 failover configuration [63](#)
 filtering searches [309](#)
 lease-state attributes [306](#)
 passwords [303](#)
 queries, enabling [303](#)
 query-timeout [313](#)
 schema checking, disabling [302](#)
 state updates [309](#)
 storing lease data [308](#)
 threadwaittime [313](#)
 timeout [313](#)
 troubleshooting [312](#)
 typical attribute settings [313](#)
 unprovisioning client entries [305](#)
 updates, enabling [309](#)
 ldap command (CLI) [290, 302, 303, 309, 311](#)
 create [302](#)
 delete [303](#)
 enable [290, 303, 309](#)
 can-query [290, 303](#)
 can-update [309](#)
 getEntry [303](#)
 list [303](#)
 listnames [303](#)
 set [303, 309, 311](#)
 create-object-classes [311](#)
 dn-attribute [311](#)
 dn-create-format [311](#)
 dn-format [311](#)
 preference [303](#)
 search-filter [303](#)

- ldap command (CLI) *(continued)*
 - set *(continued)*
 - search-path 303
 - search-scope 303
 - update-search-attribute 309
 - update-search-filter 309
 - update-search-path 309
 - update-search-scope 309
 - username 309
 - setEntry 303, 309, 311
 - create-dictionary givenname 311
 - create-dictionary localityname 311
 - create-dictionary sn 311
 - update-dictionary carlicense 309
 - update-dictionary uid 309
 - show 303
 - unsetEntry 303
- ldap-url, DHCP option 428
- lease 208
 - removing 208
- lease command (CLI) 52, 192, 195, 199, 212, 307
 - activate 195
 - deactivate 195
 - force-available 52, 212
 - list 199
 - macaddr 199
 - set 307
 - address 307
 - client-dns-name 307
 - client-domain-name 307
 - client-flags 307
 - client-host-name 307
 - client-id 307
 - client-mac-addr 307
 - expiration 307
 - flags 307
 - lease-renewal-time 307
 - start-time-of-state 307
 - state 307
 - vendor-class-identifier 307
 - show 192
- lease extensions, deferring 22
- lease history 106, 223, 224, 225, 228, 229
 - automatic trimming 229
 - age 229
 - interval 229
 - collecting 224
 - database directory 224
 - enabling 224
 - maximum age for trimming 229
 - querying 225
 - recording 106, 224
 - reports 223
 - trimming 228
- lease history reports 223
 - lease history 223
- Lease Notification, Dynamic 231
- lease-hist-detail 224
- lease-hist-list 224
- lease-hist-query 224
- lease-hist-view 224
- lease-notification command (CLI) 223, 229, 231
 - available 223
 - mail-host 229
 - recipients 229
 - scopes 229
 - specifying config file 231
- lease-state-change, extension point, DHCP 27
- lease6 command (CLI) 199
 - list 199
- leasequery 31, 216, 217, 218, 219
 - DHCPv4 pre-RFC implementation 217
 - DHCPv4 RFC 4388 implementation 218
 - DHCPv6 implementation 218
 - implementations 216
 - logging 31
 - reservations and 216
 - statistics 219
- leases 2, 3, 8, 50, 52, 166, 185, 186, 189, 190, 191, 192, 193, 195, 197, 211, 212, 214, 215, 216, 223, 229, 307, 308, 309
 - address usage reports 223
 - affinity 190
 - benefits 3
 - deactivating 52, 195
 - defined 2
 - DHCPv6 clients 189
 - DHCPv6 life cycle 190
 - excluding addresses from ranges 195
 - expired state 2
 - exporting 192
 - file 192
 - time format 192
 - forcing available 211
 - grace period 166
 - importing 192
 - inhibiting renewals 212
 - LDAP attributes 309
 - notification, receiving 229
 - orphaned 50
 - permanent 166, 186
 - querying, See leasequery 216
 - reacquiring 8
 - reactivating 195
 - releasing 8
 - renewal time, saving as state 307
 - renewing 2
 - reusing 52
 - scopes 2, 191, 192
 - listing 192
 - viewing 191
 - searching, filtering 197
 - state updates in LDAP 308

- leases (*continued*)
 - states [185, 307](#)
 - timeouts for unavailable [215](#)
 - times [186, 193](#)
 - guidelines [186](#)
 - import files [193](#)
 - overrides, allowing, policy command (CLI) [186](#)
 - enable [186](#)
 - allow-lease-time-override [186](#)
 - types [3](#)
 - unavailable [212, 214](#)
 - clearing [212](#)
 - handling [214](#)
 - utilization reports [229](#)
- length, DHCP expression [322](#)
- let, DHCP expression [322](#)
- link command (CLI) [135](#)
 - applyTemplate [135](#)
 - create [135](#)
 - template, template-root-prefix [135](#)
 - listPrefixes [135](#)
 - listPrefixNames [135](#)
- link template-pull-replica [145](#)
- link-template command (CLI) [145](#)
 - apply-to (link) [145](#)
 - create [145](#)
 - clone [145](#)
- links [4, 61](#)
 - DHCPv6 [4](#)
 - failover [61](#)
 - synchronization effect [61](#)
- linktemplate-add [144](#)
- linktemplate-edit [144](#)
- linktemplate-list [144](#)
- linktemplate-pull-replica-report [145](#)
- linktemplate-pull-replica-run [145](#)
- linktemplate-push-data [145](#)
- linktemplate-push-data-report [135](#)
- list [151, 155](#)
 - link template expression [155](#)
 - prefix template expression [151](#)
- log-servers, DHCP option [407](#)
- log, DHCP expression [322](#)
- lpr-servers, DHCP option [407](#)
- lq-client-links, DHCPv6 option [420](#)
- lq-query, DHCPv6 option [420](#)
- lq-relay-data, DHCPv6 option [420](#)
- lshift, DHCP expression [322](#)

M

- MAC addresses, clients [11](#)
- mask-blob, DHCP expression [322](#)
- mask-int, DHCP expression [322](#)
- mask-supplier, DHCP option [410](#)
- max-dgram-reassembly, DHCP option [409](#)

- maximum client lead time, failover [71](#)
- MCLT [71](#)
 - See maximum client lead time [71](#)
- mcns-security-server, DHCP option [428](#)
- merit-dump, DHCP option [407](#)
- mobile-ip-home-agents, DHCP option [428](#)
- multinetting [11](#)

N

- name-servers, DHCP option [407](#)
- name-service-search, DHCP option [428](#)
- nds-context, DHCP option [428](#)
- nds-servers, DHCP option [428](#)
- nds-tree, DHCP option [428](#)
- netbios-dd-servers, DHCP option [411](#)
- netbios-name-servers, DHCP option [411](#)
- netbios-node-type, DHCP option [411](#)
- netbios-scope, DHCP option [411](#)
- netinfo-parent-server-addr, DHCP option [428](#)
- netinfo-parent-server-tag, DHCP option [428](#)
- netwareip-domain, DHCP option [428](#)
- netwareip-information, DHCP option [428](#)
- Network Information Service [411](#)
 - See nis entries, nis-domain, DHCP option [411](#)
- Network News Transport Protocol, nntp-servers, DHCP option [411](#)
- Network Time Protocol, ntp-servers, DHCP option [411](#)
- network-tree [136](#)
- networks [136](#)
 - editing names [136](#)
 - listing [136](#)
 - managing [136](#)
- nis-domain-name, DHCPv6 option [420](#)
- nis-servers, DHCP option [411](#)
- nis-servers, DHCPv6 option [420](#)
- nis+-domain, DHCP option [411](#)
- nis+-servers, DHCP option [411](#)
- nisp-domain-name, DHCPv6 option [420](#)
- nisp-servers, DHCPv6 option [420](#)
- non-local-source-routing, DHCP option [409](#)
- NORMAL state, failover [71](#)
- not, DHCP expression [322](#)
- notification, lease [229](#)
- nslookup utility [282](#)
 - troubleshooting DNS update [282](#)
- null, DHCP expression [322](#)

O

- on-demand address pools [48](#)
 - See subnet allocation, DHCP address pools, on-demand [48](#)
- option command (CLI) [172, 180](#)
 - get [172](#)
 - listtypes [180](#)
 - show [172](#)

- option command (CLI) *(continued)*
 - unset [172](#)
 - option sets [182](#)
 - local [182](#)
 - pulling [182](#)
 - pushing, regional clusters [182](#)
 - option definition sets [182](#)
 - option-set command (CLI) [172](#)
 - show [172](#)
 - options [61, 164, 173, 179, 180](#)
 - custom DHCP [173](#)
 - data types, listing [180](#)
 - DHCPv6 [179](#)
 - setting [179](#)
 - failover synchronization effect [61](#)
 - policy hierarchy [164](#)
 - OptionSetPXE.txt file, OptionSetJumpStart.txt file [181](#)
 - or, DHCP expression [322](#)
 - Organizationally Unique Identifier [45](#)
 - See OUI, VPNs [45](#)
 - oro, DHCPv6 option [420](#)
- ## P
- pad, DHCP option [407](#)
 - path-mtu-aging-timeout, DHCP option [409](#)
 - path-mtu-plateau-tables, DHCP option [409](#)
 - PAUSED state, failover [71](#)
 - perform-mask-discovery, DHCP option [410](#)
 - pick-first-value, DHCP expression [322](#)
 - pie-chart [395](#)
 - policies [3, 11, 61, 120, 121, 161, 162, 163, 164, 168, 170, 186, 270](#)
 - cloning [168](#)
 - compared with scopes [3](#)
 - configuring [161](#)
 - DHCP [168, 170](#)
 - options [168, 170](#)
 - DHCPv6 [162](#)
 - dual zone updates, allowing, policy command (CLI) [270](#)
 - enable [270](#)
 - allow-dual-zone-dns-update [270](#)
 - embedded [121, 163, 170](#)
 - editing [170](#)
 - scopes [121](#)
 - vendor options [121](#)
 - failover synchronization effect [61](#)
 - hierarchy [164](#)
 - lease time overrides, allowing [186](#)
 - named scopes [163](#)
 - named policies [163](#)
 - options [11, 168](#)
 - adding [168](#)
 - scope command (CLI) [120](#)
 - set [120](#)
 - policy [120](#)
 - scopes, See scopes [11, 120](#)
 - policies *(continued)*
 - suboptions [170](#)
 - adding [170](#)
 - policies, DHCP [49](#)
 - address block [49](#)
 - policy command (CLI) [52, 168, 169, 268, 270, 294, 315](#)
 - create [168](#)
 - clone [168](#)
 - disable [268](#)
 - allow-client-a-record-update [268](#)
 - enable [168, 268, 270](#)
 - allow-client-a-record-update [268](#)
 - allow-dual-zone-dns-update [270](#)
 - permanent-leases [168](#)
 - getOption [168, 169](#)
 - dhcp-lease-time [168](#)
 - listOptions [168](#)
 - set [168, 294, 315](#)
 - grace-period [294](#)
 - limitation-count [315](#)
 - setLeaseTime [168](#)
 - setOption [52, 168, 169](#)
 - subnet-mask [168](#)
 - unsetOption [169](#)
 - policy-filters, DHCP option [409](#)
 - Post Office Protocol, pop3-servers, DHCP option [411](#)
 - post-class-lookup, extension point, DHCP [27, 382](#)
 - post-client-lookup, extension point, DHCP [27, 385](#)
 - post-packet-decode, extension point, DHCP [27, 381](#)
 - post-packet-encode, extension point, DHCP [27, 389](#)
 - post-send-packet, extension point, DHCP [27, 389](#)
 - POTENTIAL-CONFLICT state, failover [71](#)
 - pre-client-lookup, extension point, DHCP [27, 383](#)
 - pre-dns-add-forward, extension point, DHCP [27, 389](#)
 - pre-packet-decode, extension point, DHCP [27, 380](#)
 - pre-packet-encode, extension point, DHCP [27, 388](#)
 - preference, DHCPv6 option [420](#)
 - prefix allocation groups [130](#)
 - prefix command (CLI) [134, 248, 287](#)
 - addReservation [134](#)
 - applyTemplate [134](#)
 - create [134](#)
 - template [134](#)
 - createReverseZone [248](#)
 - deleteReverseZone [248](#)
 - listLeases [134](#)
 - set [287](#)
 - selection-tags [287](#)
 - prefix stability [128, 129](#)
 - CMTS [129](#)
 - universal [129](#)
 - prefix-template command (CLI) [144, 287](#)
 - apply-to (prefix) [144](#)
 - create [144](#)
 - clone [144](#)

- prefix-template command (CLI) *(continued)*
 - set **287**
 - selection-tags **287**
 - prefixes **4, 61, 127, 130, 134**
 - configuring **130**
 - count on server, getting **134**
 - dhcp command (CLI) **134**
 - getPrefixCount **134**
 - DHCPv6 **4**
 - failover **61**
 - synchronization effect **61**
 - interface-identifier, assigning **127**
 - prefixtemplate-add **141**
 - prefixtemplate-edit **141**
 - prefixtemplate-list **141**
 - prefixtemplate-pull-replica **143**
 - prefixtemplate-pull-replica-report **143**
 - prefixtemplate-pull-replica-run **143**
 - prefixtemplate-push-data **143**
 - prefixtemplate-push-data-report **143**
 - progn, DHCP expression **322**
 - push-subnet **100**
 - pushing, routers **100**
 - subnets, pushing **100**
 - PXE clients, importing option sets **181**
 - pxe-client-arch, DHCP option **428**
 - pxe-client-machine-id, DHCP option **428**
 - pxe-client-network-id, DHCP option **428**
- ## R
- rapid-commit, DHCPv6 option **420**
 - reclaim-subnet **97**
 - reconfigure-accept, DHCPv6 option **420**
 - reconfigure-message, DHCPv6 option **420**
 - RECOVER state, failover **71**
 - RECOVER-DONE state, failover **71**
 - regex, DHCP expression **322**
 - regional clusters **100**
 - subnets to local clusters **100**
 - subnets to routers **100**
 - pushing **100**
 - regional-addr-admin-role-add **92**
 - regional-addr-admin-role-edit **92**
 - relay-agent-info, DHCP option **428**
 - relay-agent-subscriber-id, DHCPv6 option **420**
 - relay-message, DHCPv6 option **420**
 - remote-id, DHCPv6 option **420**
 - report command (CLI) **223**
 - reports **105, 223**
 - address usage **223**
 - lease history **223**
 - subnet utilization **105**
 - request dump, DHCP expression **322**
 - request option, DHCP expression **322**
 - request packetfield, DHCP expression **322**
 - requestdictionary, DHCP expression **322**
 - reservations, lease **202, 209, 210, 216**
 - creating **202**
 - LEASEQUERY **216**
 - removing, reservation command (CLI) **209**
 - delete **209**
 - using client ID, client ID **210**
 - reservations, lease, policy command (CLI) **210**
 - enable **210**
 - use-client-id-for-reservations **210**
 - resource records **246**
 - DHCID **246**
 - resource-location-servers, DHCP option **407**
 - response dump, DHCP expression **322**
 - response option, DHCP expression **322**
 - response packetfield, DHCP expression **322**
 - responsedictionary, DHCP expression **322**
 - restricting lease dates **187**
 - return-last, DHCP expression **322**
 - reverse zones **248**
 - prefixes, creating from prefixes **248**
 - DHCPv6 **248**
 - RFCs **45, 51, 69, 127, 189, 218, 246, 271, 297, 315, 407, 409, 410, 411, 417, 420, 428, 449**
 - 1001 **411**
 - 1002 **411**
 - 1042 **411**
 - 1179 **407**
 - 1191 **409**
 - 1256 **410**
 - 1497 **407**
 - 2131 **51**
 - 2685 **45**
 - 2782 **271**
 - 3041 **189**
 - 3046 **297, 315**
 - 3074 **69**
 - 3315 **420**
 - 3319 **420**
 - 3633 **189**
 - 3898 **420**
 - 4075 **420**
 - 4174 **417**
 - 4242 **420**
 - 4280 **417, 420**
 - 4291 **127**
 - 4388 **218, 417**
 - 4580 **420**
 - 4649 **420**
 - 4701 **246**
 - 4702 **246**
 - 4704 **246**
 - 4833 **428, 449**
 - 5192 **449**
 - 865 **407**
 - 868 **407**

RFCs (*continued*)

887 [407](#)
 893 [411](#)
 894 [411](#)
 950 [407](#)

roles [91](#)

address block administrator [91](#)

root-path, DHCP option [407](#)

round-robin [11, 113](#)

scope selection [11, 113](#)

router-discovery, DHCP option [410](#)

router-solicitation-address, DHCP option [410](#)

routers [53, 168](#)

Cisco routers [53](#)

subnet [168](#)

routers, DHCP option [407](#)

S

safe period, failover [71](#)

scope command (CLI) [52, 64, 89, 119, 120, 122, 123, 124, 125, 192, 209, 212, 214, 287](#)

clearUnavailable [212](#)

create [120](#)

delete [125](#)

disable [52, 89, 124, 214](#)

dhcp [52, 89, 124](#)

ping-clients [214](#)

enable [52, 89, 123, 124, 214](#)

deactivated [124](#)

dhcp [124](#)

dynamic-bootp [52, 89](#)

ignore-declines [214](#)

renew-only [123](#)

update-dns-for-bootp [52](#)

enable bootp [123](#)

listLeases [192](#)

listnames [119](#)

removeReservation [52, 209](#)

set [64, 124, 287](#)

backup-pet [64](#)

free-address-config [124](#)

selection-tag-list [287](#)

unset [122](#)

primary-subnet [122](#)

scope templates [140, 141, 150, 151](#)

address range, expressions [150](#)

cloning, scope-template command (CLI) [141](#)

create [141](#)

clone [141](#)

creating, scope-template command (CLI) [140](#)

create [140](#)

editing, scope-template command (CLI) [140](#)

set [140](#)

embedded policy expressions [151](#)

scope name expressions [150](#)

scope-add-st-data [112](#)

scope-list [112](#)

scope-policy command (CLI) [121](#)

disable [121](#)

enable [121](#)

set [121](#)

setVendorOptions [121](#)

show [121](#)

unset [121](#)

unsetVendorOptions [121](#)

scope-template command (CLI) [150, 287](#)

set [150, 287](#)

options-exp [150](#)

ranges-exp [150](#)

scope-name [150](#)

selection-tag-list [287](#)

scope-template-add [139](#)

scope-template-list [139](#)

scopes [45, 61, 111, 112, 113, 114, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 191, 196, 208, 211, 212](#)

address allocation [113](#)

address ranges [112](#)

allocate-first-available [114, 116](#)

allocation-priority [114, 116](#)

attributes [119, 122](#)

bootp [122](#)

disabling, scope command (CLI) [119](#)

disable [119](#)

dynamic bootp [122](#)

enabling, scope command (CLI) [119](#)

enable [119](#)

getting, scope command (CLI) [119](#)

get [119](#)

listing, scope command (CLI) [119](#)

list [119](#)

primary-subnet [122](#)

setting, scope command (CLI) [119](#)

set [119](#)

showing, scope command (CLI) [119](#)

show [119](#)

count on server, getting [120](#)

deactivating [124](#)

defining [111](#)

dhcp command (CLI) [120](#)

getScopeCount [120](#)

dhcp edit mode [119](#)

synchronous, staged [119](#)

editing [118](#)

failover [61](#)

synchronization effect [61](#)

failover-backup-allocation-boundary [117](#)

forcing lease availability [211](#)

inhibiting lease renewals [212](#)

internal address allocation [117](#)

leases, See leases [191](#)

- scopes (*continued*)
 - listing ranges [196](#)
 - scope command (CLI) [196](#)
 - listRanges [196](#)
 - moving/decommissioning BOOTP clients [122](#)
 - multiple [113](#)
 - names [112](#)
 - network addresses [112](#)
 - policies, See policies [112](#)
 - primary subnets [122](#)
 - ranges [120](#)
 - adding, scope command (CLI) [120](#)
 - addRange [120](#)
 - removing [125](#)
 - if not reusing addresses [125](#)
 - if reusing addresses [125](#)
 - reusing addresses [125](#)
 - removing ranges [196](#)
 - scope command (CLI) [196](#)
 - removeRange [196](#)
 - renew-only [123](#)
 - scope command (CLI) [122](#)
 - set [122](#)
 - primary-subnet [122](#)
 - secondary subnets [121](#)
 - multiple logical, secondary [121](#)
 - staged edits, reporting, scope command (CLI) [120](#)
 - report-staged-edits [120](#)
 - traps, SNMP, free address [123](#)
 - unreserving leases, reservations [208](#)
 - VPNs [45](#)
- search, DHCP expression [322](#)
- secondary [121](#)
 - subnets [121](#)
- selection tags [290](#)
 - appending dhcp-user-class-id [290](#)
 - appending RADIUS class [290](#)
 - appending RADIUS pool [290](#)
 - mapping RADIUS class [290](#)
 - mapping RADIUS pool name [290](#)
 - mapping user class identifier [290](#)
- server-identifier, DHCPv6 option [420](#)
- server-unicast, DHCPv6 option [420](#)
- session command (CLI) [45, 120](#)
 - get [120](#)
 - dhcp-edit-mode [120](#)
 - set [45, 120](#)
 - current-vpn [45](#)
 - dhcp-edit-mode [120](#)
- setq, DHCP expression [322](#)
- setting [173](#)
 - custom DHCP options [173](#)
- SHUTDOWN state, failover [71](#)
- Simple Mail Transport Protocol [411](#)
 - See SNMP, smtp-servers, DHCP option [411](#)
- simulating, top-of-zone, A records [273](#)
 - dns command (CLI) [273](#)
 - enable [273](#)
 - simulate-zone-top-dynupdate [273](#)
- sip-servers-address, DHCPv6 option [420](#)
- sip-servers, DHCP option [428](#)
- slp-directory-agent, DHCP option [428](#)
- slp-service-scope, DHCP option [428](#)
- SNMP [87](#)
 - traps [87](#)
 - failover mismatch [87](#)
 - server not responding [87](#)
- sntp-servers, DHCPv6 option [420](#)
- SOA records [271](#)
- SRV records [271, 273](#)
 - enabling viewing [273](#)
- stacked-area-chart [395](#)
- stacked-bar-chart [395](#)
- starts-with, DHCP expression [322](#)
- STARTUP state, failover [71](#)
- static [16, 88, 410](#)
 - addresses [16](#)
 - BOOTP [88](#)
 - routes [410](#)
- static-routes, DHCP option [428](#)
- statistics [219](#)
 - leasequery [219](#)
- status-code, DHCPv6 option [420](#)
- streettalk-directory-assistance-servers, DHCP option [411](#)
- streettalk-servers, DHCP option [411](#)
- subnet allocation, DHCP [49](#)
 - configuring [49](#)
- subnet utilization [105, 106, 107, 108](#)
 - collect-addr-util-duration [106](#)
 - collect-addr-util-interval [106](#)
 - data, collecting [106](#)
 - querying [106](#)
 - reports [105](#)
 - trim-subnet-util-age [107](#)
 - trim-subnet-util-interval [107](#)
 - viewing data [108](#)
- subnet-alloc, DHCP option [428](#)
- subnet-create-revzones [97](#)
- subnet-edit [99](#)
- subnet-mask, DHCP option [407](#)
- subnet-selection, DHCP option [428](#)
- subnet-util-list [106](#)
- subnet-util-query [106](#)
- subnet-util-view [106](#)
- subnets [11, 93, 97, 100, 102, 121, 410](#)
 - client-accessible [11](#)
 - defined [93](#)
 - joining [121](#)
 - local, all-subnets-local, DHCP option [410](#)
 - pushing to local clusters [100](#)
 - reclaiming [97](#)

- subnets (*continued*)
 - viewing address blocks [102](#)
- subnets, address blocks [92](#)
- subnets, DHCP [49](#)
 - address block [49](#)
 - allocation request [49](#)
 - increment [49](#)
 - initial [49](#)
- subscriber limitation, using option 82 [297, 301](#)
 - troubleshooting [301](#)
- substring, DHCP expression [322](#)
- swap-server, DHCP option [407](#)
- synchronization, failover [61](#)
 - Complete operation [61](#)
 - Exact operation [61](#)
 - operations, failover, DHCP [61](#)
 - synchronizing [61](#)
 - Update operation [61](#)
- synthesize-host-name, DHCP expression [322](#)

T

- Tcl [26, 27, 354, 357, 358](#)
 - API [358](#)
 - extensions [26, 27, 354, 357](#)
- TCP Default TTL [411](#)
- tcp-keepalive-garbage, DHCP option [411](#)
- tcp-keepalive-interval, DHCP option [411](#)
- templates [139](#)
 - scope, scope templates [139](#)
 - managing [139](#)
- tftp-server, DHCP option [417](#)
- time-offset, DHCP option [407](#)
- time-servers, DHCP option [407](#)
- timeouts for unavailable leases [215](#)
- to-blob, DHCP expression [322](#)
- to-ip, DHCP expression [322](#)
- to-ip6, DHCP expression [322](#)
- to-lower, DHCP expression [322](#)
- to-sint, DHCP expression [322](#)
- to-string, DHCP expression [322](#)
- to-uint, DHCP expression [322](#)
- trailer-encapsulation, DHCP option [411](#)
- translate, DHCP expression [322](#)
- traps [124](#)
 - addr-trap command (CLI) [124](#)
 - set [124](#)
 - high-threshold [124](#)
 - low-threshold [124](#)
- traps, SNMP [61, 87, 124](#)
 - creating, addr-trap command (CLI) [124](#)
 - create [124](#)
 - failover synchronization effect [61](#)
 - low and high address thresholds [124](#)
- trimming [107](#)
 - immediate subnet utilization [107](#)

- trimming (*continued*)
 - subnet utilization [107](#)
 - trimming data [107](#)
 - subnet utilization records [107](#)
- try, DHCP expression [322](#)
- TSIG keys [61, 252, 253](#)
 - DNS update configuration attributes [253](#)
 - failover synchronization effect [61](#)
 - importing, import command (CLI) [252](#)
 - keys [252](#)
 - rules for secrets [253](#)
- TTL property [163](#)
 - default [163](#)
- tz-database, DHCP option [428](#)
- tz-posix, DHCP option [428](#)

U

- unified address space [100](#)
- update-policy (CLI command) [259, 261](#)
 - create [259](#)
 - rules [261](#)
 - add [261](#)
 - remove [261](#)
- user-auth, DHCP option [428](#)
- user-class, DHCPv6 option [420](#)
- username, ldap command (CLI) [303](#)
 - set [303](#)
 - password [303](#)
- users [8](#)
 - lease availability [8](#)
- utility programs [125, 194](#)
 - ipconfig, ipconfig utility [125](#)
 - ping, ping utility [194](#)
- utilization-detail [101](#)
- utilization-tree [101](#)

V

- v-i-vendor-class, DHCP option [428](#)
- v-i-vendor-info, DHCP option [428](#)
- v6-dhcp-pull-replica-report [99](#)
- v6-dhcp-pull-replica-run [99](#)
- v6-dhcp-pull-replica-select [99](#)
- v6-utilization-detail [103](#)
- v6-utilization-tree [103](#)
- v6addr-dhcp-search [198](#)
- v6addr-dhcp-util-tree [108](#)
- v6addr-lease-history [225](#)
- v6address-space [103](#)
- v6address-space-tree [103](#)
- validate-host-name, DHCP expression [322](#)
- vendor-class, DHCPv6 option [420](#)
- vendor-encapsulated-options, DHCP option [428](#)
- vendor-opts, DHCPv6 option [420](#)

- virtual channel identifier [348](#)
- virtual path identifier [348](#)
- virtual private networks [5, 45](#)
 - See VPNs [5, 45](#)
- virtual routing and forwarding table ID [43, 45](#)
 - See VRFs [43](#)
 - See VRFs, VPNs [45](#)
- vpn command (CLI) [45](#)
 - create [45](#)
 - set [45](#)
 - vrf-name [45](#)
- vpn-edit [45](#)
- vpn-id, DHCP option [428](#)
- vpn-list [43](#)
- VPNs [43, 45, 46, 50, 61, 120](#)
 - creating [45](#)
 - current [46](#)
 - failover synchronization effect [61](#)
 - identifier [45](#)
 - leases, importing [46](#)
 - orphaned leases [50](#)
 - setting current session command (CLI) [120](#)
 - set [120](#)
 - current-vpn [120](#)

W

- Windows [294](#)
 - ipconfig utility, winipcfg utility [294](#)
- Windows 2000 [273](#)
 - DNS update [273](#)
 - domain controllers [273](#)

- Windows 2000 (*continued*)
 - environments [273](#)
- Windows SMS [25](#)
 - See SMS integrating with DHCP server [25](#)
 - See SMS System Management Server [25](#)
- www-servers, DHCP option [411](#)

X

- x-display-managers, DHCP option [411](#)

Z

- zone (CLI command) [262, 264](#)
 - listRR [264](#)
 - dns [264](#)
 - set [262](#)
 - update-policy-list [262](#)
- zone command (CLI) [264, 266](#)
 - getScavengeStartTime [266](#)
 - set [264](#)
 - log-settings [264](#)
 - scvg-ignore-restart-interval [264](#)
 - scvg-interval [264](#)
 - scvg-no-refresh-interval [264](#)
 - scvg-refresh-interval [264](#)
- zones [262, 264, 265](#)
 - applying update policies [262](#)
 - scavenging [264, 265](#)
 - start time, getting [265](#)

