Kea Administrator Reference Manual Documentation

Release 1.9.0

Internet Systems Consortium

Sep 28, 2020

CONTENTS

1 Introduction 3
- 1.1 Supported platforms . . . 3
  - 1.1.1 Regularly tested platforms . . . 3
  - 1.1.2 Best effort . . . 3
  - 1.1.3 Community maintained . . . 4
  - 1.1.4 Unsupported platforms . . . 4
- 1.2 Required Software at Run-Time . . . 4
- 1.3 Kea Software . . . 5

2 Quick Start 7
- 2.1 Quick Start Guide for using tarball . . . 7
- 2.2 Quick Start Guide using native packages . . . 7
- 2.3 Quick Start Guide for DHCPv4 and DHCPv6 Services . . . 8
- 2.4 Running the Kea Servers Directly . . . 9

3 Installation 11
- 3.1 Packages . . . 11
- 3.2 Installation Hierarchy . . . 11
- 3.3 Build Requirements . . . 11
- 3.4 Installation from Source . . . 12
  - 3.4.1 Download Tar File . . . 13
  - 3.4.2 Retrieve from Git . . . 13
  - 3.4.3 Configure Before the Build . . . 13
  - 3.4.4 Build . . . 15
  - 3.4.5 Install . . . 15
  - 3.4.6 Cross-Building . . . 16
- 3.5 DHCP Database Installation and Configuration . . . 16
  - 3.5.1 Building with MySQL Support . . . 16
  - 3.5.2 Building with PostgreSQL support . . . 16
  - 3.5.3 Building with CQL (Cassandra) Support . . . 17
- 3.6 Hammer Building Tool . . . 17
- 3.7 Running Kea from non-root account on Linux . . . 19

4 Kea Database Administration 21
- 4.1 Databases and Database Version Numbers . . . 21
- 4.2 The kea-admin Tool . . . 21
- 4.3 Supported Backends . . . 22
  - 4.3.1 Memfile . . . 22
    - Upgrading Memfile Lease Files from an Earlier Version of Kea . . . 22
  - 4.3.2 MySQL . . . 23
    - First-Time Creation of the MySQL Database . . . 23

Upgrading a MySQL Database from an Earlier Version of Kea 24
Simple MySQL tweak to gain performance 25
4.3.3 PostgreSQL 25
First-Time Creation of the PostgreSQL Database 25
Initialize the PostgreSQL Database Using kea-admin 27
Upgrading a PostgreSQL Database from an Earlier Version of Kea 27
4.3.4 Cassandra 27
First-Time Creation of the Cassandra Database 27
Upgrading a Cassandra Database from an Earlier Version of Kea 28
4.3.5 Using Read-Only Databases with Host Reservations 28
4.3.6 Limitations Related to the Use of SQL Databases 29
Year 2038 Issue 29

5 Kea Configuration **31**
5.1 JSON Configuration 31
5.1.1 JSON Syntax 31
5.1.2 Comments and User Context 32
5.1.3 Simplified Notation 33
5.2 Kea Configuration Backend 34
5.2.1 Applicability 34
5.2.2 CB Capabilities and Limitations 34
5.2.3 CB Components 35
5.2.4 Configuration Sharing and Server Tags 36

6 Managing Kea with keactrl **39**
6.1 Overview 39
6.2 Command Line Options 39
6.3 The keactrl Configuration File 39
6.4 Commands 41
6.5 Overriding the Server Selection 43

7 The Kea Control Agent **45**
7.1 Overview of the Kea Control Agent 45
7.2 Configuration 45
7.3 Secure Connections 47
7.4 Starting the Control Agent 49
7.5 Connecting to the Control Agent 49

8 The DHCPv4 Server **51**
8.1 Starting and Stopping the DHCPv4 Server 51
8.2 DHCPv4 Server Configuration 52
8.2.1 Introduction 52
8.2.2 Lease Storage 54
Memfile - Basic Storage for Leases 54
Lease Database Configuration 56
Cassandra-Specific Parameters 57
8.2.3 Hosts Storage 58
DHCPv4 Hosts Database Configuration 59
Using Read-Only Databases for Host Reservations with DHCPv4 60
8.2.4 Interface Configuration 60
8.2.5 Issues with Unicast Responses to DHCPINFORM 63
8.2.6 IPv4 Subnet Identifier 63
8.2.7 IPv4 Subnet Prefix 64
8.2.8 Configuration of IPv4 Address Pools 64
8.2.9 Sending T1 (Option 58) and T2 (Option 59) 66

8.2.10 Standard DHCPv4 Options . . . 66
8.2.11 Custom DHCPv4 Options . . . 74
8.2.12 DHCPv4 Private Options . . . 76
8.2.13 DHCPv4 Vendor-Specific Options . . . 79
8.2.14 Nested DHCPv4 Options (Custom Option Spaces) . . . 81
8.2.15 Unspecified Parameters for DHCPv4 Option Configuration . . . 83
8.2.16 Stateless Configuration of DHCPv4 Clients . . . 83
8.2.17 Client Classification in DHCPv4 . . . 84
Setting Fixed Fields in Classification . . . 85
Using Vendor Class Information in Classification . . . 86
Defining and Using Custom Classes . . . 86
Required Classification . . . 87
8.2.18 DDNS for DHCPv4 . . . 87
DHCP-DDNS Server Connectivity . . . 89
When Does the kea-dhcp4 Server Generate a DDNS Request? . . . 90
kea-dhcp4 Name Generation for DDNS Update Requests . . . 91
Sanitizing Client Host Name and FQDN Names . . . 93
8.2.19 Next Server (siaddr) . . . 94
8.2.20 Echoing Client-ID (RFC 6842) . . . 94
8.2.21 Using Client Identifier and Hardware Address . . . 95
8.2.22 Authoritative DHCPv4 Server Behavior . . . 96
8.2.23 DHCPv4-over-DHCPv6: DHCPv4 Side . . . 97
8.2.24 Sanity Checks in DHCPv4 . . . 98
8.2.25 Storing Extended Lease Information . . . 98
8.2.26 Multi-threading settings . . . 99
8.2.27 Multi-threading settings in different backends . . . 99
8.2.28 IPv6-only Preferred Networks . . . 100
8.3 Host Reservation in DHCPv4 . . . 101
8.3.1 Address Reservation Types . . . 102
8.3.2 Conflicts in DHCPv4 Reservations . . . 102
8.3.3 Reserving a Hostname . . . 103
8.3.4 Including Specific DHCPv4 Options in Reservations . . . 104
8.3.5 Reserving Next Server, Server Hostname, and Boot File Name . . . 105
8.3.6 Reserving Client Classes in DHCPv4 . . . 105
8.3.7 Storing Host Reservations in MySQL, PostgreSQL, or Cassandra . . . 107
8.3.8 Fine-Tuning DHCPv4 Host Reservation . . . 107
8.3.9 Global Reservations in DHCPv4 . . . 109
8.3.10 Pool Selection with Client Class Reservations . . . 110
8.3.11 Subnet Selection with Client Class Reservations . . . 111
8.4 Shared Networks in DHCPv4 . . . 112
8.4.1 Local and Relayed Traffic in Shared Networks . . . 114
8.4.2 Client Classification in Shared Networks . . . 116
8.4.3 Host Reservations in Shared Networks . . . 118
8.5 Server Identifier in DHCPv4 . . . 119
8.6 How the DHCPv4 Server Selects a Subnet for the Client . . . 120
8.6.1 Using a Specific Relay Agent for a Subnet . . . 120
8.6.2 Segregating IPv4 Clients in a Cable Network . . . 121
8.7 Duplicate Addresses (DHCPDECLINE Support) . . . 121
8.8 Statistics in the DHCPv4 Server . . . 122
8.9 Management API for the DHCPv4 Server . . . 125
8.10 User Contexts in IPv4 . . . 127
8.11 Supported DHCP Standards . . . 127
8.12 DHCPv4 Server Limitations . . . 128
8.13 Kea DHCPv4 Server Examples . . . 128

8.14 Configuration Backend in DHCPv4 . . . 128
8.14.1 Supported Parameters . . . 129
8.14.2 Enabling Configuration Backend . . . 130

9 The DHCPv6 Server **133**
9.1 Starting and Stopping the DHCPv6 Server . . . 133
9.2 DHCPv6 Server Configuration . . . 134
9.2.1 Introduction . . . 134
9.2.2 Lease Storage . . . 136
Memfile - Basic Storage for Leases . . . 136
Lease Database Configuration . . . 137
Cassandra-Specific Parameters . . . 139
9.2.3 Hosts Storage . . . 139
DHCPv6 Hosts Database Configuration . . . 140
Using Read-Only Databases for Host Reservations with DHCPv6 . . . 141
9.2.4 Interface Configuration . . . 141
9.2.5 IPv6 Subnet Identifier . . . 142
9.2.6 IPv6 Subnet Prefix . . . 143
9.2.7 Unicast Traffic Support . . . 143
9.2.8 Configuration of IPv6 Address Pools . . . 144
9.2.9 Subnet and Prefix Delegation Pools . . . 145
9.2.10 Prefix Exclude Option . . . 146
9.2.11 Standard DHCPv6 Options . . . 146
9.2.12 Common Softwire46 Options . . . 152
Softwire46 Container Options . . . 152
S46 Rule Option . . . 153
S46 BR Option . . . 153
S46 DMR Option . . . 153
S46 IPv4/IPv6 Address Binding Option . . . 154
S46 Port Parameters . . . 154
9.2.13 Custom DHCPv6 Options . . . 154
9.2.14 DHCPv6 Vendor-Specific Options . . . 156
9.2.15 Nested DHCPv6 Options (Custom Option Spaces) . . . 157
9.2.16 Unspecified Parameters for DHCPv6 Option Configuration . . . 159
9.2.17 Controlling the Values Sent for T1 and T2 Times . . . 159
9.2.18 IPv6 Subnet Selection . . . 160
9.2.19 Rapid Commit . . . 161
9.2.20 DHCPv6 Relays . . . 161
9.2.21 Relay-Supplied Options . . . 162
9.2.22 Client Classification in DHCPv6 . . . 163
Defining and Using Custom Classes . . . 163
Required Classification . . . 164
9.2.23 DDNS for DHCPv6 . . . 165
DHCP-DDNS Server Connectivity . . . 167
When Does the kea-dhcp6 Server Generate a DDNS Request? . . . 168
kea-dhcp6 Name Generation for DDNS Update Requests . . . 169
Sanitizing Client FQDN Names . . . 170
9.2.24 DHCPv4-over-DHCPv6: DHCPv6 Side . . . 172
9.2.25 Sanity Checks in DHCPv6 . . . 173
9.2.26 Storing Extended Lease Information . . . 174
9.2.27 Multi-threading settings . . . 175
9.2.28 Multi-threading settings in different backends . . . 175
9.3 Host Reservation in DHCPv6 . . . 176
9.3.1 Address/Prefix Reservation Types . . . 177

9.3.2 Conflicts in DHCPv6 Reservations 177
9.3.3 Reserving a Hostname 178
9.3.4 Including Specific DHCPv6 Options in Reservations 179
9.3.5 Reserving Client Classes in DHCPv6 180
9.3.6 Storing Host Reservations in MySQL, PostgreSQL, or Cassandra 181
9.3.7 Fine-Tuning DHCPv6 Host Reservation 181
9.3.8 Global Reservations in DHCPv6 183
9.3.9 Pool Selection with Client Class Reservations 184
9.3.10 Subnet Selection with Client Class Reservations 185
9.4 Shared Networks in DHCPv6 186
9.4.1 Local and Relayed Traffic in Shared Networks 189
9.4.2 Client Classification in Shared Networks 191
9.4.3 Host Reservations in Shared Networks 193
9.5 Server Identifier in DHCPv6 194
9.6 DHCPv6 data directory 197
9.7 Stateless DHCPv6 (Information-Request Message) 197
9.8 Support for RFC 7550 (now part of RFC 8415) 197
9.9 Using a Specific Relay Agent for a Subnet 198
9.10 Segregating IPv6 Clients in a Cable Network 199
9.11 MAC/Hardware Addresses in DHCPv6 200
9.12 Duplicate Addresses (DECLINE Support) 201
9.13 Statistics in the DHCPv6 Server 202
9.14 Management API for the DHCPv6 Server 205
9.15 User Contexts in IPv6 207
9.16 Supported DHCPv6 Standards 208
9.17 DHCPv6 Server Limitations 209
9.18 Kea DHCPv6 server examples 209
9.19 Configuration Backend in DHCPv6 209
9.19.1 Supported Parameters 209
9.19.2 Enabling Configuration Backend 210

10 Database Connectivity **213**

11 Lease Expiration **215**
11.1 Lease Reclamation 215
11.2 Lease Reclamation Configuration Parameters 216
11.3 Configuring Lease Reclamation 216
11.4 Configuring Lease Affinity 218
11.5 Reclaiming Expired Leases via Command 219

12 Congestion Handling **221**
12.1 What is Congestion? 221
12.2 Configuring Congestion Handling 221

13 The DHCP-DDNS Server **223**
13.1 Overview 223
13.1.1 DNS Server Selection 223
13.1.2 Conflict Resolution 223
13.1.3 Dual-Stack Environments 224
13.2 Starting and Stopping the DHCP-DDNS Server 224
13.3 Configuring the DHCP-DDNS Server 225
13.3.1 Global Server Parameters 225
13.3.2 Management API for the D2 Server 226
13.3.3 TSIG Key List 227
13.3.4 Forward DDNS 228

Adding Forward DDNS Domains . . . 228
Adding Forward DNS Servers . . . 229
13.3.5 Reverse DDNS . . . 230
Adding Reverse DDNS Domains . . . 230
Adding Reverse DNS Servers . . . 231
13.3.6 User Contexts in DDNS . . . 232
13.3.7 Example DHCP-DDNS Server Configuration . . . 232
13.4 DHCP-DDNS Server Limitations . . . 234
13.5 Supported Standards . . . 234

14 The LFC Process **235**
14.1 Overview . . . 235
14.2 Command-Line Options . . . 235

15 Client Classification **237**
15.1 Client Classification Overview . . . 237
15.2 Built-in Client Classes . . . 238
15.3 Using Expressions in Classification . . . 239
15.3.1 Logical operators . . . 241
15.3.2 Substring . . . 242
15.3.3 Concat . . . 242
15.3.4 Ifelse . . . 242
15.3.5 Hexstring . . . 242
15.4 Configuring Classes . . . 242
15.5 Using Static Host Reservations In Classification . . . 244
15.6 Configuring Subnets With Class Information . . . 244
15.7 Configuring Pools With Class Information . . . 245
15.8 Using Classes . . . 247
15.9 Classes and Hooks . . . 247
15.10 Debugging Expressions . . . 247

16 Hooks Libraries **249**
16.1 Introduction . . . 249
16.2 Installing Hook Packages . . . 249
16.3 Configuring Hooks Libraries . . . 251
16.4 Available Hooks Libraries . . . 252
16.5 user_chk: Checking User Access . . . 255
16.6 legal_log: Forensic Logging Hooks . . . 256
16.6.1 Log File Naming . . . 256
16.6.2 DHCPv4 Log Entries . . . 256
16.6.3 DHCPv6 Log Entries . . . 258
16.6.4 Configuring the Forensic Log Hooks . . . 259
16.6.5 Database Backend . . . 261
16.7 flex_id: Flexible Identifiers for Host Reservations . . . 262
16.8 flex_option Flexible Option for Option value settings . . . 265
16.9 host_cmds: Host Commands . . . 266
16.9.1 The subnet-id Parameter . . . 267
16.9.2 The reservation-add Command . . . 267
16.9.3 The reservation-get Command . . . 268
16.9.4 The reservation-get-all Command . . . 269
16.9.5 The reservation-get-page command . . . 270
16.9.6 The reservation-get-by-hostname Command . . . 272
16.9.7 The reservation-get-by-id Command . . . 273
16.9.8 The reservation-del Command . . . 274

16.10 lease_cmds: Lease Commands . . . 275
16.10.1 The lease4-add, lease6-add Commands . . . 277
16.10.2 The lease6-bulk-apply Command . . . 279
16.10.3 The lease4-get, lease6-get Commands . . . 280
16.10.4 The lease4-get-all, lease6-get-all Commands . . . 282
16.10.5 The lease4-get-page, lease6-get-page Commands . . . 283
16.10.6 The lease4-get-by-*, lease6-get-by-* Commands . . . 285
16.10.7 The lease4-del, lease6-del Commands . . . 285
16.10.8 The lease4-update, lease6-update Commands . . . 286
16.10.9 The lease4-wipe, lease6-wipe Commands . . . 287
16.10.10The lease4-resend-ddns, lease6-resend-ddns Commands . . . 287
16.11 subnet_cmds: Subnet Commands . . . 288
16.11.1 The subnet4-list Command . . . 289
16.11.2 The subnet6-list Command . . . 289
16.11.3 The subnet4-get Command . . . 290
16.11.4 The subnet6-get Command . . . 291
16.11.5 The subnet4-add Command . . . 291
16.11.6 The subnet6-add Command . . . 292
16.11.7 The subnet4-update Command . . . 293
16.11.8 The subnet6-update Command . . . 294
16.11.9 The subnet4-del Command . . . 294
16.11.10The subnet6-del Command . . . 295
16.11.11The network4-list, network6-list Commands . . . 296
16.11.12The network4-get, network6-get Commands . . . 296
16.11.13The network4-add, network6-add Commands . . . 297
16.11.14The network4-del, network6-del Commands . . . 298
16.11.15The network4-subnet-add, network6-subnet-add Commands . . . 299
16.11.16The network4-subnet-del, network6-subnet-del Commands . . . 300
16.12 BOOTP support . . . 300
16.12.1 BOOTP Hooks Limitation . . . 301
16.13 class_cmds: Class Commands . . . 302
16.13.1 The class-add Command . . . 302
16.13.2 The class-update Command . . . 302
16.13.3 The class-del Command . . . 303
16.13.4 The class-list Command . . . 304
16.13.5 The class-get Command . . . 304
16.14 cb_cmds: Configuration Backend Commands . . . 305
16.14.1 Commands Structure . . . 305
16.14.2 Control Commands for DHCP Servers . . . 306
16.14.3 Metadata . . . 306
16.14.4 remote-server4-del, remote-server6-del commands . . . 307
16.14.5 remote-server4-get, remote-server6-get commands . . . 308
16.14.6 remote-server4-get-all, remote-server6-get-all commands . . . 308
16.14.7 remote-server4-set, remote-server6-set commands . . . 309
16.14.8 The remote-global-parameter4-del, remote-global-parameter6-del Commands . . . 310
16.14.9 The remote-global-parameter4-get, remote-global-parameter6-get Commands . . . 310
16.14.10The remote-global-parameter4-get-all, remote-global-parameter6-get-all Commands . . . 312
16.14.11The remote-global-parameter4-set, remote-global-parameter6-set Commands . . . 313
16.14.12The remote-network4-del, remote-network6-del Commands . . . 313
16.14.13The remote-network4-get, remote-network6-get Commands . . . 314
16.14.14The remote-network4-list, remote-network6-list Commands . . . 314
16.14.15The remote-network4-set, remote-network6-set Commands . . . 316
16.14.16The remote-option-def4-del, remote-option-def6-del Commands . . . 317
16.14.17The remote-option-def4-get, remote-option-def6-get Commands . . . 317

16.14.18 The remote-option-def4-get-all, remote-option-def6-get-all Commands . . . 318
16.14.19 The remote-option-def4-set, remote-option-def6-set Commands . . . 318
16.14.20 The remote-option4-global-del, remote-option6-global-del Commands . . . 319
16.14.21 The remote-option4-global-get, remote-option6-global-get Commands . . . 319
16.14.22 The remote-option4-global-get-all, remote-option6-global-get-all Commands . . . 320
16.14.23 The remote-option4-global-set, remote-option6-global-set Commands . . . 321
16.14.24 The remote-option4-network-del, remote-option6-network-del Commands . . . 321
16.14.25 The remote-option4-network-set, remote-option6-network-set Commands . . . 322
16.14.26 The remote-option6-pd-pool-del Command . . . 323
16.14.27 The remote-option6-pd-pool-set Command . . . 323
16.14.28 The remote-option4-pool-del, remote-option6-pool-del Commands . . . 324
16.14.29 The remote-option4-pool-set, remote-option6-pool-set Commands . . . 324
16.14.30 The remote-option4-subnet-del, remote-option6-subnet-del Commands . . . 325
16.14.31 The remote-option4-subnet-set, remote-option6-subnet-set Commands . . . 326
16.14.32 The remote-subnet4-del-by-id, remote-subnet6-del-by-id Commands . . . 326
16.14.33 The remote-subnet4-del-by-prefix, remote-subnet6-del-by-prefix Commands . . . 327
16.14.34 The remote-subnet4-get-by-id, remote-subnet6-get-by-id Commands . . . 327
16.14.35 The remote-subnet4-get-by-prefix, remote-subnet6-get-by-prefix Commands . . . 327
16.14.36 The remote-subnet4-list, remote-subnet6-list Commands . . . 328
16.14.37 The remote-subnet4-set, remote-subnet6-set Commands . . . 329
16.15 ha: High Availability . . . 331
  16.15.1 Supported Configurations . . . 331
  16.15.2 Clocks on Active Servers . . . 332
  16.15.3 Server States . . . 333
  16.15.4 Scope Transition in a Partner-Down Case . . . 335
  16.15.5 Load-Balancing Configuration . . . 336
  16.15.6 Load Balancing with Advanced Classification . . . 338
  16.15.7 Hot-Standby Configuration . . . 340
  16.15.8 Passive-Backup Configuration . . . 341
  16.15.9 Lease Information Sharing . . . 342
  16.15.10 Controlling Lease-Page Size Limit . . . 343
  16.15.11 Timeouts . . . 344
  16.15.12 Pausing the HA State Machine . . . 345
  16.15.13 Control Agent Configuration . . . 347
  16.15.14 Controlled Shutdown and Maintenance of DHCP servers . . . 348
  16.15.15 Upgrading from Older HA Versions . . . 349
  16.15.16 Control Commands for High Availability . . . 349
    The ha-sync Command . . . 349
    The ha-scopes Command . . . 350
    The ha-continue Command . . . 350
    The ha-heartbeat Command . . . 350
    The status-get Command . . . 351
    The ha-maintenance-start Command . . . 353
    The ha-maintenance-cancel Command . . . 354
    The ha-maintenance-notify Command . . . 354
16.16 stat_cmds: Supplemental Statistics Commands . . . 354
  16.16.1 The stat-lease4-get, stat-lease6-get Commands . . . 355
16.17 radius: RADIUS Server Support . . . 357
  16.17.1 Compilation and Installation of the RADIUS Hook . . . 358
  16.17.2 RADIUS Hook Configuration . . . 362
16.18 host_cache: Caching Host Reservations . . . 365
  16.18.1 The cache-flush Command . . . 366
  16.18.2 The cache-clear Command . . . 366
  16.18.3 The cache-size Command . . . 366

16.18.4 The cache-write Command . . . 366
16.18.5 The cache-load Command . . . 367
16.18.6 The cache-get Command . . . 367
16.18.7 The cache-get-by-id Command . . . 367
16.18.8 The cache-insert Command . . . 368
16.18.9 The cache-remove Command . . . 369
16.19 lease_query: Leasequery . . . 369
16.19.1 DHCPv4 Leasequery . . . 369
16.19.2 DHCPv4 Leasequery Configuration . . . 371
16.19.3 DHCPv6 Leasequery . . . 371
16.19.4 DHCPv6 Leasequery Configuration . . . 373
16.20 User Contexts in Hooks . . . 373

17 Statistics 375
17.1 Statistics Overview . . . 375
17.2 Statistics Lifecycle . . . 375
17.3 Commands for Manipulating Statistics . . . 376
17.3.1 The statistic-get Command . . . 376
17.3.2 The statistic-reset Command . . . 377
17.3.3 The statistic-remove Command . . . 377
17.3.4 The statistic-get-all Command . . . 377
17.3.5 The statistic-reset-all Command . . . 378
17.3.6 The statistic-remove-all Command . . . 378
17.3.7 The statistic-sample-age-set Command . . . 378
17.3.8 The statistic-sample-age-set-all Command . . . 379
17.3.9 The statistic-sample-count-set Command . . . 379
17.3.10 The statistic-sample-count-set-all Command . . . 379
17.4 Time series . . . 380

18 Management API 381
18.1 Data Syntax . . . 381
18.2 Using the Control Channel . . . 383
18.3 Commands Supported by Both the DHCPv4 and DHCPv6 Servers . . . 384
18.3.1 The build-report Command . . . 384
18.3.2 The config-get Command . . . 384
18.3.3 The config-reload Command . . . 384
18.3.4 The config-test Command . . . 384
18.3.5 The config-write Command . . . 385
18.3.6 The leases-reclaim Command . . . 385
18.3.7 The libreload Command . . . 386
18.3.8 The list-commands Command . . . 386
18.3.9 The config-set Command . . . 386
18.3.10 The shutdown Command . . . 387
18.3.11 The dhcp-disable Command . . . 388
18.3.12 The dhcp-enable Command . . . 388
18.3.13 The status-get Command . . . 388
18.3.14 The server-tag-get Command: . . . 389
18.3.15 The config-backend-pull Command: . . . 389
18.3.16 The version-get Command . . . 389
18.4 Commands Supported by the D2 Server . . . 389
18.5 Commands Supported by the Control Agent . . . 390

19 Logging 391
19.1 Logging Configuration . . . 391

19.1.1 Loggers . . . 391
The name (string) Logger . . . 391
The severity (string) Logger . . . 395
The debuglevel (integer) Logger . . . 395
The output_options (list) Logger . . . 395
The output (string) Option . . . 395
The flush (true of false) Option . . . 395
The maxsize (integer) Option . . . 396
The maxver (integer) Option . . . 396
The pattern (string) Option . . . 396
19.1.2 Logging Message Format . . . 396
Example Logger Configurations . . . 398
19.1.3 Logging During Kea Startup . . . 398

20 The Kea Shell **401**
20.1 Overview of the Kea Shell . . . 401
20.2 Shell Usage . . . 401

21 YANG/NETCONF Support **403**
21.1 Overview . . . 403
21.2 Installing NETCONF . . . 403
21.2.1 Installing NETCONF on Ubuntu 18.04 . . . 403
21.2.2 Installing NETCONF on CentOS 7.5 . . . 403
21.3 Quick Sysrepo Overview . . . 403
21.4 Supported YANG Models . . . 405
21.5 Using the NETCONF Agent . . . 406
21.6 Configuration . . . 406
21.7 A kea-netconf Configuration Example . . . 408
21.8 Starting and Stopping the NETCONF Agent . . . 410
21.9 A Step-by-Step NETCONF Agent Operation Example . . . 410
21.9.1 Setup of NETCONF Agent Operation Example . . . 410
21.9.2 Error Handling in NETCONF Operation Example . . . 412
21.9.3 NETCONF Operation Example with Two Pools . . . 414
21.9.4 NETCONF Operation Example with Two Subnets . . . 414
21.9.5 NETCONF Operation Example with Logging . . . 415

22 Monitoring Kea with Stork **417**
22.1 Kea statistics in Grafana . . . 417

23 API Reference **419**
23.1 build-report . . . 420
23.2 cache-clear . . . 421
23.3 cache-flush . . . 421
23.4 cache-get . . . 422
23.5 cache-get-by-id . . . 422
23.6 cache-insert . . . 423
23.7 cache-load . . . 424
23.8 cache-remove . . . 425
23.9 cache-size . . . 426
23.10 cache-write . . . 426
23.11 class-add . . . 427
23.12 class-del . . . 428
23.13 class-get . . . 428
23.14 class-list . . . 429
23.15 class-update . . . 430

23.16 config-backend-pull . . . 430
23.17 config-get . . . 431
23.18 config-reload . . . 432
23.19 config-set . . . 432
23.20 config-test . . . 433
23.21 config-write . . . 434
23.22 dhcp-disable . . . 434
23.23 dhcp-enable . . . 435
23.24 ha-continue . . . 435
23.25 ha-heartbeat . . . 436
23.26 ha-maintenance-cancel . . . 437
23.27 ha-maintenance-notify . . . 437
23.28 ha-maintenance-start . . . 438
23.29 ha-scopes . . . 438
23.30 ha-sync . . . 439
23.31 lease4-add . . . 440
23.32 lease4-del . . . 441
23.33 lease4-get . . . 441
23.34 lease4-get-all . . . 442
23.35 lease4-get-by-client-id . . . 443
23.36 lease4-get-by-hostname . . . 444
23.37 lease4-get-by-hw-address . . . 445
23.38 lease4-get-page . . . 446
23.39 lease4-resend-ddns . . . 447
23.40 lease4-update . . . 447
23.41 lease4-wipe . . . 448
23.42 lease6-add . . . 449
23.43 lease6-bulk-apply . . . 449
23.44 lease6-del . . . 451
23.45 lease6-get . . . 451
23.46 lease6-get-all . . . 452
23.47 lease6-get-by-duid . . . 454
23.48 lease6-get-by-hostname . . . 454
23.49 lease6-get-page . . . 455
23.50 lease6-resend-ddns . . . 457
23.51 lease6-update . . . 457
23.52 lease6-wipe . . . 458
23.53 leases-reclaim . . . 459
23.54 libreload . . . 459
23.55 list-commands . . . 460
23.56 network4-add . . . 460
23.57 network4-del . . . 462
23.58 network4-get . . . 462
23.59 network4-list . . . 463
23.60 network4-subnet-add . . . 464
23.61 network4-subnet-del . . . 465
23.62 network6-add . . . 465
23.63 network6-del . . . 467
23.64 network6-get . . . 467
23.65 network6-list . . . 468
23.66 network6-subnet-add . . . 469
23.67 network6-subnet-del . . . 470
23.68 remote-global-parameter4-del . . . 471
23.69 remote-global-parameter4-get . . . 471

23.70 remote-global-parameter4-get-all . . . 472
23.71 remote-global-parameter4-set . . . 473
23.72 remote-global-parameter6-del . . . 474
23.73 remote-global-parameter6-get . . . 475
23.74 remote-global-parameter6-get-all . . . 476
23.75 remote-global-parameter6-set . . . 477
23.76 remote-network4-del . . . 478
23.77 remote-network4-get . . . 479
23.78 remote-network4-list . . . 480
23.79 remote-network4-set . . . 481
23.80 remote-network6-del . . . 482
23.81 remote-network6-get . . . 483
23.82 remote-network6-list . . . 484
23.83 remote-network6-set . . . 485
23.84 remote-option-def4-del . . . 486
23.85 remote-option-def4-get . . . 486
23.86 remote-option-def4-get-all . . . 487
23.87 remote-option-def4-set . . . 488
23.88 remote-option-def6-del . . . 489
23.89 remote-option-def6-get . . . 490
23.90 remote-option-def6-get-all . . . 491
23.91 remote-option-def6-set . . . 492
23.92 remote-option4-global-del . . . 493
23.93 remote-option4-global-get . . . 494
23.94 remote-option4-global-get-all . . . 495
23.95 remote-option4-global-set . . . 496
23.96 remote-option4-network-del . . . 497
23.97 remote-option4-network-set . . . 498
23.98 remote-option4-pool-del . . . 499
23.99 remote-option4-pool-set . . . 500
23.100 remote-option4-subnet-del . . . 501
23.101 remote-option4-subnet-set . . . 502
23.102 remote-option6-global-del . . . 503
23.103 remote-option6-global-get . . . 504
23.104 remote-option6-global-get-all . . . 505
23.105 remote-option6-global-set . . . 506
23.106 remote-option6-network-del . . . 507
23.107 remote-option6-network-set . . . 508
23.108 remote-option6-pd-pool-del . . . 509
23.109 remote-option6-pd-pool-set . . . 510
23.110 remote-option6-pool-del . . . 511
23.111 remote-option6-pool-set . . . 512
23.112 remote-option6-subnet-del . . . 513
23.113 remote-option6-subnet-set . . . 514
23.114 remote-server4-del . . . 515
23.115 remote-server4-get . . . 515
23.116 remote-server4-get-all . . . 516
23.117 remote-server4-set . . . 517
23.118 remote-server6-del . . . 518
23.119 remote-server6-get . . . 519
23.120 remote-server6-get-all . . . 520
23.121 remote-server6-set . . . 521
23.122 remote-subnet4-del-by-id . . . 522
23.123 remote-subnet4-del-by-prefix . . . 523

23.124 remote-subnet4-get-by-id . . . 523
23.125 remote-subnet4-get-by-prefix . . . 524
23.126 remote-subnet4-list . . . 525
23.127 remote-subnet4-set . . . 526
23.128 remote-subnet6-del-by-id . . . 527
23.129 remote-subnet6-del-by-prefix . . . 528
23.130 remote-subnet6-get-by-id . . . 529
23.131 remote-subnet6-get-by-prefix . . . 530
23.132 remote-subnet6-list . . . 531
23.133 remote-subnet6-set . . . 532
23.134 reservation-add . . . 533
23.135 reservation-del . . . 534
23.136 reservation-get . . . 535
23.137 reservation-get-all . . . 536
23.138 reservation-get-by-hostname . . . 536
23.139 reservation-get-by-id . . . 537
23.140 reservation-get-page . . . 537
23.141 server-tag-get . . . 538
23.142 shutdown . . . 539
23.143 stat-lease4-get . . . 540
23.144 stat-lease6-get . . . 540
23.145 statistic-get . . . 541
23.146 statistic-get-all . . . 542
23.147 statistic-remove . . . 543
23.148 statistic-remove-all . . . 544
23.149 statistic-reset . . . 544
23.150 statistic-reset-all . . . 545
23.151 statistic-sample-age-set . . . 546
23.152 statistic-sample-age-set-all . . . 546
23.153 statistic-sample-count-set . . . 547
23.154 statistic-sample-count-set-all . . . 548
23.155 status-get . . . 549
23.156 subnet4-add . . . 550
23.157 subnet4-del . . . 550
23.158 subnet4-get . . . 551
23.159 subnet4-list . . . 552
23.160 subnet4-update . . . 553
23.161 subnet6-add . . . 554
23.162 subnet6-del . . . 554
23.163 subnet6-get . . . 555
23.164 subnet6-list . . . 556
23.165 subnet6-update . . . 557
23.166 version-get . . . 558

24 Manual Pages **559**
24.1 kea-dhcp4 - DHCPv4 server in Kea . . . 559
24.1.1 Synopsis . . . 559
24.1.2 Description . . . 559
24.1.3 Arguments . . . 559
24.1.4 Documentation . . . 559
24.1.5 Mailing Lists and Support . . . 560
24.1.6 History . . . 560
24.1.7 See Also . . . 560
24.2 kea-dhcp6 - DHCPv6 server in Kea . . . 560

24.2.1 Synopsis . . . 560
24.2.2 Description . . . 560
24.2.3 Arguments . . . 560
24.2.4 Documentation . . . 561
24.2.5 Mailing Lists and Support . . . 561
24.2.6 History . . . 561
24.2.7 See Also . . . 561
24.3 kea-ctrl-agent - Control Agent process in Kea . . . 561
24.3.1 Synopsis . . . 561
24.3.2 Description . . . 562
24.3.3 Arguments . . . 562
24.3.4 Documentation . . . 562
24.3.5 Mailing Lists and Support . . . 562
24.3.6 History . . . 562
24.3.7 See Also . . . 563
24.4 keactrl - Shell script for managing Kea . . . 563
24.4.1 Synopsis . . . 563
24.4.2 Description . . . 563
24.4.3 Configuration File . . . 563
24.4.4 Options . . . 563
24.4.5 Documentation . . . 564
24.4.6 Mailing Lists and Support . . . 564
24.4.7 See Also . . . 564
24.5 kea-admin - Shell script for managing Kea databases . . . 564
24.5.1 Synopsis . . . 564
24.5.2 Description . . . 564
24.5.3 Arguments . . . 564
24.5.4 Documentation . . . 565
24.5.5 Mailing Lists and Support . . . 565
24.5.6 See Also . . . 565
24.6 kea-dhcp-ddns - DHCP-DDNS process in Kea . . . 566
24.6.1 Synopsis . . . 566
24.6.2 Description . . . 566
24.6.3 Arguments . . . 566
24.6.4 Documentation . . . 566
24.6.5 Mailing Lists and Support . . . 566
24.6.6 History . . . 567
24.6.7 See Also . . . 567
24.7 kea-lfc - Lease File Cleanup process in Kea . . . 567
24.7.1 Synopsis . . . 567
24.7.2 Description . . . 567
24.7.3 Arguments . . . 567
24.7.4 Documentation . . . 568
24.7.5 Mailing Lists and Support . . . 568
24.7.6 History . . . 568
24.7.7 See Also . . . 568
24.8 kea-shell - Text client for Control Agent process . . . 568
24.8.1 Synopsis . . . 568
24.8.2 Description . . . 569
24.8.3 Arguments . . . 569
24.8.4 Documentation . . . 569
24.8.5 Mailing Lists and Support . . . 569
24.8.6 History . . . 570
24.8.7 See Also . . . 570

24.9 kea-netconf - NETCONF agent for Kea environment . . . 570
- 24.9.1 Synopsis . . . 570
- 24.9.2 Description . . . 570
- 24.9.3 Arguments . . . 570
- 24.9.4 Documentation . . . 570
- 24.9.5 Mailing Lists and Support . . . 571
- 24.9.6 History . . . 571
- 24.9.7 See Also . . . 571

24.10 perfdhcp - DHCP benchmarking tool . . . 571
- 24.10.1 Synopsis . . . 571
- 24.10.2 Description . . . 571
- 24.10.3 Templates . . . 572
- 24.10.4 Options . . . 573
- 24.10.5 DHCPv4-Only Options . . . 574
- 24.10.6 DHCPv6-Only Options . . . 575
- 24.10.7 Template-Related Options . . . 575
- 24.10.8 Options Controlling a Test . . . 575
- 24.10.9 Arguments . . . 576
- 24.10.10 Errors . . . 576
- 24.10.11 Exit Status . . . 576
- 24.10.12 Mailing Lists and Support . . . 576
- 24.10.13 History . . . 576
- 24.10.14 See Also . . . 576

25 Kea Messages Manual **577**
- 25.1 ALLOC . . . 577
- 25.2 ASIODNS . . . 585
- 25.3 BOOTP . . . 588
- 25.4 COMMAND . . . 589
- 25.5 CTRL . . . 591
- 25.6 DATABASE . . . 592
- 25.7 DCTL . . . 594
- 25.8 DHCP4 . . . 597
- 25.9 DHCP6 . . . 615
- 25.10 DHCPSRV . . . 633
- 25.11 DHCP . . . 660
- 25.12 EVAL . . . 670
- 25.13 FLEX . . . 674
- 25.14 HA . . . 674
- 25.15 HOOKS . . . 684
- 25.16 HOSTS . . . 688
- 25.17 HTTP . . . 694
- 25.18 LEASE . . . 697
- 25.19 LFC . . . 698
- 25.20 LOGIMPL . . . 699
- 25.21 LOG . . . 700
- 25.22 MYSQL . . . 702
- 25.23 NETCONF . . . 716
- 25.24 STAT . . . 719
- 25.25 USER . . . 720

26 Kea Flow Diagrams **723**
- 26.1 Main Loop . . . 723
- 26.2 DHCPv4 Packet Processing . . . 723

26.3 DHCP Request Processing 723
26.4 DHCPv4 Subnet Selection 727
26.5 DHCPv4 Special Case of Double-booting 727
26.6 DHCPv4 Allocate Lease 727
26.7 Lease States 727
26.8 Checking for Host Reservations 727
26.9 Building the Options List 727

27 Acknowledgments **739**

28 Indices and tables **741**

Kea is an open source implementation of the Dynamic Host Configuration Protocol (DHCP) servers, developed and maintained by Internet Systems Consortium (ISC).

This is the reference guide for Kea version 1.9.0. Links to the most up-to-date version of this document (in PDF, HTML, and plain text formats), along with other documents for Kea, can be found in ISC's Knowledgebase.

CHAPTER

ONE

INTRODUCTION

Kea is the next generation of DHCP software developed by ISC. It supports both DHCPv4 and DHCPv6 protocols along with their extensions, e.g. prefix delegation and dynamic updates to DNS.

This guide covers Kea version 1.9.0.

For information about supported platforms see *Supported platforms*.

1.1 Supported platforms

In general, this version of Kea will build and run on any POSIX-compliant system with a C++ compiler (with C++11 support), the Botan cryptographic library, the log4cplus logging library and the Boost system library.

The Kea build has been checked with GCC g++ 4.8.5 and some later versions, and Clang 800.0.38 and some later versions.

ISC regularly tests Kea on many operating systems and architectures, but lacks the resources to test all of them. Consequently, ISC is only able to offer support on a “best effort” basis for some.

1.1.1 Regularly tested platforms

As of January, 2020, Kea is officially supported on CentOS, Fedora, Ubuntu, Debian, and FreeBSD systems. Kea-1.9.0 builds have been tested on:

- CentOS Linux — 7, 8
- Fedora — 31, 32
- Ubuntu — 16.04, 18.04, 20.04
- Debian GNU/Linux — 9, 10

There are currently no plans to port Kea to Windows systems.

1.1.2 Best effort

The following are platforms on which Kea is known to build and run. ISC makes every effort to fix bugs on these platforms, but may be unable to do so quickly due to lack of hardware, less familiarity on the part of engineering staff, and other constraints.

- FreeBSD — 11.3, 12.0
- Alpine Linux — 3.10, 3.11
- macOS — 10.13, 10.14

1.1.3 Community maintained

These systems may not all have the required dependencies for building Kea easily available, although it will be possible in many cases to compile those directly from source. The community and interested parties may wish to help with maintenance, and we welcome patch contributions, although we cannot guarantee that we will accept them. All contributions will be assessed against the risk of adverse effect on officially supported platforms.

Platforms past or close to their respective EOL dates, such as:

- Ubuntu 14.04, 18.10, 19.04, 19.10
- Fedora 30
- CentOS 6
- Debian 8 (Jessie)
- FreeBSD 10.x

1.1.4 Unsupported platforms

These are platforms on which Kea 1.7+ is known *not* to build or run:

- Windows (all versions)
- Windows Server (all versions)
- Any platform with OpenSSL 1.0.1 or earlier, which does not also have Botan as an alternative
- Any platform with log4cplus version 1.0.2 or earlier.

1.2 Required Software at Run-Time

Running Kea uses various extra software packages which may not be provided in the default installation of some operating systems, nor in the standard package collections. You may need to install this required software separately. (For the build requirements, also see *Build Requirements*.)

- Kea supports two cryptographic libraries: Botan and OpenSSL. Only one of them is required to be installed during compilation. Kea uses the Botan library for C++ (https://botan.randombit.net/), version 2.0 or later. Note that support for Botan versions earlier than 2.0 was removed in Kea 1.7.0 and later. As an alternative to Botan, Kea can use the OpenSSL cryptographic library (https://www.openssl.org/), version 1.0.2 or later.
- Kea uses the log4cplus C++ logging library (https://sourceforge.net/p/log4cplus/wiki/Home/). It requires log4cplus version 1.0.3 or later.
- Kea requires the Boost system library (https://www.boost.org/). Building with the header-only version of Boost is no longer recommended.

Some optional features of Kea have additional dependencies.

- To store lease information in a MySQL database, Kea requires MySQL headers and libraries. This is an optional dependency; Kea can be built without MySQL support.
- To store lease information in a PostgreSQL database, Kea requires PostgreSQL headers and libraries. This is an optional dependency; Kea can be built without PostgreSQL support.
- To store lease information in a Cassandra database (CQL), Kea requires Cassandra headers and libraries. This is an optional dependency; Kea can be built without Cassandra support.

- Integration with RADIUS is provided in Kea via the hooks library available to our paid support customers. Use of this library requires the FreeRadius-client library to be present on the system where Kea is running. This is an optional dependency; Kea can be built without RADIUS support.
- Kea provides a NETCONF interface with the kea-netconf agent. This Kea module is built optionally and requires Sysrepo software when used. Building Kea with NETCONF support requires many dependencies to be installed, which are described in more detail in *Installing NETCONF*.

1.3 Kea Software

Kea is modular. Part of this modularity is accomplished using multiple cooperating processes which, together, provide the server functionality. The following software is included with Kea:

- `keactrl` — This tool starts, stops, reconfigures, and reports status for the Kea servers.
- `kea-dhcp4` — The DHCPv4 server process. This process responds to DHCPv4 queries from clients.
- `kea-dhcp6` — The DHCPv6 server process. This process responds to DHCPv6 queries from clients.
- `kea-dhcp-ddns` — The DHCP Dynamic DNS process. This process acts as an intermediary between the DHCP servers and DNS servers. It receives name update requests from the DHCP servers and sends DNS update messages to the DNS servers.
- `kea-admin` — A useful tool for database backend maintenance (creating a new database, checking versions, upgrading, etc.).
- `kea-lfc` — This process removes redundant information from the files used to provide persistent storage for the memfile database backend. While it can be run standalone, it is normally run as and when required by the Kea DHCP servers.
- `kea-ctrl-agent` — Kea Control Agent (CA) is a daemon that exposes a RESTful control interface for managing Kea servers.
- `kea-netconf` - kea-netconf is an agent that provides a YANG/NETCONF interface for the Kea environment.
- `kea-shell` — This simple text client uses the REST interface to connect to the Kea Control Agent.
- `perfdhcp` — A DHCP benchmarking tool which simulates multiple clients to test both DHCPv4 and DHCPv6 server performance.

The tools and modules are covered in full detail in this guide. In addition, manual pages are also provided in the default installation.

Kea also provides C++ libraries and programmer interfaces for DHCP. These include detailed developer documentation and code examples.

CHAPTER

TWO

QUICK START

This section describes the basic steps needed to get Kea up and running. For further details, full customizations, and troubleshooting, see the respective chapters elsewhere in this Kea Administrator Reference Manual (ARM).

2.1 Quick Start Guide for using tarball

1. Install required run-time and build dependencies. See *Build Requirements* for details.
2. Download the Kea source tarball from the ISC.org downloads page or the ISC downloads.isc.org.
3. Extract the tarball. For example:

   ```
   $ tar xvzf kea-1.9.0.tar.gz
   ```

4. Go into the source directory and run the configure script:

   ```
   $ cd kea-1.9.0
   $ ./configure [your extra parameters]
   ```

5. Build it:

   ```
   $ make
   ```

6. Install it (by default it will be placed in `/usr/local/`, so it is likely that you will need root privileges for this step):

   ```
   $ make install
   ```

2.2 Quick Start Guide using native packages

Starting with Kea 1.6.0, ISC now provides native RPM, deb and APK packages, which make Kea installation much easier. Unless you want to tweak specific compilation options, it is usually easier to install Kea using native packages.

1. Go to Kea on cloudsmith.io and choose Kea version and enter repository.
2. Use `Set Me Up` and follow instructions to add repository on your system.
3. Update system repositories. For example:

   ```
   $ apt-get update
   ```

4. Kea is split into various packages. You may check the entire list on cloudsmith.io or using apt/yum/dnf. For example:

```
$ apt-cache search isc-kea
```

5. Install specified packages:

   ```
   $ sudo apt-get install isc-kea-dhcp6-server
   ```

   or all packages:

   ```
   $ sudo apt-get install isc-kea*
   ```

   or all packages with specifying version number:

   ```
   $ sudo apt-get install isc-kea*=1.6.2-isc0043420200221140216
   ```

6. All installed packages should be now available directly, for example:

   ```
   # kea-dhcp6 -c /path/to/your/kea6/config/file.json
   ```

   or using systemd:

   ```
   # systemctl restart isc-kea-dhcp6-server
   ```

   `keactrl` is not available in packages as similar functionality is provided by the native systemctl scripts.

2.3 Quick Start Guide for DHCPv4 and DHCPv6 Services

1. Edit the Kea configuration files which by default are installed in the `[kea-install-dir]/etc/kea/` directory. These are: `kea-dhcp4.conf`, `kea-dhcp6.conf`, `kea-dhcp-ddns.conf` and `kea-ctrl-agent.conf`, for DHCPv4 server, DHCPv6 server, D2, and Control Agent, respectively.

2. In order to start the DHCPv4 server in the background, run the following command (as root):

   ```
   # keactrl start -s dhcp4
   ```

   Or run the following command to start the DHCPv6 server instead:

   ```
   # keactrl start -s dhcp6
   ```

   Note that it is also possible to start all servers simultaneously:

   ```
   # keactrl start
   ```

3. Verify that the Kea server(s) is/are running:

   ```
   # keactrl status
   ```

   A server status of “inactive” may indicate a configuration error. Please check the log file (by default named `[kea-install-dir]/var/log/kea-dhcp4.log`, `[kea-install-dir]/var/log/kea-dhcp6.log`, `[kea-install-dir]/var/log/kea-ddns.log` or `[kea-install-dir]/var/log/kea-ctrl-agent.log`) for the details of the error.

4. If the server has been started successfully, test that it is responding to DHCP queries and that the client receives a configuration from the server; for example, use the ISC DHCP client.

5. Stop running the server(s):

```
# keactrl stop
```

For instructions specific to your system, please read the system-specific notes, available in the Kea section of ISC's Knowledgebase.

The details of `keactrl` script usage can be found in *Managing Kea with keactrl*.

Once you have Kea services up and running, you may consider deploying a dashboard solution that would monitor running services. For more details, see *Monitoring Kea with Stork*.

2.4 Running the Kea Servers Directly

The Kea servers can be started directly, without the need to use `keactrl` or `systemctl`. To start the DHCPv4 server run the following command:

```
# kea-dhcp4 -c /path/to/your/kea4/config/file.json
```

Similarly, to start the DHCPv6 server run the following command:

```
# kea-dhcp6 -c /path/to/your/kea6/config/file.json
```

CHAPTER

THREE

INSTALLATION

3.1 Packages

Starting with Kea 1.6.0, ISC now publishes native RPM, deb and APK packages along with the tarballs with the source code. The packages are available on Cloudsmith at https://cloudsmith.io/~isc/repos. You can download the native packages and install them using the system available in your distribution (such as dpkg or rpm). Also, you can add the Kea repository to your system, which will make it easier to install updates. For details, please go to https://cloudsmith.io/~isc/repos, choose the repository of interest and then click the `Set Me Up` button for detailed instructions.

3.2 Installation Hierarchy

The following is the directory layout of the complete Kea installation. (All directory paths are relative to the installation directory):

- `etc/kea/` — configuration files.
- `include/` — C++ development header files.
- `lib/` — libraries.
- `lib/kea/hooks` — additional hooks libraries.
- `sbin/` — server software and commands used by the system administrator.
- `share/kea/` — configuration specifications and examples.
- `share/doc/kea/` — this guide, other supplementary documentation, and examples.
- `share/man/` — manual pages (online documentation).
- `var/lib/kea/` — server identification, and lease databases files.
- `var/log/` - log files.
- `var/run/kea` - pid and logger lock files.

3.3 Build Requirements

In addition to the run-time requirements (listed in *Required Software at Run-Time*), building Kea from source code requires various development include headers and program development tools.

Note: Some operating systems have split their distribution packages into a run-time and a development package. You will need to install the development package versions, which include header files and libraries, to build Kea from the source code.

Building from source code requires the following software installed on the system:

- Boost C++ libraries (https://www.boost.org/). The oldest Boost version used for testing is 1.57 (although it may also work with older versions). The Boost system library must also be installed. Installing a header-only version of Boost is no longer recommended.
- OpenSSL (at least version 1.0.1) or Botan (at least version 2). Note that OpenSSL version 1.0.2 or 1.1.0 or later is strongly recommended.
- log4cplus (at least version 1.0.3) development include headers.
- A C++ compiler (with C++11 support) and standard development headers. The Kea build has been checked with GCC g++ 4.8.5 and some later versions, and Clang 800.0.38 and some later versions.
- The development tools automake, libtool, and pkg-config.
- The MySQL client and the client development libraries, when using the –with-mysql configuration flag to build the Kea MySQL database backend. In this case, an instance of the MySQL server running locally or on a machine reachable over a network is required. Note that running the unit tests requires a local MySQL server.
- The PostgreSQL client and the client development libraries, when using the –with-pgsql configuration flag to build the Kea PostgreSQL database backend. In this case an instance of the PostgreSQL server running locally or on some other machine, reachable over the network from the machine running Kea, is required. Note that running the unit tests requires a local PostgreSQL server.
- The cpp-driver from DataStax is needed when using the –with-cql configuration flag to build Kea with the Cassandra database backend. In this case, an instance of the Cassandra server running locally or on some other machine, reachable over the network from the machine running Kea, is required. Note that running the unit tests requires a local Cassandra server.
- The FreeRADIUS client library is required to connect to a RADIUS server. (This is specified using the –with-freeradius configuration switch.)
- Sysrepo (version 0.7.6 or later) and libyang (version 0.16-r2 or later) are needed to connect to a Sysrepo database. (This is specified using the –with-sysrepo switch when running "configure".)
- googletest (version 1.8 or later) is required when using the –with-gtest configuration option to build the unit tests.
- The documentation generation tools Sphinx, texlive with its extensions and Doxygen, if using the –enable-generate-docs configuration option to create the documentation. Particularly, in case of Fedora: python3-sphinx, texlive and texlive-collection-latexextra; in case of Ubuntu: python3-sphinx, python3-sphinx-rtd-theme and texlive-binaries. If LaTeX packages are missing, Kea will skip PDF generation and will produce only HTML documents.

Visit ISC's Knowledgebase at https://kb.isc.org/docs/installing-kea for system-specific installation tips.

3.4 Installation from Source

Although Kea may be available in pre-compiled, ready-to-use packages from operating system vendors, it is open source software written in C++. As such, it is freely available in source code form from ISC as a downloadable tar file. The source code can also be obtained from the Kea Gitlab repository at https://gitlab.isc.org/isc-projects/kea. This section describes how to build Kea from the source code.

3.4.1 Download Tar File

The Kea release tarballs may be downloaded from: https://downloads.isc.org/isc/kea/ .

3.4.2 Retrieve from Git

Downloading this "bleeding edge" code is recommended only for developers or advanced users. Using development code in a production environment is not recommended.

Note: When building from source code retrieved via git, additional software will be required: automake (v1.11 or later), libtoolize, and autoconf (v2.69 or later). These may need to be installed.

The latest development code is available on GitLab (see https://gitlab.isc.org/isc-projects/kea). The Kea source is public and development is done in the "master" branch.

The code can be checked out from `https://gitlab.isc.org/isc-projects/kea.git`:

```
$ git clone https://gitlab.isc.org/isc-projects/kea.git
```

The code checked out from the git repository does not include the generated configure script, the Makefile.in files, nor their related build files. They can be created by running `autoreconf` with the `--install` switch. This will run `autoconf`, `aclocal`, `libtoolize`, `autoheader`, `automake`, and related commands.

Write access to the Kea repository is only granted to ISC staff. If you are a developer planning to contribute to Kea, please check our Contributor's Guide. The Kea Developer's Guide contains more information about the process, as well as describes the requirements for contributed code to be accepted by ISC.

3.4.3 Configure Before the Build

Kea uses the GNU Build System to discover build environment details. To generate the makefiles using the defaults, simply run:

```
$ ./configure
```

Run `./configure` with the `--help` switch to view the different options. Some commonly used options are:

- `--prefix` Define the installation location (the default is `/usr/local`).
- `--with-mysql` Build Kea with code to allow it to store leases and host reservations in a MySQL database.
- `--with-pgsql` Build Kea with code to allow it to store leases and host reservations in a PostgreSQL database.
- `--with-cql` Build Kea with code to allow it to store leases and host reservations in a Cassandra (CQL) database.
- `--with-log4cplus` Define the path to find the Log4cplus headers and libraries. Normally this is not necessary.
- `--with-boost-include` Define the path to find the Boost headers. Normally this is not necessary.
- `--with-botan-config` Specify the path to the botan-config script to build with Botan for cryptographic functions. It is preferable to use OpenSSL (see below).
- `--with-openssl` Replace Botan by the OpenSSL the cryptographic library. By default `configure` searches for a valid Botan installation. If one is not found, it searches for OpenSSL. Normally this is not necessary.

- `--enable-shell` Build the optional `kea-shell` tool (more in *The Kea Shell*). The default is to not build it.
- `--with-site-packages` Only useful when `kea-shell` is enabled. It causes the kea-shell python packages to be installed in specified directory. This is mostly useful for Debian related distros. While most systems store python packages in ${prefix}/usr/lib/pythonX/site-packages, Debian introduced separate directory for packages installed from DEB. Such python packages are expected to be installed in /usr/lib/python3/dist-packages.
- `--enable-perfdhcp` Build the optional `perfdhcp` DHCP benchmarking tool. The default is to not build it.
- `--with-freeradius` Build the optional `RADIUS` hook. This option specifies path to the patched version of FreeRADIUS client. Available in subscriber only version. This option requires subscription-only RADIUS hook.
- `--with-freeradius-dictionary` Specifies a non-standard location for a FreeRADIUS dictionary file. That file contains list of supported RADIUS attributes. Available in subscriber only version. This option requires subscription-only RADIUS hook.

If you don't see the RADIUS options, make sure you have RADIUS hook sources in the `premium` directory and rerun `autoreconf -i`.

Note: The `--runstatedir` in the installation directories is particular. There are three cases:

- You use autoconf 2.70 or greater which supports this, but this autoconf version has not been released yet.
- You use autoconf 2.69 patched to add support of this. In this case and the previous simply use when needed the‘‘–runstatedir‘‘ configure parameter.
- There is no support (the configure parameter is not recognized and configure directly raises an error). For autoconf 2.69 the `runstatedir` environment variable is supported so simply remove the `--` before `runstatedir` in the configure script call, e.g.: `./configure runstatedir=/opt/run ...`

Note: For instructions concerning the installation and configuration of database backends for Kea, see *DHCP Database Installation and Configuration*.

There are also many additional options that are typically not necessary for regular users. However, they may be useful for package maintainers, developers, or people who want to extend Kea code or send patches:

- `--with-gtest`, `--with-gtest-source` Enable the building of the C++ Unit Tests using the Google Test framework. This option specifies the path to the gtest source. (If the framework is not installed on your system, it can be downloaded from https://github.com/google/googletest.)
- `--enable-generate-docs` Enable the rebuilding Kea documentation. ISC publishes Kea documentation for each release; however, in some cases you may want to rebuild it. For example, if you want to change something in the docs, or want to generate new ones from git sources that are not released yet.
- `--enable-generate-parser` Many Kea components have parsers implemented using flex (.ll files) and bison (.yy files). Kea sources have C++/h files generated out from them. By default Kea does not use flex or bison to avoid requiring installation of unnecessary dependencies for users. However, if you change anything in the parses (such as adding a new parameter), you will need to use flex and bison to regenerate parsers. This option lets you do that.
- `--enable-generate-messages` Enable the regeneration of messages files from their messages source files, e.g. regenerate xxx_messages.h and xxx_messages.cc from xxx_messages.mes using the Kea message compiler. By default Kea is built using these .h and .cc files from the distribution. However, if you change

anything in a .mes file (such as adding a new message), you will need to build and use the Kea message compiler. This option lets you do that.

- `--with-benchmark`, `--with-benchmark-source` Enable the building of the database backend benchmarks using the Google Benchmark framework. This option specifies the path to the gtest source. (If the framework is not installed on your system, it can be downloaded from https://github.com/google/benchmark.) This support is experimental.

For example, the following command configures Kea to find the Boost headers in /usr/pkg/include, specifies that PostgreSQL support should be enabled, and sets the installation location to /opt/kea:

```
$ ./configure \
      --with-boost-include=/usr/pkg/include \
      --with-pgsql=/usr/local/bin/pg_config \
      --prefix=/opt/kea
```

If you have any problems with building Kea using the header-only Boost code, or you'd like to use the Boost system library (assumed for the sake of this example to be located in /usr/pkg/lib):

```
$ ./configure \
      --with-boost-libs=-lboost_system \
      --with-boost-lib-dir=/usr/pkg/lib
```

If `configure` fails, it may be due to missing or old dependencies.

If `configure` succeeds, it displays a report with the parameters used to build the code. This report is saved into the file `config.report` and is also embedded into the executable binaries, e.g., `kea-dhcp4`.

3.4.4 Build

After the configure step is complete, build the executables from the C++ code and prepare the Python scripts by running the command:

```
$ make
```

3.4.5 Install

To install the Kea executables, support files, and documentation, issue the command:

```
$ make install
```

Do not use any form of parallel or job server options (such as GNU make's `-j` option) when performing this step; doing so may cause errors.

Note: The install step may require superuser privileges.

If required, run `ldconfig` as root with `/usr/local/lib` (or with prefix/lib if configured with –prefix) in `/etc/ld.so.conf` (or the relevant linker cache configuration file for your OS):

```
$ ldconfig
```

Note: If you do not run `ldconfig` where it is required, you may see errors like the following:

```
program: error while loading shared libraries: libkea-something.so.1:
cannot open shared object file: No such file or directory
```

3.4.6 Cross-Building

It is possible to cross-build Kea, i.e. to create binaries in another system (the `build` system) than the system where Kea will run (the `host` system).

It is outside of the scope of common administrator operations and requires some developer skills so the Developer Guide explains how to do that using a x86_64 Linux system to build Kea for a Raspberry Pi box running Raspbian: Kea Cross-Compiling Example.

3.5 DHCP Database Installation and Configuration

Kea stores its leases in a lease database. The software has been written in a way that makes it possible to choose which database product should be used to store the lease information. Kea supports four database backends: MySQL, PostgreSQL, Cassandra, and memfile. To limit external dependencies, MySQL, PostgreSQL, and Cassandra support are disabled by default and only memfile is available. Support for the optional external database backend must be explicitly included when Kea is built. This section covers the building of Kea with one of the optional backends and the creation of the lease database.

Note: When unit tests are built with Kea (i.e. the –with-gtest configuration option is specified), the databases must be manually pre-configured for the unit tests to run. The details of this configuration can be found in the Kea Developer's Guide.

3.5.1 Building with MySQL Support

Install MySQL according to the instructions for your system. The client development libraries must be installed.

Build and install Kea as described in *Installation*, with the following modification. To enable the MySQL database code, at the "configure" step (see *Configure Before the Build*), the –with-mysql switch should be specified:

```
$ ./configure [other-options] --with-mysql
```

If MySQL was not installed in the default location, the location of the MySQL configuration program "mysql_config" should be included with the switch, i.e.

```
$ ./configure [other-options] --with-mysql=path-to-mysql_config
```

See *First-Time Creation of the MySQL Database* for details regarding MySQL database configuration.

3.5.2 Building with PostgreSQL support

Install PostgreSQL according to the instructions for your system. The client development libraries must be installed. Client development libraries are often packaged as "libpq".

Build and install Kea as described in *Installation*, with the following modification. To enable the PostgreSQL database code, at the "configure" step (see *Configure Before the Build*), the –with-pgsql switch should be specified:

```
$ ./configure [other-options] --with-pgsql
```

If PostgreSQL was not installed in the default location, the location of the PostgreSQL configuration program "pg_config" should be included with the switch, i.e.

```
$ ./configure [other-options] --with-pgsql=path-to-pg_config
```

See *First-Time Creation of the PostgreSQL Database* for details regarding PostgreSQL database configuration.

3.5.3 Building with CQL (Cassandra) Support

Install Cassandra according to the instructions for your system. The Cassandra project website contains useful pointers: https://cassandra.apache.org.

If you have a cpp-driver package available as binary or as source, simply install or build and install the package. Then build and install Kea as described in *Installation*. To enable the Cassandra (CQL) database code, at the "configure" step (see *Configure Before the Build*), enter:

```
$ ./configure [other-options] --with-cql=path-to-pkg-config
```

Note if `pkg-config` is at its standard location (and thus in the shell path) you do not need to supply its path. If it does not work (e.g. no pkg-config, package not available in pkg-config with the cassandra name), you can still use the `cql_config` script in tools/ as described below.

Download and compile cpp-driver from DataStax. For details regarding dependencies for building cpp-driver, see the project homepage https://github.com/datastax/cpp-driver. In June 2016, the following commands were used:

```
$ git clone https://github.com/datastax/cpp-driver
$ cd cpp-driver
$ mkdir build
$ cd build
$ cmake ..
$ make
```

As of January 2019, cpp-driver does not include cql_config script. Work is in progress to contribute such a script to the cpp-driver project but, until that is complete, intermediate steps need to be conducted. A cql_config script is present in the tools/ directory of the Kea sources. Before using it, please create a cql_config_defines.sh file in the same directory (there is an example available in cql_config_define.sh.sample; you may copy it over to cql_config_defines.sh and edit the path specified in it) and change the environment variable CPP_DRIVER_PATH to point to the directory where the cpp-driver sources are located. Make sure that appropriate access rights are set on this file. It should be executable by the system user building Kea.

Build and install Kea as described in *Installation*, with the following modification. To enable the Cassandra (CQL) database code, at the "configure" step (see *Configure Before the Build*), enter:

```
$ ./configure [other-options] --with-cql=path-to-cql_config
```

3.6 Hammer Building Tool

An optional building tool called Hammer was introduced with Kea 1.6.0. It is a Python 3 script that lets users automate tasks related to building Kea, such as setting up virtual machines, installing Kea dependencies, compiling Kea with various options, running unit-tests and more. This tool was created primarily for internal QA purposes at ISC and it is not included in the Kea distribution. However, it is available in the Kea git repository. This tool was developed primarily for internal purposes and ISC cannot guarantee its proper operation. If you decide to use it, please do so with care.

Note: Use of this tool is completely optional. Everything it does can be done manually.

The first-time user is strongly encouraged to look at Hammer's built-in help:

```
$ ./hammer.py --help
```

It will list available parameters.

Hammer is able to set up various operating systems running either in LXC or in VirtualBox. To list of supported systems, use the `supported-systems` command:

```
$ ./hammer.py supported-systems
fedora:
  - 27: lxc, virtualbox
  - 28: lxc, virtualbox
  - 29: lxc, virtualbox
centos:
  - 7: lxc, virtualbox
rhel:
  - 8: virtualbox
ubuntu:
  - 16.04: lxc, virtualbox
  - 18.04: lxc, virtualbox
  - 18.10: lxc, virtualbox
debian:
  - 8: lxc, virtualbox
  - 9: lxc, virtualbox
freebsd:
  - 11.2: virtualbox
  - 12.0: virtualbox
```

It is also possible to run the build locally, in the current system (if the OS is supported).

First, you must install the Hammer dependencies: Vagrant and either VirtualBox or LXC. To make life easier, Hammer can install Vagrant and the required Vagrant plugins using the command:

```
$ ./hammer.py ensure-hammer-deps
```

VirtualBox and LXC need to be installed manually.

The basic functions provided by Hammer are to prepare the build environment and perform the actual build, and to run the unit tests locally in the current system. This can be achieved by running the command:

```
$ ./hammer.py build -p local
```

The scope of the process can be defined using –with (-w) and –without (-x) options. By default the build command will build Kea with documentation, install it locally, and run unit tests.

To exclude the installation and generation of docs, type:

```
$ ./hammer.py build -p local -x install docs
```

The basic scope can be extended by: mysql, pgsql, cql, native-pkg, radius, shell, and forge.

Note: To build Kea locally, Hammer dependencies like Vagrant are not needed.

Hammer can be told to set up a new virtual machine with a specified operating system, without the build:

```
$ ./hammer.py prepare-system -p virtualbox -s freebsd -r 12.0
```

This way we can prepare a system for our own use. To get to such a system using SSH, invoke:

```
$ ./hammer.py ssh -p virtualbox -s freebsd -r 12.0
```

It is possible to speed up subsequent Hammer builds. To achieve this Hammer employs ccache. During compilation, ccache stores objects in a shared folder. In subsequent runs, instead of doing an actual compilation, ccache returns the stored earlier objects. The cache with these objects for reuse needs to be stored outside of VM or LXC. To indicate the folder, you must indicate the –ccache-dir parameter for Hammer. In the indicated folder, there are separate stored objects for each target operating system.

```
$ ./hammer.py build -p lxc -s ubuntu -r 18.04 --ccache-dir ~/kea-ccache
```

Note: ccache is currently only supported for LXC in Hammer; support for VirtualBox may be added later.

For more information check:

```
$ ./hammer.py --help
```

3.7 Running Kea from non-root account on Linux

Both Kea DHCPv4 and DHCPv6 servers perform operations that in general require root access privileges. In particular, DHCPv4 opens raw sockets and both DHCPv4 and DHCPv6 open UDP sockets on privileged ports. However, with some extra system configuration, it is possible to run Kea from non-root accounts.

First, a regular user account must be created:

```
useradd admin
```

Then, change the binaries ownership and group to new user. Note your path may be different. Please refer to the `--prefix` parameter passed to the configure script.:

```
chown -R admin /opt/kea
chgrp -R admin /opt/kea
chown -R admin /var/log/kea-dhcp4.log
chgrp -R admin /var/log/kea-dhcp4.log
chown -R admin /var/log/kea-dhcp6.log
chgrp -R admin /var/log/kea-dhcp6.log
```

Assuming you are using systemd, you also should modify its service file (e.g. /etc/systemd/system/kea-dhcp6.service):

```
User=admin
Group=admin
```

The most important step is to set capabilities of the binaries. Refer to *man capabilities* to get more information.

```
setcap 'cap_net_bind_service=+ep' /opt/kea/sbin/kea-dhcp4
setcap 'cap_net_raw=+ep' /opt/kea/sbin/kea-dhcp4
setcap 'cap_net_bind_service=+ep' /opt/kea/sbin/kea-dhcp6
```

After this step is complete, the admin user should be able to run Kea. Note that DHCPv4 server by default opens raw sockets. If your network is only using relayed traffic, you can instruct Kea to use regular UDP sockets (refer to `dhcp-socket-type` parameter in the *Interface Configuration* section) and the `cap_net_raw` capability can be skipped.

Note: An alternative approach to avoiding running Kea with root privileges assumes instructing Kea to use non-privileged (greater than 1024) posts and redirecting traffic. This, however, will work only for relayed traffic. This approach in general is considered experimental and not tested enough for deployment in production environment. Use with caution!

To use this approach, configure the server to listen on other non privileged port (eg: 1547 and 1548) by running the process with `-p` option in `/etc/systemd/system/kea-dhcp4.service`:

```
ExecStart=/opt/kea/sbin/kea-dhcp4 -d -c /etc/kea/kea-dhcp4.conf -p 2067
```

and `/etc/systemd/system/kea-dhcp4.service`:

```
ExecStart=/opt/kea/sbin/kea-dhcp6 -d -c /etc/kea/kea-dhcp6.conf -p 1547
```

and then configure port redirection with iptables and ip6tables for new ports (eg: 1547 and 1548). Make sure you replace ens4 with your specific interface name.

```
iptables -t nat -A PREROUTING -i ens4 -p udp --dport 67 -j REDIRECT --to-port 2067
iptables -t nat -A PREROUTING -i ens4 -p udp --dport 2068 -j REDIRECT --to-port 68
ip6tables -t nat -A PREROUTING -i ens4 -p udp --dport 547 -j REDIRECT --to-port 1547
ip6tables -t nat -A PREROUTING -i ens4 -p udp --dport 1548 -j REDIRECT --to-port 548
```

CHAPTER

FOUR

KEA DATABASE ADMINISTRATION

4.1 Databases and Database Version Numbers

Kea may be configured to use a database as a storage for leases or as a source of servers' configurations and host reservations (i.e. static assignments of addresses, prefixes, options, etc.). Kea updates introduce changes to the database schemas to faciliate new features and correct discovered issues with the existing schemas.

A given version of Kea expects a particular structure in the backend and checks for this by examining the version of the database it is using. Separate version numbers are maintained for backends, independent of the version of Kea itself. It is possible that the backend version will stay the same through several Kea revisions; similarly, it is possible that the version of the backend may go up several revisions during a Kea upgrade. Versions for each backend are independent, so an increment in the MySQL backend version does not imply an increment in that of PostgreSQL.

Backend versions are specified in a major.minor format. The minor number is increased when there are backwards-compatible changes introduced; for example, the addition of a new index. It is desirable but not mandatory to apply such a change; running an older backend version is possible. (Although, in the example given, running without the new index may introduce a performance penalty.) On the other hand, the major number is increased when an incompatible change is introduced; for example, an extra column is added to a table. If Kea is run on a backend that is too old (as signified by a mismatched backend major version number), Kea will refuse to run; administrative action will be required to upgrade the backend.

4.2 The kea-admin Tool

To manage the databases, Kea provides the `kea-admin` tool. It is able to initialize a new backend, check its version number, perform a backend upgrade, and dump lease data to a text file.

`kea-admin` takes two mandatory parameters: `command` and `backend`. Additional, non-mandatory options may be specified. The currently supported commands are:

- `db-init` — Initializes a new database schema. This is useful during a new Kea installation. The database is initialized to the latest version supported by the version of the software being installed.
- `db-version` — Reports the database backend version number. This is not necessarily equal to the Kea version number as each backend has its own versioning scheme.
- `db-upgrade` — Conducts a database schema upgrade. This is useful when upgrading Kea.
- `lease-dump` — Dumps the contents of the lease database (for MySQL, PostgreSQL, or CQL backends) to a CSV (comma-separated values) text file. The first line of the file contains the column names. This is meant to be used as a diagnostic tool, so it provides a portable, human-readable form of the lease data.

Note: In previous versions of Kea earlier than 1.6.0 *db-init*, *db-version* and *db-upgrade* commands were named *lease-init*, *lease-version* and *lease-upgrade*.

`backend` specifies the type of backend database. The currently supported types are:

- `memfile` — Lease information is stored on disk in a text file.
- `mysql` — Information is stored in a MySQL relational database.
- `pgsql` — Information is stored in a PostgreSQL relational database.
- `cql` — Information is stored in an Apache Cassandra database.

Additional parameters may be needed, depending on the setup and specific operation: username, password, and database name or the directory where specific files are located. See the appropriate manual page for details (`man 8 kea-admin`).

4.3 Supported Backends

The following table presents the capabilities of available backends. Please refer to the specific sections dedicated to each backend to better understand their capabilities and limitations. Choosing the right backend may be essential for the success of the deployment.

Table 4.1: List of available backends

Feature	Memfile	MySQL	PostgreSQL	CQL (Cassandra)
Status	Stable	Stable	Stable	Experimental
Data format	CSV file	SQL RMDB	SQL RMDB	NoSQL database (Cassandra)
Leases	yes	yes	yes	yes
Host Reservations	no	yes	yes	yes
Options defined on per host basis	no	yes	yes	yes
Configuration Backend	no	yes	no	no

4.3.1 Memfile

The memfile backend is able to store lease information, but cannot store host reservation details; these must be stored in the configuration file. (There are no plans to add a host reservations storage capability to this backend.)

No special initialization steps are necessary for the memfile backend. During the first run, both `kea-dhcp4` and `kea-dhcp6` will create an empty lease file if one is not present. Necessary disk-write permission is required.

Upgrading Memfile Lease Files from an Earlier Version of Kea

There are no special steps required to upgrade memfile lease files from an earlier version of Kea to a new version of Kea. During startup the servers will check the schema version of the lease files against their own. If there is a mismatch, the servers will automatically launch the LFC process to convert the files to the server's schema version. While this mechanism is primarily meant to ease the process of upgrading to newer versions of Kea, it can also be used for downgrading should the need arise. When upgrading, any values not present in the original lease files will be assigned appropriate default values. When downgrading, any data present in the files but not in the server's schema will be dropped. To convert the files manually prior to starting the servers, run the LFC process. See *The LFC Process* for more information.

4.3.2 MySQL

MySQL is able to store leases, host reservations, options defined on a per-host basis, and a subset of the server configuration parameters (serving as a configuration backend). This section can be safely ignored if the data will be stored in other backends.

First-Time Creation of the MySQL Database

When setting up the MySQL database for the first time, the database area must be created within MySQL, and the MySQL user ID under which Kea will access the database must be set up. This needs to be done manually, rather than via `kea-admin`.

To create the database:

1. Log into MySQL as "root":

```
$ mysql -u root -p
Enter password:
mysql>
```

2. Create the MySQL database:

```
mysql> CREATE DATABASE database_name;
```

(database_name is the name chosen for the database.)

3. Create the user under which Kea will access the database (and give it a password), then grant it access to the database tables:

```
mysql> CREATE USER 'user-name'@'localhost' IDENTIFIED BY 'password';
mysql> GRANT ALL ON database-name.* TO 'user-name'@'localhost';
```

(user-name and password are the user ID and password being used to allow Kea access to the MySQL instance. All apostrophes in the command lines above are required.)

4. Create the database.

You'll need to exit mysql client

```
mysql> quit
Bye
```

and then use the `kea-admin` tool to create the database.

```
$ kea-admin db-init mysql -u database-user -p database-password -n␣
↪database-name
```

While it is possible to create the database from within mysql client, we recommend you use the kea-admin tool as it performs some necessary validations to ensure Kea can access the database at runtime. Among those checks is that the schema does not contain any pre-existing tables. If there are any pre-existing tables they must be removed manaully. An additional check examines the user's ability to create functions and triggers. If you encounter the following error:

```
ERROR 1419 (HY000) at line 1: You do not have the SUPER privilege and␣
↪binary logging is
enabled (you *might* want to use the less safe log_bin_trust_function_
↪creators variable)
ERROR/kea-admin: mysql_can_create cannot trigger, check user permissions,␣
↪mysql_status = 1
```

```
mysql: [Warning] Using a password on the command line interface can be␣
↪insecure.
ERROR/kea-admin: Create failed, the user, keatest, has insufficient␣
↪privileges.
```

Then user does not have the necessary permissions to create functions or triggers. The simplest way around this is to set the global MySQL variable, log_bin_trust_function_creators to 1 via mysql client. Note you must do this as a user with SUPER privileges:

```
mysql> set @@global.log_bin_trust_function_creators = 1;
Query OK, 0 rows affected (0.00 sec)
```

If you choose to create the database with mysql directly, you may do as as follows:

```
mysql> CONNECT database-name;
mysql> SOURCE path-to-kea/share/kea/scripts/mysql/dhcpdb_create.mysql
```

(path-to-kea is the location where Kea is installed.)

The database may also be dropped manually as follows:

```
mysql> CONNECT database-name;
mysql> SOURCE path-to-kea/share/kea/scripts/mysql/dhcpdb_drop.mysql
```

(path-to-kea is the location where Kea is installed.)

Warning: Dropping the database will result in the unrecoverable loss of any data it contains.

5. Exit MySQL:

```
mysql> quit
Bye
```

If the tables were not created in Step 4, run the `kea-admin` tool to create them now:

```
$ kea-admin db-init mysql -u database-user -p database-password -n database-name
```

Do not do this if the tables were created in Step 4. `kea-admin` implements rudimentary checks; it will refuse to initialize a database that contains any existing tables. To start from scratch, all must be removed data manually. (This process is a manual operation on purpose, to avoid possibly irretrievable mistakes by `kea-admin`.)

Upgrading a MySQL Database from an Earlier Version of Kea

Sometimes a new Kea version may use a newer database schema, so the existing database will need to be upgraded. This can be done using the `kea-admin db-upgrade` command.

To check the current version of the database, use the following command:

```
$ kea-admin db-version mysql -u database-user -p database-password -n database-name
```

(See *Databases and Database Version Numbers* for a discussion about versioning.) If the version does not match the minimum required for the new version of Kea (as described in the release notes), the database needs to be upgraded.

Before upgrading, please make sure that the database is backed up. The upgrade process does not discard any data, but depending on the nature of the changes, it may be impossible to subsequently downgrade to an earlier version. To perform an upgrade, issue the following command:

```
$ kea-admin db-upgrade mysql -u database-user -p database-password -n database-name
```

Note: To search host reservations by hostname it is critical the collation of the hostname column in the host table to be case-insensitive. Fortunately the default collation in MySQL is case-insensitive. You can verify this on your MySQL installation by:

```
mysql> SELECT COLLATION('');
+-----------------+
| COLLATION('')   |
+-----------------+
| utf8_general_ci |
+-----------------+
```

According to the naming of collations when the name finishes by `_ci` the collation is case-insensitive.

Simple MySQL tweak to gain performance

Changing MySQL internal value `innodb_flush_log_at_trx_commit` from default value `1` to `2` can result with huge gain in Kea performance. It can be set per session for testing:

```
mysql> SET GLOBAL innodb_flush_log_at_trx_commit=2;
mysql> SHOW SESSION VARIABLES LIKE 'innodb_flush_log%';
```

or permanently in `/etc/mysql/my.cnf`:

```
[mysqld]
innodb_flush_log_at_trx_commit=2
```

Be aware that changing this value can result with problems during data recovery after crash, we strongly recommend to check MySQL documentation.

4.3.3 PostgreSQL

PostgreSQL is able to store leases, host reservations, and options defined on a per-host basis. This step can be safely ignored if other database backends will be used.

First-Time Creation of the PostgreSQL Database

The first task is to create both the database and the user under which the servers will access it. A number of steps are required:

1. Log into PostgreSQL as “root”:

   ```
   $ sudo -u postgres psql postgres
   Enter password:
   postgres=#
   ```

2. Create the database:

   ```
   postgres=# CREATE DATABASE database-name;
   CREATE DATABASE
   postgres=#
   ```

(database-name is the name chosen for the database.)

3. Create the user under which Kea will access the database (and give it a password), then grant it access to the database:

```
postgres=# CREATE USER user-name WITH PASSWORD 'password';
CREATE ROLE
postgres=# GRANT ALL PRIVILEGES ON DATABASE database-name TO user-name;
GRANT
postgres=#
```

4. Exit PostgreSQL:

```
postgres=# \q
Bye
$
```

5. At this point, create the database tables either using the `kea-admin` tool, as explained in the next section (recommended), or manually. To create the tables manually, enter the following command. Note that PostgreSQL will prompt the administrator to enter the new user's password that was specified in Step 3. When the command completes, Kea will return to the shell prompt. The output should be similar to the following:

```
$ psql -d database-name -U user-name -f path-to-kea/share/kea/scripts/pgsql/
↪dhcpdb_create.pgsql
Password for user user-name:
CREATE TABLE
CREATE INDEX
CREATE INDEX
CREATE TABLE
CREATE INDEX
CREATE TABLE
START TRANSACTION
INSERT 0 1
INSERT 0 1
INSERT 0 1
COMMIT
CREATE TABLE
START TRANSACTION
INSERT 0 1
COMMIT
$
```

(path-to-kea is the location where Kea is installed.)

If instead an error is encountered, such as:

```
psql: FATAL:  no pg_hba.conf entry for host "[local]", user "user-name", database
↪"database-name", SSL off
```

... the PostgreSQL configuration will need to be altered. Kea uses password authentication when connecting to the database and must have the appropriate entries added to PostgreSQL's pg_hba.conf file. This file is normally located in the primary data directory for the PostgreSQL server. The precise path may vary depending on the operating system and version, but the default location for PostgreSQL 9.3 on Centos 6.5 is: `/var/lib/pgsql/9.3/data/pg_hba.conf`.

Assuming Kea is running on the same host as PostgreSQL, adding lines similar to the following should be sufficient to provide password-authenticated access to Kea's database:

```
local   database-name    user-name                                  password
host    database-name    user-name         127.0.0.1/32             password
host    database-name    user-name         ::1/128                  password
```

These edits are primarily intended as a starting point, and are not a definitive reference on PostgreSQL administration or database security. Please consult the PostgreSQL user manual before making these changes, as they may expose other databases that are running. It may be necessary to restart PostgreSQL in order for the changes to take effect.

Initialize the PostgreSQL Database Using kea-admin

If the tables were not created manually, do so now by running the `kea-admin` tool:

```
$ kea-admin db-init pgsql -u database-user -p database-password -n database-name
```

Do not do this if the tables were already created manually. `kea-admin` implements rudimentary checks; it will refuse to initialize a database that contains any existing tables. To start from scratch, all data must be removed manually. (This process is a manual operation on purpose, to avoid possibly irretrievable mistakes by `kea-admin`.)

Upgrading a PostgreSQL Database from an Earlier Version of Kea

The PostgreSQL database schema can be upgraded using the same tool and commands as described in *Upgrading a MySQL Database from an Earlier Version of Kea*, with the exception that the "pgsql" database backend type must be used in the commands.

Use the following command to check the current schema version:

```
$ kea-admin db-version pgsql -u database-user -p database-password -n database-name
```

Use the following command to perform an upgrade:

```
$ kea-admin db-upgrade pgsql -u database-user -p database-password -n database-name
```

4.3.4 Cassandra

Cassandra (sometimes for historical reasons referred to in documentation and commands as CQL) is the newest backend added to Kea; initial development was contributed by Deutsche Telekom. The Cassandra backend is able to store leases, host reservations, and options defined on a per-host basis.

Cassandra must be properly set up if Kea is to store information in it. This section can be safely ignored if the data will be stored in other backends.

First-Time Creation of the Cassandra Database

When setting up the Cassandra database for the first time, the keyspace area within it must be created. This needs to be done manually; it cannot be performed by `kea-admin`.

To create the database:

1. Export CQLSH_HOST environment variable:

```
$ export CQLSH_HOST=localhost
```

2. Log into CQL:

```
$ cqlsh
cql>
```

3. Create the CQL keyspace:

```
cql> CREATE KEYSPACE keyspace-name WITH replication = {'class' : 'SimpleStrategy',
↪'replication_factor' : 1};
```

(keyspace-name is the name chosen for the keyspace.)

4. At this point, the database tables can be created. (It is also possible to exit Cassandra and create the tables using the `kea-admin` tool, as explained below.) To do this:

```
cqslh -k keyspace-name -f path-to-kea/share/kea/scripts/cql/dhcpdb_create.cql
```

(path-to-kea is the location where Kea is installed.)

If the tables were not created in Step 4, do so now by running the `kea-admin` tool:

```
$ kea-admin db-init cql -n database-name
```

Do not do this if the tables were created in Step 4. `kea-admin` implements rudimentary checks; it will refuse to initialize a database that contains any existing tables. To start from scratch, all data must be removed manually. (This process is a manual operation on purpose, to avoid possibly irretrievable mistakes by `kea-admin`.)

Upgrading a Cassandra Database from an Earlier Version of Kea

Sometimes a new Kea version may use a newer database schema, so the existing database will need to be upgraded. This can be done using the `kea-admin db-upgrade` command.

To check the current version of the database, use the following command:

```
$ kea-admin db-version cql -n database-name
```

(See *Databases and Database Version Numbers* for a discussion about versioning.) If the version does not match the minimum required for the new version of Kea (as described in the release notes), the database needs to be upgraded.

Before upgrading, please make sure that the database is backed up. The upgrade process does not discard any data, but depending on the nature of the changes, it may be impossible to subsequently downgrade to an earlier version. To perform an upgrade, issue the following command:

```
$ kea-admin db-upgrade cql -n database-name
```

4.3.5 Using Read-Only Databases with Host Reservations

If a read-only database is used for storing host reservations, Kea must be explicitly configured to operate on the database in read-only mode. Sections *Using Read-Only Databases for Host Reservations with DHCPv4* and *Using Read-Only Databases for Host Reservations with DHCPv6* describe when such a configuration may be required, and how to configure Kea to operate in this way for both DHCPv4 and DHCPv6.

4.3.6 Limitations Related to the Use of SQL Databases

Year 2038 Issue

The lease expiration time is stored in the SQL database for each lease as a timestamp value. Kea developers observed that the MySQL database doesn't accept timestamps beyond 2147483647 seconds (the maximum signed 32-bit number) from the beginning of the UNIX epoch (00:00:00 on 1 January 1970). Some versions of PostgreSQL do accept greater values, but the value is altered when it is read back. For this reason, the lease database backends put a restriction on the maximum timestamp to be stored in the database, which is equal to the maximum signed 32-bit number. This effectively means that the current Kea version cannot store leases whose expiration time is later than 2147483647 seconds since the beginning of the epoch (around year 2038). This will be fixed when the database support for longer timestamps is available.

CHAPTER

FIVE

KEA CONFIGURATION

Kea uses JSON structures to represent server configurations. The following sections describe how the configuration structures are organized.

5.1 JSON Configuration

JSON is the notation used throughout the Kea project. The most obvious usage is for the configuration file, but JSON is also used for sending commands over the Management API (see *Management API*) and for communicating between DHCP servers and the DDNS update daemon.

Typical usage assumes that the servers are started from the command line, either directly or using a script, e.g. `keactrl`. The configuration file is specified upon startup using the -c parameter.

5.1.1 JSON Syntax

Configuration files for the DHCPv4, DHCPv6, DDNS, Control Agent, and NETCONF modules are defined in an extended JSON format. Basic JSON is defined in RFC 7159 and ECMA 404. In particular, the only boolean values allowed are true or false (all lowercase). The capitalized versions (True or False) are not accepted.

Even if the JSON standard (ECMA 404) does not require JSON objects (i.e. name/value maps) to have unique entries Kea implements them using a C++ STL map with unique entries so:

- if there are multiple values for the same name in an object/map the last value overwrites previous values
- configuration file parsers since Kea 1.9.0 version raise a syntax error in such cases

Kea components use an extended JSON with additional features allowed:

- shell comments: any text after the hash (#) character is ignored.
- C comments: any text after the double slashes (//) character is ignored.
- Multiline comments: any text between /* and */ is ignored. This commenting can span multiple lines.
- File inclusion: JSON files can include other JSON files by using a statement of the form <?include "file.json"?>.

The configuration file consists of a single object (often colloquially called a map) started with a curly bracket. It comprises only one of the "Dhcp4", "Dhcp6", "DhcpDdns", "Control-agent", or "Netconf" objects. It is possible to define additional elements but they will be ignored.

A very simple configuration for DHCPv4 could look like this:

```
# The whole configuration starts here.
{
    # DHCPv4 specific configuration starts here.
    "Dhcp4": {
        "interfaces-config": {
            "interfaces": [ "eth0" ],
            "dhcp-socket-type": "raw"
        },
        "valid-lifetime": 4000,
        "renew-timer": 1000,
        "rebind-timer": 2000,
        "subnet4": [{
           "pools": [ { "pool": "192.0.2.1-192.0.2.200" } ],
           "subnet": "192.0.2.0/24"
        }],

       # Now loggers are inside the DHCPv4 object.
       "loggers": [{
            "name": "*",
            "severity": "DEBUG"
        }]
    }

# The whole configuration structure ends here.
}
```

More examples are available in the installed `share/doc/kea/examples` directory.

Note: The “Logging” element is removed in Kea 1.6.0 and its contents (the “loggers” object) moved inside the configuration objects (maps) for the respective Kea modules. For example: the “Dhcp4” map contains the “loggers” object specifying logging configuration for the DHCPv4 server. Backward compatibility is maintained until Kea 1.7.10 release; it will be possible to specify the “Logging” object at the top configuration level and “loggers” objects at the module configuration level. Finally, support for the top-level “Logging” object was removed in Kea 1.7.10.

The specification for supporting several elements (e.g. “Dhcp4”, “Dhcp6”) in one file has been removed in Kea 1.7.10, so that each component requires one separate configuration file.

To avoid repetition of mostly similar structures, examples in the rest of this guide will showcase only the subset of parameters appropriate for a given context. For example, when discussing the IPv6 subnets configuration in DHCPv6, only subnet6 parameters will be mentioned. It is implied that the remaining elements (the global map that holds Dhcp6) are present, but they are omitted for clarity. Usually, locations where extra parameters may appear are denoted by an ellipsis (...).

5.1.2 Comments and User Context

You can specify shell, C or C++ style comments in the JSON configuration file if you use the file locally. This is convenient and works in simple cases where your configuration is kept statically using local file. However, since comments are not part of JSON syntax, most JSON tools will detect them as errors. Another problem with them is once Kea loads its configuration, the shell, C and C++ style comments are ignored. If you use commands such as *config-get* or *config-write*, those comments will be lost. An example of such comments has been presented in the previous section.

Historically, to address the problem Kea code allowed to put *comment* strings as valid JSON entities. This had the benefit of being retained through various operations (such as *config-get*) or allowed processing by JSON tools. An example JSON comment looks like this:

```
"Dhcp4": {
    "subnet4": [{
        "subnet": "192.0.2.0/24",
        "pools": [{ "pool": "192.0.2.10 - 192.0.2.20" }],
        "comment": "second floor"
    }]
}
```

However, users complained that the comment is only a single line and it's not possible to add any other information in more structured form. One specific example was a request to add floor levels and building numbers to subnets. This was one of the reasons why the concept of user context has been introduced. It allows adding arbitrary JSON structure to most Kea configuration structures. It has a number of benefits compared to earlier approaches. First, it is fully compatible with JSON tools and Kea commands. Second, it allows storing simple comment strings if you want to, but it can store much more complex data, such as multiple lines (as string array), extra typed data (such as floor numbers being actual numbers) and more. Third, the data is exposed to hooks, so it's possible to develop third party hooks that take advantage of that extra information. An example user context would look like this:

```
"Dhcp4": {
    "subnet4": [{
        "subnet": "192.0.2.0/24",
        "pools": [{ "pool": "192.0.2.10 - 192.0.2.20" }],
        "user-context": {
            "comment": "second floor",
            "floor": 2
        }
    }]
}
```

User contexts can store an arbitrary data file as long as it has valid JSON syntax and its top-level element is a map (i.e. the data must be enclosed in curly brackets). However, some hook libraries may expect specific formatting; please consult the specific hook library documentation for details.

In a sense the user context mechanism has superseeded the JSON comment capabilities. ISC would like to encourage people to use user-context in favor of the older mechanisms and we hope to deprecate JSON comments one day in the future. To promote this way of storing comments, Kea code is able to understand JSON comments, but converts them to user context on the fly. The comments entries in user-context were converted back to JSON comments to keep backward compatibility, but that conversion went away in version 1.7.9.

The is one side effect, however. If your configuration uses the old JSON comment, the *config-get* command will return a slightly modified configuration. Don't be surprised if you call *config-set* followed by a *config-get* and receive a slightly different structure. If this bothers you, the best way to avoid this problem is simply abandon JSON comments and start using user-context.

For a discussion about user context used in hooks, see *User Contexts in Hooks*.

5.1.3 Simplified Notation

It is sometimes convenient to refer to a specific element in the configuration hierarchy. Each hierarchy level is separated by a slash. If there is an array, a specific instance within that array is referenced by a number in square brackets (with numbering starting at zero). For example, in the above configuration the valid-lifetime in the Dhcp4 component can be referred to as Dhcp4/valid-lifetime and the pool in the first subnet defined in the DHCPv4 configuration as Dhcp4/subnet4[0]/pool.

5.2 Kea Configuration Backend

5.2.1 Applicability

Kea Configuration Backend (abbreviated as CB) is a feature first introduced in the 1.6.0 release, which provides Kea servers with the ability to manage and fetch their configuration from one or more databases. In the documentation, the term "Configuration Backend" may also refer to the particular Kea module providing support to manage and fetch the configuration information from the particular database type. For example: MySQL Configuration Backend is the logic implemented within the "mysql_cb" hooks library which provides a complete set of functions to manage and fetch the configuration information from the MySQL database.

In small deployments, e.g. those comprising a single DHCP server instance with limited and infrequently changing number of subnets, it may be impractical to use the CB as a configuration repository because it requires additional third-party software to be installed and configured - in particular the MySQL server and MySQL client. Once the number of DHCP servers and/or the number of managed subnets in the network grows, the usefulness of the CB becomes obvious.

A good example of a use case for the CB is a pair of Kea DHCP servers which can be configured to support High Availability as described in *ha: High Availability*. The configurations of both servers (including the value of the `server-tag` parameter) are almost exactly the same. They may differ by the server identifier and designation of the server as a primary or standby (or secondary). They may also differ by the interfaces configuration. Typically, the subnets, shared networks, option definitions, global parameters are the same for both servers and can be sourced from a single database instance to both Kea servers.

Using the database as a single source of configuration for subnets and/or other configuration information supported by the CB has the advantage that any modifications to the configuration in the database are automatically applied to both servers.

Another case when the centralized configuration repository is desired is in deployments including a large number of DHCP servers, possibly using a common lease database to provide redundancy. New servers can be added to the pool frequently to fulfill growing scalability requirements. Adding a new server does not require replicating the entire configuration to the new server when a common database is used.

Using the database as a configuration repository for Kea servers also brings other benefits, such as:

- the ability to use database specific tools to access the configuration information,
- the ability to create customized statistics based on the information stored in the database, and
- the ability to backup the configuration information using the database's built-in replication mechanisms.

5.2.2 CB Capabilities and Limitations

Kea CB, introduced in the 1.6.0 release, comes with a number of limitations as a result of the overall complexity of this feature and the development time constraints. This feature will evolve over time and the new capabilities will be added in subsequent releases. In this section we present the capabilities and limitations of the CB in the Kea 1.6.0 release:

- Kea CB is supported for the MySQL database only.
- Kea CB is only supported for DHCPv4 and DHCPv6 servers. Neither the Control Agent nor the D2 daemon can be configured via the database.
- Potential configurations to be stored for the DHCP servers include: global parameters, option definitions, global options, shared networks, and subnets. Other configuration parameters are not stored in the database and must be configured via the JSON configuration file.

Note: Kea CB stores data in a MySQL schema that is public. It's possible to insert configuration data into the MySQL tables manually, or automatically using SQL scripts, but this requires a reasonably good knowledge of SQL and the schema. The supported method for managing the data is through our cb-cmds hook library which provides management commands for config backends. It simplifies many typical operations, such as listing, adding, retrieving, and deleting of global parameters, shared networks, subnets, pools, options, and option definitions. In addition, it provides essential business logic that ensures logical integrity of the data. For a complete list, see commands starting with "remote-" in Appendix A of the Kea Administrator Reference Manual. The cb_cmds hooks library is available to subscribers only. If you are not a subscriber and would like to subscribe, please contact info@isc.org and our sales team will assist you.

The schema creation script can be found here dhcpdb_create.mysql and we have some related design documents in gitlab CB Design.

Note: We strongly recommend against duplication of the configuration information in the file and the database. For example, when specifying subnets for the DHCP server, please store them in either the configuration backend or in the configuration file, not both. Storing some subnets in the database and others in the file may put you at risk of potential configuration conflicts. Note that the configuration instructions from the database take precedence over instructions from the file, so it is possible that parts of the configuration specified in the file may be overridden if contradicted by information in the database.

Note: It is recommended that the `subnet_cmds` hooks library not be used to manage the subnets when the configuration backend is used as a source of information about the subnets. The `subnet_cmds` hooks library modifies the local subnets configuration in the server's memory, not in the database. Use the `cb_cmds` hooks library to manage the subnets information in the database instead.

5.2.3 CB Components

In order to use the Kea CB feature, the Kea 1.6.0 version or later is required. The `mysql_cb` open source hooks library implementing the Configuration Backend for MySQL must be compiled and loaded by the DHCP servers. This hooks library is compiled when the `--with-mysql` configuration switch is used during the Kea build. The MySQL C client libraries must be installed, as explained in *DHCP Database Installation and Configuration*.

Note: Any existing MySQL schema must be upgraded to the latest schema required by the particular Kea version using the `kea-admin` tool, as described in *The kea-admin Tool*.

The `cb_cmds` premium hooks library, which is available to ISC's paid support customers, provides a complete set of commands to manage the servers' configuration information within the database. This library can be attached to both DHCPv4 and DHCPv6 server instances. It is still possible to manage the configuration information without the `cb_cmds` hooks library with commonly available tools, such as MySQL Workbench or the command-line MySQL client, by directly working with the database.

Refer to *cb_cmds: Configuration Backend Commands* for the details regarding the `cb_cmds` hooks library.

The DHCPv4 and DHCPv6 server-specific configurations of the CB, as well as the list of supported configuration parameters, can be found in *Configuration Backend in DHCPv4* and *Configuration Backend in DHCPv6* respectively.

5.2.4 Configuration Sharing and Server Tags

The configuration database is designed to store the configuration information for multiple Kea servers. Depending on the use case, the entire configuration may be shared by all servers, parts of the configuration may be shared by multiple servers and the rest of the configuration may be different for these servers or, finally, each server may have its own non-shared configuration.

The configuration elements in the database are associated with the servers by "server tags". The server tag is an arbitrary string holding the name of the Kea server instance. The tags of the DHCPv4 and DHCPv6 servers are independent in the database, i.e. the same server tag can be created for the DHCPv4 and the DHCPv6 server respectively. The value is configured using `server-tag` parameter in the Dhcp4 or Dhcp6 scope. The current server-tag can be checked with the `server-tag-get` command.

The server definition, which consists of the server tag and the server description, must be stored in the configuration database prior to creating the dedicated configuration for that server. In cases when all servers use the same configuration, e.g. a pair of servers running as the High Availability peers, there is no need to configure the server tags for these servers in the database. The database by default includes the logical server *all*, which is used as a keyword to indicate that the particular piece of configuration must be shared between all servers connecting to the database. The *all* server can't be deleted or modified. It is not even returned among other servers as a result of the *remote-server[46]-get-all* commands. Also, slightly different rules may apply to "all" keyword than to any user defined server when running the commands provided by the *cb_cmds* hooks library *cb_cmds: Configuration Backend Commands*.

In the simplest case there are no server tags defined in the configuration database and all connecting servers will get the same configuration regardless of the server tag they are using. The server tag that the particular Kea instance presents to the database to fetch its configuration is specified in the Kea configuration file, using the *config-control* map (please refer to the *Enabling Configuration Backend* and *Enabling Configuration Backend* for details).

All Kea instances presenting the same server tag to the configuration database are given the same configuration. It is the administrator's choice whether multiple Kea instances use the same server tag or each Kea instance is using a different sever tag. Also, there is no requirement that the instances running on the same physical or virtual machine use the same server tag. It is even possible to configure the Kea server without assigning it a server tag. In such case the server will be given the configuration specified for "all" servers.

In order to differentiate the configurations between the Kea servers, a collection of the server tags used by the servers must be stored in the database. For the DHCPv4 and DHCPv6 servers, it can be done using the commands described in *remote-server4-set, remote-server6-set commands* and *remote-server4-set, remote-server6-set commands*. Next, the server tags can be used to associate the configuration information with the servers. However, it is important to note that some DHCP configuration elements may be associated with multiple server tags and other configuration elements may be associated with exactly one server tag. The former configuration elements are referred to as shareable configuration elements and the latter are referred to as non-shareable configuration elements. The *Configuration Backend in DHCPv4* and *Configuration Backend in DHCPv6* list the DHCP specific shareable and non-shareable configuration elements. However, in this section we want to briefly explain the difference between them.

The shareable configuration element is the one having some unique property identifying it and which instance may appear only once in the database. An example of the shareable DHCP element is a subnet instance. The subnet is a part of the network topology and we assume that the particular subnet may have only one definition within this network. The subnet has two unique identifiers: subnet id and the subnet prefix. The subnet identifier is used in Kea to uniquely identify the subnet and to connect it with other configuration elements, e.g. in host reservations. The subnet identifier uniquely identifies the subnet within the network. Some commands provided by the *cb_cmds* hooks library allow for accessing the subnet information by subnet identifier (or prefix) and explicitly prohibit using the server tag to access the subnet. This is because, in a general case, the subnet definition is associated with multiple servers rather than single server. In fact, it may even be associated with no servers (unassigned). Still, the unassigned subnet has an identifier and prefix which can be used to access the subnet.

A shareable configuration element may be associated with multiple servers, one server or no servers. Deletion of the server which is associated with the shareable element does not cause the deletion of the shareable element. It merely deletes the association of the deleted server with the element.

Unlike the shareable element, the non-shareable element must not be explicitly associated with more than one server and must not exist after the server is deleted (must not remain unassigned). The non-shareable element only exists within the context of the server. An example of the non-shareable element in DHCP is a global parameter, e.g. *renew-timer*. The renew timer is the value to be used by the particular server and only this server. Other servers may have their respective renew timers set to the same or different value. The renew timer is the parameter which has no unique identifier by which it could be accessed, modified or otherwise used. The global parameters like the renew timer can be accessed by the parameter name and the tag of the server for which they are configured. For example: the commands described in *The remote-global-parameter4-get, remote-global-parameter6-get Commands* allow for fetching the value of the global parameter by the parameter name and the server name. Getting the global parameter only by its name (without specifying the server tag) is not possible because there may be many global parameters with the given name in the database.

When the server associated with a non-shareable configuration element is deleted, the configuration element is automatically deleted from the database along with the server because the non-shareable element must be always assigned to some server (or the logical server "all").

The terms "shareable" and "non-shareable" only apply to the associations with user defined servers. All configuration elements associated with the logical server "all" are by definition shareable. For example: the *renew-timer* associated with "all" servers is used by all servers connecting to the database which don't have their specific renew timers defined. In the special case, when none of the configuration elements are associated with user defined servers, the entire configuration in the database is shareable because all its pieces belong to "all" servers.

Note: Be very careful when associating the configuration elements with different server tags. The configuration backend doesn't protect you against some possible misconfigurations that may arise from the wrong server tags' assignments. For example: if you assign a shared network to one server and the subnets belonging to this shared network to another server, the servers will fail upon trying to fetch and use this configuration. The server fetching the subnets will be aware that the subnets are associated with the shared network but the shared network will not be found by this server as it doesn't belong to it. In such case, both the shared network and the subnets should be assigned to the same set of servers.

CHAPTER

SIX

MANAGING KEA WITH KEACTRL

6.1 Overview

keactrl is a shell script which controls the startup, shutdown, and reconfiguration of the Kea servers (`kea-dhcp4`, `kea-dhcp6`, `kea-dhcp-ddns`, `kea-ctrl-agent`, and `kea-netconf`). It also provides the means for checking the current status of the servers and determining the configuration files in use.

6.2 Command Line Options

`keactrl` is run as follows:

```
# keactrl <command> [-c keactrl-config-file] [-s server[,server,...]]
```

`<command>` is one of the commands described in *Commands*.

The optional `-c keactrl-config-file` switch allows specification of an alternate `keactrl` configuration file. (`--ctrl-config` is a synonym for `-c`.) In the absence of `-c`, `keactrl` will use the default configuration file `[kea-install-dir]/etc/kea/keactrl.conf`.

The optional `-s server[,server,...]` switch selects the servers to which the command is issued. (`--server` is a synonym for `-s`.) If absent, the command is sent to all servers enabled in the keactrl configuration file. If multiple servers are specified, they should be separated by commas with no intervening spaces.

6.3 The keactrl Configuration File

Depending on requirements, not all of the available servers need to be run. The keactrl configuration file sets which servers are enabled and which are disabled. The default configuration file is `[kea-install-dir]/etc/kea/keactrl.conf`, but this can be overridden on a per-command basis using the `-c` switch.

The contents of `keactrl.conf` are:

```
# This is a configuration file for keactrl script which controls
# the startup, shutdown, reconfiguration and gathering the status
# of the Kea's processes.

# prefix holds the location where the Kea is installed.
prefix=@prefix@

# Location of Kea configuration file.
```

```
kea_dhcp4_config_file=@sysconfdir@/@PACKAGE@/kea-dhcp4.conf
kea_dhcp6_config_file=@sysconfdir@/@PACKAGE@/kea-dhcp6.conf
kea_dhcp_ddns_config_file=@sysconfdir@/@PACKAGE@/kea-dhcp-ddns.conf
kea_ctrl_agent_config_file=@sysconfdir@/@PACKAGE@/kea-ctrl-agent.conf
kea_netconf_config_file=@sysconfdir@/@PACKAGE@/kea-netconf.conf

# Location of Kea binaries.
exec_prefix=@exec_prefix@
dhcp4_srv=@sbindir@/kea-dhcp4
dhcp6_srv=@sbindir@/kea-dhcp6
dhcp_ddns_srv=@sbindir@/kea-dhcp-ddns
ctrl_agent_srv=@sbindir@/kea-ctrl-agent
netconf_srv=@sbindir@/kea-netconf

# Start DHCPv4 server?
dhcp4=yes

# Start DHCPv6 server?
dhcp6=yes

# Start DHCP DDNS server?
dhcp_ddns=no

# Start Control Agent?
ctrl_agent=yes

# Start Netconf?
netconf=no

# Be verbose?
kea_verbose=no
```

Note: In the example above, strings of the form @something@ are replaced by the appropriate values when Kea is installed.

The `dhcp4`, `dhcp6`, `dhcp_ddns`, `ctrl_agent`, and `netconf` parameters set to "yes" will configure `keactrl` to manage (start, reconfigure) all servers, i.e. `kea-dhcp4`, `kea-dhcp6`, `kea-dhcp-ddns`, `kea-ctrl-agent`, and `kea-netconf`. When any of these parameters is set to "no", the `keactrl` will ignore the corresponding server when starting or reconfiguring Kea. Some daemons (ddns and netconf) are disabled by default.

By default, Kea servers managed by `keactrl` are located in `[kea-install-dir]/sbin`. This should work for most installations. If the default location needs to be altered for any reason, the paths specified with the `dhcp4_srv`, `dhcp6_srv`, `dhcp_ddns_srv`, `ctrl_agent_srv`, and `netconf_srv` parameters should be modified.

The `kea_verbose` parameter specifies the verbosity of the servers being started. When `kea_verbose` is set to "yes" the logging level of the server is set to DEBUG. Modification of the logging severity in a configuration file, as described in *Logging*, will have no effect as long as the `kea_verbose` is set to "yes." Setting it to "no" will cause the server to use the logging levels specified in the Kea configuration file. If no logging configuration is specified, the default settings will be used.

Note: The verbosity for the server is set when it is started. Once started, the verbosity can be only changed by stopping the server and starting it again with the new value of the `kea_verbose` parameter.

6.4 Commands

The following commands are supported by `keactrl`:

- `start` - starts selected servers.
- `stop` - stops all running servers.
- `reload` - triggers reconfiguration of the selected servers by sending the SIGHUP signal to them.
- `status` - returns the status of the servers (active or inactive) and the names of the configuration files in use.
- `version` - prints out the version of the keactrl tool itself, together with the versions of the Kea daemons.

Typical output from `keactrl` when starting the servers looks similar to the following:

```
$ keactrl start
INFO/keactrl: Starting kea-dhcp4 -c /usr/local/etc/kea/kea-dhcp4.conf -d
INFO/keactrl: Starting kea-dhcp6 -c /usr/local/etc/kea/kea-dhcp6.conf -d
INFO/keactrl: Starting kea-dhcp-ddns -c /usr/local/etc/kea/kea-dhcp-ddns.conf -d
INFO/keactrl: Starting kea-ctrl-agent -c /usr/local/etc/kea/kea-ctrl-agent.conf -d
INFO/keactrl: Starting kea-netconf -c /usr/local/etc/kea/kea-netconf.conf -d
```

Kea's servers create PID files upon startup. These files are used by keactrl to determine whether a given server is running. If one or more servers are running when the start command is issued, the output will look similar to the following:

```
$ keactrl start
INFO/keactrl: kea-dhcp4 appears to be running, see: PID 10918, PID file: /usr/local/
↪var/run/kea/kea.kea-dhcp4.pid.
INFO/keactrl: kea-dhcp6 appears to be running, see: PID 10924, PID file: /usr/local/
↪var/run/kea/kea.kea-dhcp6.pid.
INFO/keactrl: kea-dhcp-ddns appears to be running, see: PID 10930, PID file: /usr/
↪local/var/run/kea/kea.kea-dhcp-ddns.pid.
INFO/keactrl: kea-ctrl-agent appears to be running, see: PID 10931, PID file: /usr/
↪local/var/run/kea/kea.kea-ctrl-agent.pid.
INFO/keactrl: kea-netconf appears to be running, see: PID 10123, PID file: /usr/local/
↪var/run/kea/kea.kea-netconf.pid.
```

During normal shutdowns these PID files are deleted. They may, however, be left over as remnants following a system crash. It is possible, though highly unlikely, that upon system restart the PIDs they contain may actually refer to processes unrelated to Kea. This condition will cause keactrl to decide that the servers are running, when in fact they are not. In such a case the PID files listed in the keactrl output must be manually deleted.

The following command stops all servers:

```
$ keactrl stop
INFO/keactrl: Stopping kea-dhcp4...
INFO/keactrl: Stopping kea-dhcp6...
INFO/keactrl: Stopping kea-dhcp-ddns...
INFO/keactrl: Stopping kea-ctrl-agent...
INFO/keactrl: Stopping kea-netconf...
```

Note that the `stop` command will attempt to stop all servers regardless of whether they are "enabled" in `keactrl.conf`. If any of the servers are not running, an informational message is displayed as in the `stop` command output below.

```
$ keactrl stop
INFO/keactrl: kea-dhcp4 isn't running.
INFO/keactrl: kea-dhcp6 isn't running.
```

```
INFO/keactrl: kea-dhcp-ddns isn't running.
INFO/keactrl: kea-ctrl-agent isn't running.
INFO/keactrl: kea-netconf isn't running.
```

As already mentioned, the reconfiguration of each Kea server is triggered by the SIGHUP signal. The `reload` command sends the SIGHUP signal to any servers that are enabled in the `keactrl` configuration file and that are currently running. When a server receives the SIGHUP signal it re-reads its configuration file and, if the new configuration is valid, uses the new configuration. A reload is executed as follows:

```
$ keactrl reload
INFO/keactrl: Reloading kea-dhcp4...
INFO/keactrl: Reloading kea-dhcp6...
INFO/keactrl: Reloading kea-dhcp-ddns...
INFO/keactrl: Reloading kea-ctrl-agent...
```

If any of the servers are not running, an informational message is displayed as in the `reload` command output below. Note that as of version 1.5.0, kea-netconf does not support the SIGHUP signal. If its configuration has changed, please stop and restart it for the change to take effect. This limitation will be removed in a future release.

```
$ keactrl stop
INFO/keactrl: kea-dhcp4 isn't running.
INFO/keactrl: kea-dhcp6 isn't running.
INFO/keactrl: kea-dhcp-ddns isn't running.
INFO/keactrl: kea-ctrl-agent isn't running.
INFO/keactrl: kea-netconf isn't running.
```

Note: NETCONF is an optional feature that is disabled by default and can be enabled during compilation. If Kea was compiled without NETCONF support, keactrl will do its best to not bother the user with information about it. The NETCONF entries will still be present in the keactrl.conf file, but NETCONF status will not be shown and other commands will ignore it.

Note: Currently `keactrl` does not report configuration failures when the server is started or reconfigured. To check if the server's configuration succeeded, the Kea log must be examined for errors. By default, this is written to the syslog file.

Sometimes it is useful to check which servers are running. The `status` command reports this, with typical output that looks like:

```
$ keactrl status
DHCPv4 server: active
DHCPv6 server: inactive
DHCP DDNS: active
Control Agent: active
Netconf agent: inactive
Kea configuration file: /usr/local/etc/kea/kea.conf
Kea DHCPv4 configuration file: /usr/local/etc/kea/kea-dhcp4.conf
Kea DHCPv6 configuration file: /usr/local/etc/kea/kea-dhcp6.conf
Kea DHCP DDNS configuration file: /usr/local/etc/kea/kea-dhcp-ddns.conf
Kea Control Agent configuration file: /usr/local/etc/kea/kea-ctrl-agent.conf
Kea Netconf configuration file: /usr/local/etc/kea/kea-netconf.conf
keactrl configuration file: /usr/local/etc/kea/keactrl.conf
```

`keactrl status` reporting capabilities are rather basic. If you need more extensive insight into the Kea health and status, you may consider deploying Stork. For details, see *Monitoring Kea with Stork*.

6.5 Overriding the Server Selection

The optional `-s` switch allows the selection of the server(s) to which the `keactrl` command is issued. For example, the following instructs `keactrl` to stop the `kea-dhcp4` and `kea-dhcp6` servers and leave the `kea-dhcp-ddns` and `kea-ctrl-agent` running:

```
$ keactrl stop -s dhcp4,dhcp6
```

Similarly, the following will start only the `kea-dhcp4` and `kea-dhcp-ddns` servers, but not `kea-dhcp6` or `kea-ctrl-agent`.

```
$ keactrl start -s dhcp4,dhcp_ddns
```

Note that the behavior of the `-s` switch with the `start` and `reload` commands is different from its behavior with the `stop` command. On `start` and `reload`, `keactrl` will check if the servers given as parameters to the `-s` switch are enabled in the `keactrl` configuration file; if not, the server will be ignored. For `stop`, however, this check is not made; the command is applied to all listed servers, regardless of whether they have been enabled in the file.

The following keywords can be used with the `-s` command line option:

- `dhcp4` for `kea-dhcp4`.
- `dhcp6` for `kea-dhcp6`.
- `dhcp_ddns` for `kea-dhcp-ddns`.
- `ctrl_agent` for `kea-ctrl-agent`.
- `netconf` for `kea-netconf`.
- `all` for all servers (default).

CHAPTER

SEVEN

THE KEA CONTROL AGENT

7.1 Overview of the Kea Control Agent

The Kea Control Agent (CA) is a daemon which exposes a RESTful control interface for managing Kea servers. The daemon can receive control commands over HTTP and either forward these commands to the respective Kea servers or handle these commands on its own. The determination whether the command should be handled by the CA or forwarded is made by checking the value of the "service" parameter, which may be included in the command from the controlling client. The details of the supported commands, as well as their structures, are provided in *Management API*.

The CA can use hook libraries to provide support for additional commands or custom behavior of existing commands. Such hook libraries must implement callouts for the "control_command_receive" hook point. Details about creating new hook libraries and supported hook points can be found in the Kea Developer's Guide.

The CA processes received commands according to the following algorithm:

- Pass command into any installed hooks (regardless of service value(s)). If the command is handled by a hook, return the response.
- If the service specifies one more or services, forward the command to the specified services and return the accumulated responses.
- If the service is not specified or is an empty list, handle the command if the CA supports it.

7.2 Configuration

The following example demonstrates the basic CA configuration.

```
{
    "Control-agent": {
        "http-host": "10.20.30.40",
        "http-port": 8000,
        "authentication": {
            "type": "basic",
            "realm": "kea-control-agent",
            "clients": [
            {
                "user": "admin",
                "password": "1234"
            } ]
        },

        "control-sockets": {
            "dhcp4": {
                "comment": "main server",
                "socket-type": "unix",
                "socket-name": "/path/to/the/unix/socket-v4"
            },
            "dhcp6": {
                "socket-type": "unix",
                "socket-name": "/path/to/the/unix/socket-v6",
                "user-context": { "version": 3 }
            },
            "d2": {
                "socket-type": "unix",
                "socket-name": "/path/to/the/unix/socket-d2"
            },
        },

        "hooks-libraries": [
        {
            "library": "/opt/local/control-agent-commands.so",
            "parameters": {
                "param1": "foo"
            }
        } ],

        "loggers": [ {
            "name": "kea-ctrl-agent",
            "severity": "INFO"
        } ]
    }
}
```

The `http-host` and `http-port` parameters specify an IP address and port to which HTTP service will be bound. In the example configuration provided above, the RESTful service will be available under the URL of `http://10.20.30.40:8000/`. If these parameters are not specified, the default URL is `http://127.0.0.1:8000/`.

As mentioned in *Overview of the Kea Control Agent*, the CA can forward received commands to the Kea servers for processing. For example, `config-get` is sent to retrieve the configuration of one of the Kea services. When the CA receives this command, including a `service` parameter indicating that the client wishes to retrieve the configuration of the DHCPv4 server, the CA forwards the command to that server and passes the received response back to the client. More about the `service` parameter and the general structure of commands can be found in *Management API*.

The CA uses UNIX domain sockets to forward control commands and receive responses from other Kea services. The `dhcp4`, `dhcp6`, and `d2` maps specify the files to which UNIX domain sockets are bound. In the configuration above, the CA will connect to the DHCPv4 server via `/path/to/the/unix/socket-v4` to forward the commands to it. Obviously, the DHCPv4 server must be configured to listen to connections via this same socket. In other words, the command socket configuration for the DHCPv4 server and the CA (for this server) must match. Consult *Management API for the DHCPv4 Server*, *Management API for the DHCPv6 Server* and *Management API for the D2 Server* to learn how the socket configuration is specified for the DHCPv4, DHCPv6, and D2 services.

> **Warning:** “dhcp4-server”, “dhcp6-server”, and “d2-server” were renamed to “dhcp4”, “dhcp6”, and “d2” respectively in Kea 1.2. If you are migrating from Kea 1.2, you must modify your CA configuration to use this new naming convention.

User contexts can store arbitrary data as long as they are in valid JSON syntax and their top-level element is a map (i.e. the data must be enclosed in curly brackets). Some hook libraries may expect specific formatting; please consult the relevant hook library documentation for details.

User contexts can be specified on either global scope, control socket, basic authentication, or loggers. One other useful feature is the ability to store comments or descriptions; the parser translates a "comment" entry into a user context with the entry, which allows a comment to be attached within the configuration itself.

The basic HTTP authentication was added by Kea 1.9.0. It protects against not authorized uses of the control agent by local users. For the protection against remote attackers HTTPS and reverse proxy of *Secure Connections* provide a stronger security.

The authentication is described in the `authentication` block with the mandatory `type` parameter which selects the authentication. Currently only the basic HTTP authentication (type basic) is supported.

The `realm` authentication parameter is used for error message when the basic HTTP authentication is required but the client is not authorized.

When the `clients` authentication list is configured and not empty the basic HTTP authentication is required. Each element of the list specifies a user id and a password. The user id is mandatory, must be not empty and must not contain the colon (:) character. The password is optional: when it is not specified an empty password is used.

Note: The basic HTTP authentication user id and password are encoded in UTF-8 but the current Kea JSON syntax only supports the latin-1 (i.e. 0x00..0xff) unicode subset.

Hooks libraries can be loaded by the Control Agent in the same way as they are loaded by the DHCPv4 and DHCPv6 servers. The CA currently supports one hook point - "control_command_receive" - which makes it possible to delegate processing of some commands to the hooks library. The `hooks-libraries` list contains the list of hooks libraries that should be loaded by the CA, along with their configuration information specified with `parameters`.

Please consult *Logging* for the details how to configure logging. The CA's root logger's name is `kea-ctrl-agent`, as given in the example above.

7.3 Secure Connections

The Control Agent does not natively support secure HTTP connections like SSL or TLS. In order to setup a secure connection, please use one of the available third-party HTTP servers and configure it to run as a reverse proxy to the Control Agent. Kea has been tested with two major HTTP server implementations working as a reverse proxy: Apache2 and nginx. Example configurations, including extensive comments, are provided in the `doc/examples/https/` directory.

The reverse proxy forwards HTTP requests received over a secure connection to the Control Agent using unsecured HTTP. Typically, the reverse proxy and the Control Agent are running on the same machine, but it is possible to configure them to run on separate machines as well. In this case, security depends on the protection of the communications between the reverse proxy and the Control Agent.

Apart from providing the encryption layer for the control channel, a reverse proxy server is also often used for authentication of the controlling clients. In this case, the client must present a valid certificate when it connects via reverse proxy. The proxy server authenticates the client by checking whether the presented certificate is signed by the certificate authority used by the server.

To illustrate this, the following is a sample configuration for the nginx server running as a reverse proxy to the Kea Control Agent. The server enables authentication of the clients using certificates.

```
#   The server certificate and key can be generated as follows:
#
#   openssl genrsa -des3 -out kea-proxy.key 4096
#   openssl req -new -x509 -days 365 -key kea-proxy.key -out kea-proxy.crt
#
#   The CA certificate and key can be generated as follows:
#
#   openssl genrsa -des3 -out ca.key 4096
#   openssl req -new -x509 -days 365 -key ca.key -out ca.crt
#
#
#   The client certificate needs to be generated and signed:
#
#   openssl genrsa -des3 -out kea-client.key 4096
#   openssl req -new -key kea-client.key -out kea-client.csr
#   openssl x509 -req -days 365 -in kea-client.csr -CA ca.crt \
#           -CAkey ca.key -set_serial 01 -out kea-client.crt
#
#   Note that the "common name" value used when generating the client
#   and the server certificates must differ from the value used
#   for the CA certificate.
#
#   The client certificate must be deployed on the client system.
#   In order to test the proxy configuration with "curl", run a
#   command similar to the following:
#
#   curl -k --key kea-client.key --cert kea-client.crt -X POST \
#        -H Content-Type:application/json -d '{ "command": "list-commands" }' \
#         https://kea.example.org/kea
#
#   curl syntax for basic authentication is -u user:password
#
#
#   nginx configuration starts here.

events {
}

http {
        #   HTTPS server
    server {
        #     Use default HTTPS port.
        listen 443 ssl;
        #     Set server name.
        server_name kea.example.org;

        #   Server certificate and key.
        ssl_certificate /path/to/kea-proxy.crt;
        ssl_certificate_key /path/to/kea-proxy.key;

        #   Certificate Authority. Client certificate must be signed by the CA.
        ssl_client_certificate /path/to/ca.crt;

        # Enable verification of the client certificate.
        ssl_verify_client on;

        # For URLs such as https://kea.example.org/kea, forward the
        # requests to http://127.0.0.1:8000.
        location /kea {
            proxy_pass http://127.0.0.1:8000;
        }
    }
}
```

Note: Note that the configuration snippet provided above is for testing purposes only. It should be modified according to the security policies and best practices of your organization.

When you use an HTTP client without TLS support as `kea-shell`, you can use an HTTP/HTTPS translator such as stunnel in client mode. A sample configuration is provided in the `doc/examples/https/shell/` directory.

7.4 Starting the Control Agent

The CA is started by running its binary and specifying the configuration file it should use. For example:

```
$ ./kea-ctrl-agent -c /usr/local/etc/kea/kea-ctrl-agent.conf
```

It can be started by keactrl as well (see *Managing Kea with keactrl*).

7.5 Connecting to the Control Agent

For an example of a tool that can take advantage of the RESTful API, see *The Kea Shell*.

CHAPTER

EIGHT

THE DHCPV4 SERVER

8.1 Starting and Stopping the DHCPv4 Server

It is recommended that the Kea DHCPv4 server be started and stopped using `keactrl` (described in *Managing Kea with keactrl*); however, it is also possible to run the server directly. It accepts the following command-line switches:

- `-c file` - specifies the configuration file. This is the only mandatory switch.
- `-d` - specifies whether the server logging should be switched to debug/verbose mode. In verbose mode, the logging severity and debuglevel specified in the configuration file are ignored; "debug" severity and the maximum debuglevel (99) are assumed. The flag is convenient for temporarily switching the server into maximum verbosity, e.g. when debugging.
- `-p server-port` - specifies the local UDP port on which the server will listen. This is only useful during testing, as a DHCPv4 server listening on ports other than the standard ones will not be able to handle regular DHCPv4 queries.
- `-P client-port` - specifies the remote UDP port to which the server will send all responses. This is only useful during testing, as a DHCPv4 server sending responses to ports other than the standard ones will not be able to handle regular DHCPv4 queries.
- `-t file` - specifies a configuration file to be tested. Kea-dhcp4 will load it, check it, and exit. During the test, log messages are printed to standard output and error messages to standard error. The result of the test is reported through the exit code (0 = configuration looks ok, 1 = error encountered). The check is not comprehensive; certain checks are possible only when running the server.
- `-v` - displays the Kea version and exits.
- `-V` - displays the Kea extended version with additional parameters and exits. The listing includes the versions of the libraries dynamically linked to Kea.
- `-W` - displays the Kea configuration report and exits. The report is a copy of the `config.report` file produced by `./configure`; it is embedded in the executable binary.

On startup, the server will detect available network interfaces and will attempt to open UDP sockets on all interfaces mentioned in the configuration file. Since the DHCPv4 server opens privileged ports, it requires root access. This daemon must be run as root.

During startup, the server will attempt to create a PID file of the form: [runstatedir]/kea/[conf name].kea-dhcp4.pid where:

- `runstatedir`: The value as passed into the build configure script; it defaults to "/usr/local/var/run". Note that this value may be overridden at runtime by setting the environment variable KEA_PIDFILE_DIR, although this is intended primarily for testing purposes.
- `conf name`: The configuration file name used to start the server, minus all preceding paths and the file extension. For example, given a pathname of "/usr/local/etc/kea/myconf.txt", the portion used would be "myconf".

If the file already exists and contains the PID of a live process, the server will issue a DHCP4_ALREADY_RUNNING log message and exit. It is possible, though unlikely, that the file is a remnant of a system crash and the process to which the PID belongs is unrelated to Kea. In such a case it would be necessary to manually delete the PID file.

The server can be stopped using the `kill` command. When running in a console, the server can also be shut down by pressing ctrl-c. It detects the key combination and shuts down gracefully.

8.2 DHCPv4 Server Configuration

8.2.1 Introduction

This section explains how to configure the DHCPv4 server using a configuration file. Before DHCPv4 is started, its configuration file must be created. The basic configuration is as follows:

```
{
# DHCPv4 configuration starts on the next line
"Dhcp4": {

# First we set up global values
    "valid-lifetime": 4000,
    "renew-timer": 1000,
    "rebind-timer": 2000,

# Next we set up the interfaces to be used by the server.
    "interfaces-config": {
        "interfaces": [ "eth0" ]
    },

# And we specify the type of lease database
    "lease-database": {
        "type": "memfile",
        "persist": true,
        "name": "/var/lib/kea/dhcp4.leases"
    },

# Finally, we list the subnets from which we will be leasing addresses.
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [
                {
                     "pool": "192.0.2.1 - 192.0.2.200"
                }
            ]
        }
    ]
# DHCPv4 configuration ends with the next line
}

}
```

The following paragraphs provide a brief overview of the parameters in the above example, along with their format. Subsequent sections of this chapter go into much greater detail for these and other parameters.

The lines starting with a hash (#) are comments and are ignored by the server; they do not impact its operation in any way.

The configuration starts in the first line with the initial opening curly bracket (or brace). Each configuration must contain an object specifying the configuration of the Kea module using it. In the example above this object is called `Dhcp4`.

Note: In the current Kea release it is possible to specify configurations of multiple modules within a single configuration file, but this is not recommended and support for it was removed in 1.7.10 release, including the `Logging` object: its previous content, the list of loggers, must now be inside the `Dhcp4` object.

The Dhcp4 configuration starts with the `"Dhcp4": {` line and ends with the corresponding closing brace (in the above example, the brace after the last comment). Everything defined between those lines is considered to be the Dhcp4 configuration.

In general, the order in which those parameters appear does not matter, but there are two caveats. The first one is to remember that the configuration file must be well-formed JSON. That means that the parameters for any given scope must be separated by a comma, and there must not be a comma after the last parameter. When reordering a configuration file, keep in mind that moving a parameter to or from the last position in a given scope may also require moving the comma. The second caveat is that it is uncommon — although legal JSON — to repeat the same parameter multiple times. If that happens, the last occurrence of a given parameter in a given scope is used, while all previous instances are ignored. This is unlikely to cause any confusion as there are no real-life reasons to keep multiple copies of the same parameter in the configuration file.

The first few DHCPv4 configuration elements define some global parameters. `valid-lifetime` defines how long the addresses (leases) given out by the server are valid. If nothing changes, a client that got an address is allowed to use it for 4000 seconds. (Note that integer numbers are specified as is, without any quotes around them.) `renew-timer` and `rebind-timer` are values (also in seconds) that define T1 and T2 timers that govern when the client will begin the renewal and rebind procedures.

Note: Beginning with Kea 1.6.0 the lease valid lifetime is extended from a single value to a triplet with minimum, default and maximum values using `min-valid-lifetime`, `valid-lifetime` and `max-valid-lifetime`. When the client does not specify a lifetime the default value is used, when it specifies using a DHCP option code 51 this value is used if it is not less than the minimum (in this case the minimum is returned) or greater than the maximum (in this case the maximum is used).

Note: Both `renew-timer` and `rebind-timer` are optional. The server will only send `rebind-timer` to the client, via DHCPv4 option code 59, if it is less than `valid-lifetime`; and it will only send `renew-timer`, via DHCPv4 option code 58, if it is less than `rebind-timer` (or `valid-lifetime` if `rebind-timer` was not specified). In their absence, the client should select values for T1 and T2 timers according to RFC 2131. See section *Sending T1 (Option 58) and T2 (Option 59)* for more details on generating T1 and T2.

The `interfaces-config` map specifies the server configuration concerning the network interfaces on which the server should listen to the DHCP messages. The `interfaces` parameter specifies a list of network interfaces on which the server should listen. Lists are opened and closed with square brackets, with elements separated by commas. To listen on two interfaces, the `interfaces-config` command should look like this:

```
"interfaces-config": {
    "interfaces": [ "eth0", "eth1" ]
},
```

The next couple of lines define the lease database, the place where the server stores its lease information. This particular example tells the server to use `memfile`, which is the simplest (and fastest) database backend. It uses an in-memory database and stores leases on disk in a CSV (comma-separated values) file. This is a very simple configuration; usually the lease database configuration is more extensive and contains additional parameters. Note that `lease-database` is an object and opens up a new scope, using an opening brace. Its parameters (just one in this example: `type`) follow. If there were more than one, they would be separated by commas. This scope is closed with a closing brace. As more parameters for the Dhcp4 definition follow, a trailing comma is present.

Finally, we need to define a list of IPv4 subnets. This is the most important DHCPv4 configuration structure, as the server uses that information to process clients' requests. It defines all subnets from which the server is expected to receive DHCP requests. The subnets are specified with the `subnet4` parameter. It is a list, so it starts and ends with square brackets. Each subnet definition in the list has several attributes associated with it, so it is a structure and is opened and closed with braces. At a minimum, a subnet definition has to have at least two parameters: `subnet` (which defines the whole subnet) and `pools` (which is a list of dynamically allocated pools that are governed by the DHCP server).

The example contains a single subnet. If more than one were defined, additional elements in the `subnet4` parameter would be specified and separated by commas. For example, to define three subnets, the following syntax would be used:

```
"subnet4": [
    {
        "pools": [ { "pool":  "192.0.2.1 - 192.0.2.200" } ],
        "subnet": "192.0.2.0/24"
    },
    {
        "pools": [ { "pool": "192.0.3.100 - 192.0.3.200" } ],
        "subnet": "192.0.3.0/24"
    },
    {
        "pools": [ { "pool": "192.0.4.1 - 192.0.4.254" } ],
        "subnet": "192.0.4.0/24"
    }
]
```

Note that indentation is optional and is used for aesthetic purposes only. In some cases it may be preferable to use more compact notation.

After all the parameters have been specified, we have two contexts open: global and Dhcp4; thus, we need two closing curly brackets to close them.

8.2.2 Lease Storage

All leases issued by the server are stored in the lease database. Currently there are four database backends available: memfile (which is the default backend), MySQL, PostgreSQL, and Cassandra.

Memfile - Basic Storage for Leases

The server is able to store lease data in different repositories. Larger deployments may elect to store leases in a database. *Lease Database Configuration* describes this option. In typical smaller deployments, though, the server will store lease information in a CSV file rather than a database. As well as requiring less administration, an advantage of using a file for storage is that it eliminates a dependency on third-party database software.

The configuration of the file backend (memfile) is controlled through the Dhcp4/lease-database parameters. The `type` parameter is mandatory and it specifies which storage for leases the server should use. The value of `"memfile"` indicates that the file should be used as the storage. The following list gives additional optional parameters that can be used to configure the memfile backend.

- `persist`: controls whether the new leases and updates to existing leases are written to the file. It is strongly recommended that the value of this parameter be set to `true` at all times during the server's normal operation.

Not writing leases to disk means that if a server is restarted (e.g. after a power failure), it will not know which addresses have been assigned. As a result, it may assign new clients addresses that are already in use. The value of `false` is mostly useful for performance-testing purposes. The default value of the `persist` parameter is `true`, which enables writing lease updates to the lease file.

- `name`: specifies an absolute location of the lease file in which new leases and lease updates will be recorded. The default value for this parameter is `"[kea-install-dir]/var/lib/kea/kea-leases4.csv"`.
- `lfc-interval`: specifies the interval, in seconds, at which the server will perform a lease file cleanup (LFC). This removes redundant (historical) information from the lease file and effectively reduces the lease file size. The cleanup process is described in more detail later in this section. The default value of the `lfc-interval` is `3600`. A value of 0 disables the LFC.
- `max-row-errors`: when the server loads a lease file, it is processed row by row, each row contaning a single lease. If a row is flawed and cannot be processed correctly the server will log it, discard the row, and go on to the next row. This parameter can be used to set a limit on the number of such discards that may occur after which the server will abandon the effort and exit. The default value of 0 disables the limit and allows the server to process the entire file, regardless of how many rows are discarded.

```
"Dhcp4": {
    "lease-database": {
        "type": "memfile",
        "persist": true,
        "name": "/tmp/kea-leases4.csv",
        "lfc-interval": 1800,
        "max-row-errors": 100
    }
}
```

This configuration selects the `/tmp/kea-leases4.csv` as the storage for lease information and enables persistence (writing lease updates to this file). It also configures the backend to perform a periodic cleanup of the lease file every 30 minutes and sets the maximum number of row errors to 100.

It is important to know how the lease file contents are organized to understand why the periodic lease file cleanup is needed. Every time the server updates a lease or creates a new lease for the client, the new lease information must be recorded in the lease file. For performance reasons, the server does not update the existing client's lease in the file, as this would potentially require rewriting the entire file. Instead, it simply appends the new lease information to the end of the file; the previous lease entries for the client are not removed. When the server loads leases from the lease file, e.g. at the server startup, it assumes that the latest lease entry for the client is the valid one. The previous entries are discarded, meaning that the server can re-construct the accurate information about the leases even though there may be many lease entries for each client. However, storing many entries for each client results in a bloated lease file and impairs the performance of the server's startup and reconfiguration, as it needs to process a larger number of lease entries.

Lease file cleanup (LFC) removes all previous entries for each client and leaves only the latest ones. The interval at which the cleanup is performed is configurable, and it should be selected according to the frequency of lease renewals initiated by the clients. The more frequent the renewals, the smaller the value of `lfc-interval` should be. Note, however, that the LFC takes time and thus it is possible (although unlikely) that, if the `lfc-interval` is too short, a new cleanup may be started while the previous one is still running. The server would recover from this by skipping the new cleanup when it detected that the previous cleanup was still in progress. But it implies that the actual cleanups will be triggered more rarely than configured. Moreover, triggering a new cleanup adds overhead to the server, which will not be able to respond to new requests for a short period of time when the new cleanup process is spawned. Therefore, it is recommended that the `lfc-interval` value be selected in a way that allows the LFC to complete the cleanup before a new cleanup is triggered.

Lease file cleanup is performed by a separate process (in the background) to avoid a performance impact on the server process. To avoid conflicts between two processes both using the same lease files, the LFC process starts with Kea opening a new lease file; the actual LFC process operates on the lease file that is no longer used by the server. There are also other files created as a side effect of the lease file cleanup. The detailed description of the LFC process is located later in this Kea Administrator's Reference Manual: *The LFC Process*.

Lease Database Configuration

Note: Lease database access information must be configured for the DHCPv4 server, even if it has already been configured for the DHCPv6 server. The servers store their information independently, so each server can use a separate database or both servers can use the same database.

Lease database configuration is controlled through the Dhcp4/lease-database parameters. The database type must be set to "memfile", "mysql", "postgresql", or "cql", e.g.:

```
"Dhcp4": { "lease-database": { "type": "mysql", ... }, ... }
```

Next, the name of the database to hold the leases must be set; this is the name used when the database was created (see *First-Time Creation of the MySQL Database*, *First-Time Creation of the PostgreSQL Database*, or *First-Time Creation of the Cassandra Database*).

```
"Dhcp4": { "lease-database": { "name": "database-name" , ... }, ... }
```

For Cassandra:

```
"Dhcp4": { "lease-database": { "keyspace": "database-name" , ... }, ... }
```

If the database is located on a different system from the DHCPv4 server, the database host name must also be specified:

```
"Dhcp4": { "lease-database": { "host": "remote-host-name", ... }, ... }
```

(It should be noted that this configuration may have a severe impact on server performance.)

Normally, the database will be on the same machine as the DHCPv4 server. In this case, set the value to the empty string:

```
"Dhcp4": { "lease-database": { "host" : "", ... }, ... }
```

Should the database use a port other than the default, it may be specified as well:

```
"Dhcp4": { "lease-database": { "port" : 12345, ... }, ... }
```

Should the database be located on a different system, the administrator may need to specify a longer interval for the connection timeout:

```
"Dhcp4": { "lease-database": { "connect-timeout" : timeout-in-seconds, ... }, ... }
```

The default value of five seconds should be more than adequate for local connections. If a timeout is given, though, it should be an integer greater than zero.

The maximum number of times the server will automatically attempt to reconnect to the lease database after connectivity has been lost may be specified:

```
"Dhcp4": { "lease-database": { "max-reconnect-tries" : number-of-tries, ... }, ... }
```

If the server is unable to reconnect to the database after making the maximum number of attempts, the server will exit. A value of zero (the default) disables automatic recovery and the server will exit immediately upon detecting a loss of connectivity (MySQL and PostgreSQL only). For Cassandra, Kea uses an interface that connects to all nodes in a cluster at the same time. Any connectivity issues should be handled by internal Cassandra mechanisms.

The number of milliseconds the server will wait between attempts to reconnect to the lease database after connectivity has been lost may also be specified:

```
"Dhcp4": { "lease-database": { "reconnect-wait-time" : number-of-milliseconds, ... },␣
↪... }
```

The default value for MySQL and PostgreSQL is 0, which disables automatic recovery and causes the server to exit immediately upon detecting the loss of connectivity. The default value for Cassandra is 2000 ms.

Note: Automatic reconnection to database backends is configured individually per backend. This allows users to tailor the recovery parameters to each backend they use. We do suggest that users enable it either for all backends or none, so behavior is consistent. Losing connectivity to a backend for which reconnect is disabled will result in the server shutting itself down. This includes cases when the lease database backend and the hosts database backend are connected to the same database instance.

Note: Note that the host parameter is used by the MySQL and PostgreSQL backends. Cassandra has a concept of contact points that can be used to contact the cluster, instead of a single IP or hostname. It takes a list of comma-separated IP addresses, which may be specified as:

```
"Dhcp4": { "lease-database": { "contact-points" : "192.0.2.1,192.0.2.2", ... }, ... }
```

Finally, the credentials of the account under which the server will access the database should be set:

```
"Dhcp4": { "lease-database": { "user": "user-name",
                               "password": "password",
                              ... },
           ... }
```

If there is no password to the account, set the password to the empty string “”. (This is also the default.)

Cassandra-Specific Parameters

The Cassandra backend is configured slightly differently. Cassandra has a concept of contact points that can be used to contact the cluster, instead of a single IP or hostname. It takes a list of comma-separated IP addresses, which may be specified as:

```
"Dhcp4": {
    "lease-database": {
        "type": "cql",
        "contact-points": "ip-address1, ip-address2 [,...]",
        ...
    },
    ...
}
```

Cassandra also supports a number of optional parameters:

- `reconnect-wait-time` - governs how long Kea waits before attempting to reconnect. Expressed in milliseconds. The default is 2000 [ms].

- `connect-timeout` - sets the timeout for connecting to a node. Expressed in milliseconds. The default is 5000 [ms].
- `request-timeout` - sets the timeout for waiting for a response from a node. Expressed in milliseconds. The default is 12000 [ms].
- `tcp-keepalive` - governs the TCP keep-alive mechanism. Expressed in seconds of delay. If the parameter is not present, the mechanism is disabled.
- `tcp-nodelay` - enables/disables Nagle's algorithm on connections. The default is true.
- `consistency` - configures consistency level. The default is "quorum". Supported values: any, one, two, three, quorum, all, local-quorum, each-quorum, serial, local-serial, local-one. See Cassandra consistency for more details.
- `serial-consistency` - configures serial consistency level which manages lightweight transaction isolation. The default is "serial". Supported values: any, one, two, three, quorum, all, local-quorum, each-quorum, serial, local-serial, local-one. See Cassandra serial consistency for more details.

For example, a complex Cassandra configuration with most parameters specified could look as follows:

```
"Dhcp4": {
  "lease-database": {
      "type": "cql",
      "keyspace": "keatest",
      "contact-points": "192.0.2.1, 192.0.2.2, 192.0.2.3",
      "port": 9042,
      "reconnect-wait-time": 2000,
      "connect-timeout": 5000,
      "request-timeout": 12000,
      "tcp-keepalive": 1,
      "tcp-nodelay": true
    },
    ...
}
```

Similar parameters can be specified for the hosts database.

8.2.3 Hosts Storage

Kea is also able to store information about host reservations in the database. The hosts database configuration uses the same syntax as the lease database. In fact, a Kea server opens independent connections for each purpose, be it lease or hosts information. This arrangement gives the most flexibility. Kea can keep leases and host reservations separately, but can also point to the same database. Currently the supported hosts database types are MySQL, PostgreSQL, and Cassandra.

Please note that usage of hosts storage is optional. A user can define all host reservations in the configuration file, and that is the recommended way if the number of reservations is small. However, when the number of reservations grows, it is more convenient to use host storage. Please note that both storage methods (configuration file and one of the supported databases) can be used together. If hosts are defined in both places, the definitions from the configuration file are checked first and external storage is checked later, if necessary.

In fact, host information can be placed in multiple stores. Operations are performed on the stores in the order they are defined in the configuration file, although this leads to a restriction in ordering in the case of a host reservation addition; read-only stores must be configured after a (required) read-write store, or the addition will fail.

DHCPv4 Hosts Database Configuration

Hosts database configuration is controlled through the Dhcp4/hosts-database parameters. If enabled, the type of database must be set to “mysql” or “postgresql”.

```
"Dhcp4": { "hosts-database": { "type": "mysql", ... }, ... }
```

Next, the name of the database to hold the reservations must be set; this is the name used when the lease database was created (see *Supported Backends* for instructions on how to set up the desired database type):

```
"Dhcp4": { "hosts-database": { "name": "database-name" , ... }, ... }
```

If the database is located on a different system than the DHCPv4 server, the database host name must also be specified:

```
"Dhcp4": { "hosts-database": { "host": remote-host-name, ... }, ... }
```

(Again, it should be noted that this configuration may have a severe impact on server performance.)

Normally, the database will be on the same machine as the DHCPv4 server. In this case, set the value to the empty string:

```
"Dhcp4": { "hosts-database": { "host" : "", ... }, ... }
```

Should the database use a port different than the default, it may be specified as well:

```
"Dhcp4": { "hosts-database": { "port" : 12345, ... }, ... }
```

The maximum number of times the server will automatically attempt to reconnect to the host database after connectivity has been lost may be specified:

```
"Dhcp4": { "hosts-database": { "max-reconnect-tries" : number-of-tries, ... }, ... }
```

If the server is unable to reconnect to the database after making the maximum number of attempts, the server will exit. A value of zero (the default) disables automatic recovery and the server will exit immediately upon detecting a loss of connectivity (MySQL and PostgreSQL only).

The number of milliseconds the server will wait between attempts to reconnect to the host database after connectivity has been lost may also be specified:

```
"Dhcp4": { "hosts-database": { "reconnect-wait-time" : number-of-milliseconds, ... },␣
↪... }
```

The default value for MySQL and PostgreSQL is 0, which disables automatic recovery and causes the server to exit immediately upon detecting the loss of connectivity. The default value for Cassandra is 2000 ms.

Note: Automatic reconnection to database backends is configured individually per backend. This allows users to tailor the recovery parameters to each backend they use. We do suggest that users enable it either for all backends or none, so behavior is consistent. Losing connectivity to a backend for which reconnect is disabled will result in the server shutting itself down. This includes cases when the lease database backend and the hosts database backend are connected to the same database instance.

Finally, the credentials of the account under which the server will access the database should be set:

```
"Dhcp4": { "hosts-database": { "user": "user-name",
                               "password": "password",
                              ... },
           ... }
```

If there is no password to the account, set the password to the empty string “”. (This is also the default.)

The multiple storage extension uses a similar syntax; a configuration is placed into a “hosts-databases” list instead of into a “hosts-database” entry, as in:

```
"Dhcp4": { "hosts-databases": [ { "type": "mysql", ... }, ... ], ... }
```

For additional Cassandra-specific parameters, see *Cassandra-Specific Parameters*.

Using Read-Only Databases for Host Reservations with DHCPv4

In some deployments the database user whose name is specified in the database backend configuration may not have write privileges to the database. This is often required by the policy within a given network to secure the data from being unintentionally modified. In many cases administrators have deployed inventory databases, which contain substantially more information about the hosts than just the static reservations assigned to them. The inventory database can be used to create a view of a Kea hosts database and such a view is often read-only.

Kea host database backends operate with an implicit configuration to both read from and write to the database. If the database user does not have write access to the host database, the backend will fail to start and the server will refuse to start (or reconfigure). However, if access to a read-only host database is required for retrieving reservations for clients and/or assigning specific addresses and options, it is possible to explicitly configure Kea to start in “read-only” mode. This is controlled by the `readonly` boolean parameter as follows:

```
"Dhcp4": { "hosts-database": { "readonly": true, ... }, ... }
```

Setting this parameter to `false` configures the database backend to operate in “read-write” mode, which is also the default configuration if the parameter is not specified.

Note: The `readonly` parameter is currently only supported for MySQL and PostgreSQL databases.

8.2.4 Interface Configuration

The DHCPv4 server must be configured to listen on specific network interfaces. The simplest network interface configuration tells the server to listen on all available interfaces:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "*" ]
    }
    ...
},
```

The asterisk plays the role of a wildcard and means “listen on all interfaces.” However, it is usually a good idea to explicitly specify interface names:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "eth1", "eth3" ]
    },
    ...
}
```

It is possible to use a wildcard interface name (asterisk) concurrently with explicit interface names:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "eth1", "eth3", "*" ]
    },
    ...
}
```

It is anticipated that this form of usage will only be used when it is desired to temporarily override a list of interface names and listen on all interfaces.

Some deployments of DHCP servers require that the servers listen on interfaces with multiple IPv4 addresses configured. In these situations, the address to use can be selected by appending an IPv4 address to the interface name in the following manner:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "eth1/10.0.0.1", "eth3/192.0.2.3" ]
    },
    ...
}
```

Should the server be required to listen on multiple IPv4 addresses assigned to the same interface, multiple addresses can be specified for an interface as in the example below:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "eth1/10.0.0.1", "eth1/10.0.0.2" ]
    },
    ...
}
```

Alternatively, if the server should listen on all addresses for the particular interface, an interface name without any address should be specified.

Kea supports responding to directly connected clients which don’t have an address configured. This requires the server to inject the hardware address of the destination into the data link layer of the packet being sent to the client. The DHCPv4 server uses raw sockets to achieve this, and builds the entire IP/UDP stack for the outgoing packets. The downside of raw socket use, however, is that incoming and outgoing packets bypass the firewalls (e.g. iptables).

Handling traffic on multiple IPv4 addresses assigned to the same interface can be a challenge, as raw sockets are bound to the interface. When the DHCP server is configured to use the raw socket on an interface to receive DHCP traffic, advanced packet filtering techniques (e.g. the BPF) must be used to receive unicast traffic on the desired addresses assigned to the interface. Whether clients use the raw socket or the UDP socket depends on whether they are directly connected (raw socket) or relayed (either raw or UDP socket).

Therefore, in deployments where the server does not need to provision the directly connected clients and only receives the unicast packets from the relay agents, the DHCP server should be configured to use UDP sockets instead of raw sockets. The following configuration demonstrates how this can be achieved:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "eth1", "eth3" ],
        "dhcp-socket-type": "udp"
    },
    ...
}
```

The `dhcp-socket-type` specifies that the IP/UDP sockets will be opened on all interfaces on which the server listens, i.e. “eth1” and “eth3” in our case. If `dhcp-socket-type` is set to `raw`, it configures the server to use raw sockets instead. If the `dhcp-socket-type` value is not specified, the default value `raw` is used.

Using UDP sockets automatically disables the reception of broadcast packets from directly connected clients. This effectively means that UDP sockets can be used for relayed traffic only. When using raw sockets, both the traffic from the directly connected clients and the relayed traffic are handled. Caution should be taken when configuring the server to open multiple raw sockets on the interface with several IPv4 addresses assigned. If the directly connected client sends the message to the broadcast address, all sockets on this link will receive this message and multiple responses will be sent to the client. Therefore, the configuration with multiple IPv4 addresses assigned to the interface should not be used when the directly connected clients are operating on that link. To use a single address on such interface, the “interface-name/address” notation should be used.

Note: Specifying the value `raw` as the socket type doesn’t guarantee that the raw sockets will be used! The use of raw sockets to handle the traffic from the directly connected clients is currently supported on Linux and BSD systems only. If the raw sockets are not supported on the particular OS in use, the server will issue a warning and fall back to using IP/UDP sockets.

In a typical environment, the DHCP server is expected to send back a response on the same network interface on which the query was received. This is the default behavior. However, in some deployments it is desired that the outbound (response) packets will be sent as regular traffic and the outbound interface will be determined by the routing tables. This kind of asymmetric traffic is uncommon, but valid. Kea supports a parameter called `outbound-interface` that controls this behavior. It supports two values; the first one, `same-as-inbound`, tells Kea to send back the response on the same interface where the query packet was received. This is the default behavior. The second one, `use-routing`, tells Kea to send regular UDP packets and let the kernel’s routing table determine the most appropriate interface. This only works when `dhcp-socket-type` is set to `udp`. An example configuration looks as follows:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "eth1", "eth3" ],
        "dhcp-socket-type": "udp",
        "outbound-interface": "use-routing"
    },
    ...
}
```

Interfaces are re-detected at each reconfiguration. This behavior can be disabled by setting the `re-detect` value to `false`, for instance:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "eth1", "eth3" ],
        "re-detect": false
    },
    ...
}
```

Note that interfaces are not re-detected during `config-test`.

Usually loopback interfaces (e.g. the “lo” or “lo0” interface) may not be configured, but if a loopback interface is explicitely configured and IP/UDP sockets are specified, the loopback interface is accepted.

For example, it can be used to run Kea in a FreeBSD jail having only a loopback interface, to service a relayed DHCP request:

```
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "lo0" ],
        "dhcp-socket-type": "udp"
    },
    ...
}
```

8.2.5 Issues with Unicast Responses to DHCPINFORM

The use of UDP sockets has certain benefits in deployments where the server receives only relayed traffic; these benefits are mentioned in *Interface Configuration*. From the administrator's perspective it is often desirable to configure the system's firewall to filter out unwanted traffic, and the use of UDP sockets facilitates this. However, the administrator must also be aware of the implications related to filtering certain types of traffic, as it may impair the DHCP server's operation.

In this section we are focusing on the case when the server receives the DHCPINFORM message from the client via a relay. According to RFC 2131, the server should unicast the DHCPACK response to the address carried in the "ciaddr" field. When the UDP socket is in use, the DHCP server relies on the low-level functions of an operating system to build the data link, IP, and UDP layers of the outgoing message. Typically, the OS will first use ARP to obtain the client's link-layer address to be inserted into the frame's header, if the address is not cached from a previous transaction that the client had with the server. When the ARP exchange is successful, the DHCP message can be unicast to the client, using the obtained address.

Some system administrators block ARP messages in their network, which causes issues for the server when it responds to the DHCPINFORM messages because the server is unable to send the DHCPACK if the preceding ARP communication fails. Since the OS is entirely responsible for the ARP communication and then sending the DHCP packet over the wire, the DHCP server has no means to determine that the ARP exchange failed and the DHCP response message was dropped. Thus, the server does not log any error messages when the outgoing DHCP response is dropped. At the same time, all hooks pertaining to the packet-sending operation will be called, even though the message never reaches its destination.

Note that the issue described in this section is not observed when the raw sockets are in use, because, in this case, the DHCP server builds all the layers of the outgoing message on its own and does not use ARP. Instead, it inserts the value carried in the "chaddr" field of the DHCPINFORM message into the link layer.

Server administrators willing to support DHCPINFORM messages via relays should not block ARP traffic in their networks or should use raw sockets instead of UDP sockets.

8.2.6 IPv4 Subnet Identifier

The subnet identifier is a unique number associated with a particular subnet. In principle, it is used to associate clients' leases with their respective subnets. When a subnet identifier is not specified for a subnet being configured, it will be automatically assigned by the configuration mechanism. The identifiers are assigned from 1 and are monotonically increased for each subsequent subnet: 1, 2, 3 ....

If there are multiple subnets configured with auto-generated identifiers and one of them is removed, the subnet identifiers may be renumbered. For example: if there are four subnets and the third is removed, the last subnet will be assigned the identifier that the third subnet had before removal. As a result, the leases stored in the lease database for subnet 3 are now associated with subnet 4, something that may have unexpected consequences. The only remedy for this issue at present is to manually specify a unique identifier for each subnet.

Note: Subnet IDs must be greater than zero and less than 4294967295.

The following configuration will assign the specified subnet identifier to a newly configured subnet:

```
"Dhcp4": {
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "id": 1024,
            ...
        }
    ]
}
```

This identifier will not change for this subnet unless the "id" parameter is removed or set to 0. The value of 0 forces auto-generation of the subnet identifier.

8.2.7 IPv4 Subnet Prefix

The subnet prefix is the second way to identify a subnet. It does not need to have the address part to match the prefix length, for instance this configuration is accepted:

```
"Dhcp4": {
    "subnet4": [
        {
           "subnet": "192.0.2.1/24",
            ...
        }
    ]
}
```

Even there is another subnet with the "192.0.2.0/24" prefix: only the textual form of subnets are compared to avoid duplicates.

Note: Abuse of this feature can lead to incorrect subnet selection (see *How the DHCPv4 Server Selects a Subnet for the Client*).

8.2.8 Configuration of IPv4 Address Pools

The main role of a DHCPv4 server is address assignment. For this, the server must be configured with at least one subnet and one pool of dynamic addresses to be managed. For example, assume that the server is connected to a network segment that uses the 192.0.2.0/24 prefix. The administrator of that network decides that addresses from range 192.0.2.10 to 192.0.2.20 are going to be managed by the Dhcp4 server. Such a configuration can be achieved in the following way:

```
"Dhcp4": {
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [
                { "pool": "192.0.2.10 - 192.0.2.20" }
            ],
            ...
        }
    ]
}
```

Note that `subnet` is defined as a simple string, but the `pools` parameter is actually a list of pools; for this reason, the pool definition is enclosed in square brackets, even though only one range of addresses is specified.

Each `pool` is a structure that contains the parameters that describe a single pool. Currently there is only one parameter, `pool`, which gives the range of addresses in the pool.

It is possible to define more than one pool in a subnet; continuing the previous example, further assume that 192.0.2.64/26 should be also be managed by the server. It could be written as 192.0.2.64 to 192.0.2.127. Alternatively, it can be expressed more simply as 192.0.2.64/26. Both formats are supported by Dhcp4 and can be mixed in the pool list. For example, one could define the following pools:

```
"Dhcp4": {
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [
                { "pool": "192.0.2.10-192.0.2.20" },
                { "pool": "192.0.2.64/26" }
            ],
            ...
        }
    ],
    ...
}
```

White space in pool definitions is ignored, so spaces before and after the hyphen are optional. They can be used to improve readability.

The number of pools is not limited, but for performance reasons it is recommended to use as few as possible.

The server may be configured to serve more than one subnet:

```
"Dhcp4": {
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [ { "pool": "192.0.2.1 - 192.0.2.200" } ],
            ...
        },
        {
            "subnet": "192.0.3.0/24",
            "pools": [ { "pool": "192.0.3.100 - 192.0.3.200" } ],
            ...
        },
        {
            "subnet": "192.0.4.0/24",
            "pools": [ { "pool": "192.0.4.1 - 192.0.4.254" } ],
            ...
        }
    ]
}
```

When configuring a DHCPv4 server using prefix/length notation, please pay attention to the boundary values. When specifying that the server can use a given pool, it will also be able to allocate the first (typically a network address) and the last (typically a broadcast address) address from that pool. In the aforementioned example of pool 192.0.3.0/24, both the 192.0.3.0 and 192.0.3.255 addresses may be assigned as well. This may be invalid in some network configurations. To avoid this, use the “min-max” notation.

8.2.9 Sending T1 (Option 58) and T2 (Option 59)

According to RFC 2131, servers should send values for T1 and T2 that are 50% and 87.5% of the lease lifetime, respectively. By default, kea-dhcp4 does not send either value. It can be configured to send values that are specified explicitly or that are calculated as percentages of the lease time. The server's behavior is governed by a combination of configuration parameters, two of which have already been mentioned. To send specific, fixed values use the following two parameters:

- `renew-timer` - specifies the value of T1 in seconds.
- `rebind-timer` - specifies the value of T2 in seconds.

The server will only send T2 if it is less than the valid lease time. T1 will only be sent if: T2 is being sent and T1 is less than T2; or T2 is not being sent and T1 is less than the valid lease time.

Calculating the values is controlled by the following three parameters.

- `calculate-tee-times` - when true, T1 and T2 will be calculated as percentages of the valid lease time. It defaults to false.
- `t1-percent` - the percentage of the valid lease time to use for T1. It is expressed as a real number between 0.0 and 1.0 and must be less than t2-percent. The default value is 0.50 per RFC 2131.
- `t2-percent` - the percentage of the valid lease time to use for T2. It is expressed as a real number between 0.0 and 1.0 and must be greater than t1-percent. The default value is .875 per RFC 2131.

Note: In the event that both explicit values are specified and calculate-tee-times is true, the server will use the explicit values. Administrators with a setup where some subnets or share-networks will use explicit values and some will use calculated values must not define the explicit values at any level higher than where they will be used. Inheriting them from too high a scope, such as global, will cause them to have values at every level underneath (shared-networks and subnets), effectively disabling calculated values.

8.2.10 Standard DHCPv4 Options

One of the major features of the DHCPv4 server is the ability to provide configuration options to clients. Most of the options are sent by the server only if the client explicitly requests them using the Parameter Request List option. Those that do not require inclusion in the Parameter Request List option are commonly used options, e.g. “Domain Server”, and options which require special behavior, e.g. “Client FQDN”, which is returned to the client if the client has included this option in its message to the server.

List of standard DHCPv4 options configurable by an administrator comprises the list of the standard DHCPv4 options whose values can be configured using the configuration structures described in this section. This table excludes the options which require special processing and thus cannot be configured with fixed values. The last column of the table indicates which options can be sent by the server even when they are not requested in the Parameter Request List option, and those which are sent only when explicitly requested.

The following example shows how to configure the addresses of DNS servers, which is one of the most frequently used options. Options specified in this way are considered global and apply to all configured subnets.

```
"Dhcp4": {
    "option-data": [
        {
           "name": "domain-name-servers",
           "code": 6,
           "space": "dhcp4",
           "csv-format": true,
           "data": "192.0.2.1, 192.0.2.2"
        },
        ...
    ]
}
```

Note that only one of name or code is required; there is no need to specify both. Space has a default value of "dhcp4", so this can be skipped as well if a regular (not encapsulated) DHCPv4 option is defined. Finally, csv-format defaults to true, so it too can be skipped, unless the option value is specified as a hexadecimal string. Therefore, the above example can be simplified to:

```
"Dhcp4": {
    "option-data": [
        {
           "name": "domain-name-servers",
           "data": "192.0.2.1, 192.0.2.2"
        },
        ...
    ]
}
```

Defined options are added to the response when the client requests them at a few exceptions, which are always added. To enforce the addition of a particular option, set the always-send flag to true as in:

```
"Dhcp4": {
    "option-data": [
        {
           "name": "domain-name-servers",
           "data": "192.0.2.1, 192.0.2.2",
           "always-send": true
        },
        ...
    ]
}
```

The effect is the same as if the client added the option code in the Parameter Request List option (or its equivalent for vendor options):

```
"Dhcp4": {
    "option-data": [
        {
           "name": "domain-name-servers",
           "data": "192.0.2.1, 192.0.2.2",
           "always-send": true
        },
        ...
    ],
    "subnet4": [
        {
           "subnet": "192.0.3.0/24",
           "option-data": [
               {
                   "name": "domain-name-servers",
                   "data": "192.0.3.1, 192.0.3.2"
               },
               ...
           ],
           ...
        },
        ...
    ],
    ...
}
```

The Domain Name Servers option is always added to responses (the always-send is “sticky”), but the value is the subnet one when the client is localized in the subnet.

The `name` parameter specifies the option name. For a list of currently supported names, see *List of standard DHCPv4 options configurable by an administrator* below. The `code` parameter specifies the option code, which must match one of the values from that list. The next line specifies the option space, which must always be set to “dhcp4” as these are standard DHCPv4 options. For other option spaces, including custom option spaces, see *Nested DHCPv4 Options (Custom Option Spaces)*. The next line specifies the format in which the data will be entered; use of CSV (comma-separated values) is recommended. The sixth line gives the actual value to be sent to clients. The data parameter is specified as normal text, with values separated by commas if more than one value is allowed.

Options can also be configured as hexadecimal values. If `csv-format` is set to false, option data must be specified as a hexadecimal string. The following commands configure the domain-name-servers option for all subnets with the following addresses: 192.0.3.1 and 192.0.3.2. Note that `csv-format` is set to false.

```
"Dhcp4": {
    "option-data": [
        {
            "name": "domain-name-servers",
            "code": 6,
            "space": "dhcp4",
            "csv-format": false,
            "data": "C0 00 03 01 C0 00 03 02"
        },
        ...
    ],
    ...
}
```

Kea supports the following formats when specifying hexadecimal data:

- `Delimited octets` - one or more octets separated by either colons or spaces (‘:’ or ‘ ‘). While each octet may contain one or two digits, we strongly recommend always using two digits. Valid examples are “ab:cd:ef” and “ab cd ef”.
- `String of digits` - a continuous string of hexadecimal digits with or without a “0x” prefix. Valid examples are “0xabcdef” and “abcdef”.

Care should be taken to use proper encoding when using hexadecimal format; Kea’s ability to validate data correctness in hexadecimal is limited.

As of Kea 1.6.0, it is also possible to specify data for binary options as a single-quoted text string within double quotes as shown (note that `csv-format` must be set to false):

```
"Dhcp4": {
    "option-data": [
        {
            "name": "user-class",
            "code": 77,
            "space": "dhcp4",
            "csv-format": false,
            "data": "'convert this text to binary'"
        },
        ...
    ],
    ...
}
```

Most of the parameters in the "option-data" structure are optional and can be omitted in some circumstances, as discussed in *Unspecified Parameters for DHCPv4 Option Configuration*.

It is possible to specify or override options on a per-subnet basis. If clients connected to most subnets are expected to get the same values of a given option, administrators should use global options; it is possible to override specific values for a small number of subnets. On the other hand, if different values are used in each subnet, it does not make sense to specify global option values; rather, only subnet-specific ones should be set.

The following commands override the global DNS servers option for a particular subnet, setting a single DNS server with address 192.0.2.3:

```
"Dhcp4": {
    "subnet4": [
        {
            "option-data": [
                {
                    "name": "domain-name-servers",
                    "code": 6,
                    "space": "dhcp4",
                    "csv-format": true,
                    "data": "192.0.2.3"
                },
                ...
            ],
            ...
        },
        ...
    ],
    ...
}
```

In some cases it is useful to associate some options with an address pool from which a client is assigned a lease. Pool-specific option values override subnet-specific and global option values. The server's administrator must not try to prioritize assignment of pool-specific options by trying to order pool declarations in the server configuration.

The following configuration snippet demonstrates how to specify the DNS servers option, which will be assigned to a client only if the client obtains an address from the given pool:

```
"Dhcp4": {
    "subnet4": [
        {
            "pools": [
                {
                    "pool": "192.0.2.1 - 192.0.2.200",
                    "option-data": [
                        {
                            "name": "domain-name-servers",
                            "data": "192.0.2.3"
                         },
                         ...
                    ],
                    ...
                },
                ...
            ],
            ...
        },
        ...
    ],
    ...
}
```

Options can also be specified in class or host reservation scope. The current Kea options precedence order is (from most important): host reservation, pool, subnet, shared network, class, global.

The currently supported standard DHCPv4 options are listed in *List of standard DHCPv4 options configurable by an administrator*. “Name” and “Code” are the values that should be used as a name/code in the option-data structures. “Type” designates the format of the data; the meanings of the various types are given in *List of Standard DHCP Option Types*.

When a data field is a string and that string contains the comma (,; U+002C) character, the comma must be escaped with two backslashes (; U+005C). This double escape is required because both the routine splitting CSV data into fields and JSON use the same escape character; a single escape (,) would make the JSON invalid. For example, the string “foo,bar” must be represented as:

```
"Dhcp4": {
    "subnet4": [
        {
            "pools": [
                {
                    "option-data": [
                        {
                            "name": "boot-file-name",
                            "data": "foo\\,bar"
                        }
                    ]
                },
                ...
            ],
            ...
        },
        ...
    ],
    ...
}
```

Some options are designated as arrays, which means that more than one value is allowed in such an option. For example, the option time-servers allows the specification of more than one IPv4 address, enabling clients to obtain the addresses of multiple NTP servers.

Custom DHCPv4 Options describes the configuration syntax to create custom option definitions (formats). Creation of custom definitions for standard options is generally not permitted, even if the definition being created matches the actual option format defined in the RFCs. There is an exception to this rule for standard options for which Kea currently does not provide a definition. In order to use such options, a server administrator must create a definition as described in *Custom DHCPv4 Options* in the “dhcp4” option space. This definition should match the option format described in the relevant RFC, but the configuration mechanism will allow any option format as it currently has no means to validate it.

Table 8.1: List of standard DHCPv4 options configurable by an administrator

Name	Code	Type	Array?	Returned if not reque
time-offset	2	int32	false	false
routers	3	ipv4-address	true	true
time-servers	4	ipv4-address	true	false
name-servers	5	ipv4-address	true	false
domain-name-servers	6	ipv4-address	true	true
log-servers	7	ipv4-address	true	false
cookie-servers	8	ipv4-address	true	false
lpr-servers	9	ipv4-address	true	false
impress-servers	10	ipv4-address	true	false
resource-location-servers	11	ipv4-address	true	false
boot-size	13	uint16	false	false
merit-dump	14	string	false	false
domain-name	15	fqdn	false	true
swap-server	16	ipv4-address	false	false
root-path	17	string	false	false
extensions-path	18	string	false	false
ip-forwarding	19	boolean	false	false
non-local-source-routing	20	boolean	false	false
policy-filter	21	ipv4-address	true	false
max-dgram-reassembly	22	uint16	false	false
default-ip-ttl	23	uint8	false	false
path-mtu-aging-timeout	24	uint32	false	false
path-mtu-plateau-table	25	uint16	true	false
interface-mtu	26	uint16	false	false
all-subnets-local	27	boolean	false	false
broadcast-address	28	ipv4-address	false	false
perform-mask-discovery	29	boolean	false	false
mask-supplier	30	boolean	false	false
router-discovery	31	boolean	false	false
router-solicitation-address	32	ipv4-address	false	false
static-routes	33	ipv4-address	true	false
trailer-encapsulation	34	boolean	false	false
arp-cache-timeout	35	uint32	false	false
ieee802-3-encapsulation	36	boolean	false	false
default-tcp-ttl	37	uint8	false	false
tcp-keepalive-interval	38	uint32	false	false
tcp-keepalive-garbage	39	boolean	false	false
nis-domain	40	string	false	false
nis-servers	41	ipv4-address	true	false
ntp-servers	42	ipv4-address	true	false
vendor-encapsulated-options	43	empty	false	false
netbios-name-servers	44	ipv4-address	true	false
netbios-dd-server	45	ipv4-address	true	false
netbios-node-type	46	uint8	false	false
netbios-scope	47	string	false	false
font-servers	48	ipv4-address	true	false
x-display-manager	49	ipv4-address	true	false

Continued on next

Table 8.1 – continued from previous page

Name	Code	Type	Array?	Returned if not reque
dhcp-option-overload	52	uint8	false	false
dhcp-server-identifier	54	ipv4-address	false	true
dhcp-message	56	string	false	false
dhcp-max-message-size	57	uint16	false	false
vendor-class-identifier	60	string	false	false
nwip-domain-name	62	string	false	false
nwip-suboptions	63	binary	false	false
nisplus-domain-name	64	string	false	false
nisplus-servers	65	ipv4-address	true	false
tftp-server-name	66	string	false	false
boot-file-name	67	string	false	false
mobile-ip-home-agent	68	ipv4-address	true	false
smtp-server	69	ipv4-address	true	false
pop-server	70	ipv4-address	true	false
nntp-server	71	ipv4-address	true	false
www-server	72	ipv4-address	true	false
finger-server	73	ipv4-address	true	false
irc-server	74	ipv4-address	true	false
streettalk-server	75	ipv4-address	true	false
streettalk-directory-assistance-server	76	ipv4-address	true	false
user-class	77	binary	false	false
slp-directory-agent	78	record (boolean, ipv4-address)	true	false
slp-service-scope	79	record (boolean, string)	false	false
nds-server	85	ipv4-address	true	false
nds-tree-name	86	string	false	false
nds-context	87	string	false	false
bcms-controller-names	88	fqdn	true	false
bcms-controller-address	89	ipv4-address	true	false
client-system	93	uint16	true	false
client-ndi	94	record (uint8, uint8, uint8)	false	false
uuid-guid	97	record (uint8, binary)	false	false
uap-servers	98	string	false	false
geoconf-civic	99	binary	false	false
pcode	100	string	false	false
tcode	101	string	false	false
v6-only-preferred	108	uint32	false	false
netinfo-server-address	112	ipv4-address	true	false
netinfo-server-tag	113	string	false	false
default-url	114	string	false	false
auto-config	116	uint8	false	false
name-service-search	117	uint16	true	false
subnet-selection	118	ipv4-address	false	false
domain-search	119	fqdn	true	false
vivco-suboptions	124	binary	false	false
vivso-suboptions	125	binary	false	false
pana-agent	136	ipv4-address	true	false
v4-lost	137	fqdn	false	false
capwap-ac-v4	138	ipv4-address	true	false
sip-ua-cs-domains	141	fqdn	true	false

Continued on next

Table 8.1 – continued from previous page

Name	Code	Type	Array?	Returned if not reque
rdnss-selection	146	record (uint8, ipv4-address, ipv4-address, fqdn)	true	false
v4-portparams	159	record (uint8, psid)	false	false
v4-captive-portal	160	string	false	false
option-6rd	212	record (uint8, uint8, ipv6-address, ipv4-address)	true	false
v4-access-domain	213	fqdn	false	false

Kea supports more options than the listed above. The following list is mostly useful for readers who want to understand whether Kea is able to support certain options. The following options are returned by the Kea engine itself and in general should not be configured manually. The *subnet-mask* is calculated based on subnet definition. The *host-name* is generally governed by the DNS configuration. The *dhcp-requested-address* may be sent by the client and the server should not set it. The *dhcp-lease-time* is set by Kea automatically based on the *valid-lifetime* parameter. The *dhcp-message-type* is set by the Kea engine depending on the situation and should never be configured explicitly by sysadmin. The *dhcp-parameter-request-list* is sent by clients and should never be sent by the server. The *dhcp-renewal-time* and *dhcp-rebinding-time* are governed by *renew-timer* and *rebind-timer* parameters. The *dhcp-client-identifier* is echoed back with the value sent by the client. The *fqdn* option is part of the DDNS and D2 configuration. The *dhcp-agent-options* is being sent by the relay agent.

Table 8.2: List of standard DHCPv4 options managed by Kea on its own and not directly configurable by an administrator

Name	Code	Type	Array?	Returned if not requested?
subnet-mask	1	ipv4-address	false	true
host-name	12	string	false	false
dhcp-requested-address	50	ipv6-address	false	true
dhcp-lease-time	51	uint32	false	true
dhcp-message-type	53	string	false	true
dhcp-parameter-request-list	55	uint8	true	false
dhcp-renewal-time	58	uint32	false	true
dhcp-rebinding-time	59	uint32	false	true
dhcp-client-identifier	61	hex	false	true
fqdn	81	fqdn	false	true
dhcp-agent-options	82	empty	false	true

The following table lists all option types used in the previous two tables with a description of what values are accepted for them.

Table 8.3: List of Standard DHCP Option Types

Name	Meaning
binary	An arbitrary string of bytes, specified as a set of hexadecimal digits.
boolean	A boolean value with allowed values true or false.
empty	No value; data is carried in sub-options.
fqdn	Fully qualified domain name (e.g. www.example.com).
ipv4-address	IPv4 address in the usual dotted-decimal notation (e.g. 192.0.2.1).
ipv6-address	IPv6 address in the usual colon notation (e.g. 2001:db8::1).
ipv6-prefix	IPv6 prefix and prefix length specified using CIDR notation, e.g. 2001:db8:1::/64. This data type is used to represent an 8-bit field conveying a prefix length and the variable length prefix value.
psid	PSID and PSID length separated by a slash, e.g. 3/4 specifies PSID=3 and PSID length=4. In the wire format it is represented by an 8-bit field carrying PSID length (in this case equal to 4) and the 16-bits-long PSID value field (in this case equal to "0011000000000000b" using binary notation). Allowed values for a PSID length are 0 to 16. See RFC 7597 for details about the PSID wire representation.
record	Structured data that may be comprised of any types (except "record" and "empty"). The array flag applies to the last field only.
string	Any text. Please note that Kea will silently discard any terminating/trailing nulls from the end of 'string' options when unpacking received packets. This is in keeping with RFC 2132, Section 2.
tuple	A length encoded as an 8- (16- for DHCPv6) bit unsigned integer followed by a string of this length.
uint8	8-bit unsigned integer with allowed values 0 to 255.
uint16	16-bit unsigned integer with allowed values 0 to 65535.
uint32	32-bit unsigned integer with allowed values 0 to 4294967295.
int8	8-bit signed integer with allowed values -128 to 127.
int16	16-bit signed integer with allowed values -32768 to 32767.
int32	32-bit signed integer with allowed values -2147483648 to 2147483647.

8.2.11 Custom DHCPv4 Options

Kea supports custom (non-standard) DHCPv4 options. Assume that we want to define a new DHCPv4 option called "foo" which will have code 222 and will convey a single, unsigned, 32-bit integer value. We can define such an option by putting the following entry in the configuration file:

```
"Dhcp4": {
    "option-def": [
        {
            "name": "foo",
            "code": 222,
            "type": "uint32",
            "array": false,
            "record-types": "",
            "space": "dhcp4",
            "encapsulate": ""
        }, ...
    ],
    ...
}
```

The `false` value of the `array` parameter determines that the option does NOT comprise an array of "uint32" values but is, instead, a single value. Two other parameters have been left blank: `record-types` and `encapsulate`.

The former specifies the comma-separated list of option data fields, if the option comprises a record of data fields. The `record-types` value should be non-empty if `type` is set to "record"; otherwise it must be left blank. The latter parameter specifies the name of the option space being encapsulated by the particular option. If the particular option does not encapsulate any option space, the parameter should be left blank. Note that the `option-def` configuration statement only defines the format of an option and does not set its value(s).

The `name`, `code`, and `type` parameters are required; all others are optional. The `array` default value is `false`. The `record-types` and `encapsulate` default values are blank (i.e. ""). The default `space` is "dhcp4".

Once the new option format is defined, its value is set in the same way as for a standard option. For example, the following commands set a global value that applies to all subnets.

```
"Dhcp4": {
    "option-data": [
        {
            "name": "foo",
            "code": 222,
            "space": "dhcp4",
            "csv-format": true,
            "data": "12345"
        }, ...
    ],
    ...
}
```

New options can take more complex forms than simple use of primitives (uint8, string, ipv4-address, etc.); it is possible to define an option comprising a number of existing primitives.

For example, assume we want to define a new option that will consist of an IPv4 address, followed by an unsigned 16-bit integer, followed by a boolean value, followed by a text string. Such an option could be defined in the following way:

```
"Dhcp4": {
    "option-def": [
        {
            "name": "bar",
            "code": 223,
            "space": "dhcp4",
            "type": "record",
            "array": false,
            "record-types": "ipv4-address, uint16, boolean, string",
            "encapsulate": ""
        }, ...
    ],
    ...
}
```

The `type` is set to "record" to indicate that the option contains multiple values of different types. These types are given as a comma-separated list in the `record-types` field and should be ones from those listed in *List of Standard DHCP Option Types*.

The values of the option are set in an `option-data` statement as follows:

```
"Dhcp4": {
    "option-data": [
        {
            "name": "bar",
            "space": "dhcp4",
            "code": 223,
            "csv-format": true,
            "data": "192.0.2.100, 123, true, Hello World"
        }
    ],
    ...
}
```

`csv-format` is set to `true` to indicate that the `data` field comprises a comma-separated list of values. The values in `data` must correspond to the types set in the `record-types` field of the option definition.

When `array` is set to `true` and `type` is set to "record", the last field is an array, i.e. it can contain more than one value, as in:

```
"Dhcp4": {
    "option-def": [
        {
            "name": "bar",
            "code": 223,
            "space": "dhcp4",
            "type": "record",
            "array": true,
            "record-types": "ipv4-address, uint16",
            "encapsulate": ""
        }, ...
    ],
    ...
}
```

The new option content is one IPv4 address followed by one or more 16- bit unsigned integers.

Note: In general, boolean values are specified as `true` or `false`, without quotes. Some specific boolean parameters may also accept `"true"`, `"false"`, `0`, `1`, `"0"`, and `"1"`.

Note: Numbers can be specified in decimal or hexadecimal format. The hexadecimal format can be either plain (e.g. abcd) or prefixed with 0x (e.g. 0xabcd).

8.2.12 DHCPv4 Private Options

Options with a code between 224 and 254 are reserved for private use. They can be defined at the global scope or at the client-class local scope; this allows option definitions to be used depending on context and option data to be set accordingly. For instance, to configure an old PXEClient vendor:

```
"Dhcp4": {
    "client-classes": [
        {
            "name": "pxeclient",
            "test": "option[vendor-class-identifier].text == 'PXEClient'",
            "option-def": [
                {
                    "name": "configfile",
                    "code": 209,
                    "type": "string"
                }
            ],
            ...
        }, ...
    ],
    ...
}
```

As the Vendor-Specific Information option (code 43) has vendor-specific format, i.e. can carry either raw binary value or sub-options, this mechanism is available for this option too.

In the following example taken from a real configuration, two vendor classes use the option 43 for different and incompatible purposes:

```
"Dhcp4": {
    "option-def": [
        {
            "name": "cookie",
            "code": 1,
            "type": "string",
            "space": "APC"
        },
        {
            "name": "mtftp-ip",
            "code": 1,
            "type": "ipv4-address",
            "space": "PXE"
        },
        ...
    ],
    "client-classes": [
        {
            "name": "APC",
            "test": "(option[vendor-class-identifier].text == 'APC'",
            "option-def": [
                {
                    "name": "vendor-encapsulated-options",
                    "type": "empty",
                    "encapsulate": "APC"
                }
            ],
            "option-data": [
                {
                    "name": "cookie",
                    "space": "APC",
                    "data": "1APC"
                },
                {
                    "name": "vendor-encapsulated-options"
                },
                ...
            ],
            ...
        },
        {
            "name": "PXE",
            "test": "(option[vendor-class-identifier].text == 'PXE'",
            "option-def": [
                {
                    "name": "vendor-encapsulated-options",
                    "type": "empty",
                    "encapsulate": "PXE"
                }
            ],
            "option-data": [
                {
                    "name": "mtftp-ip",
                    "space": "PXE",
                    "data": "0.0.0.0"
                },
                {
                    "name": "vendor-encapsulated-options"
                },
                ...
            ],
            ...
        },
        ...
    ],
    ...
}
```

The definition used to decode a VSI option is:

1. The local definition of a client class the incoming packet belongs to;
2. If none, the global definition;
3. If none, the last-resort definition described in the next section, *DHCPv4 Vendor-Specific Options* (backward-compatible with previous Kea versions).

Note: This last-resort definition for the Vendor-Specific Information option (code 43) is not compatible with a raw binary value. When there are known cases where a raw binary value will be used, a client class must be defined with both a classification expression matching these cases and an option definition for the VSI option with a binary type and no encapsulation.

Note: By default, in the Vendor-Specific Information option (code 43) sub-option code 0 and 255 mean PAD and END respectively according to RFC 2132. In other words, the sub-option code values of 0 and 255 are reserved. Kea does, however, allow you to define sub-option codes from 0 to 255. If you define sub-options with codes 0 and/or 255, bytes with that value will no longer be treated as a PAD or an END, but as the sub-option code when parsing a VSI option in an incoming query.

Option 43 input processing (aka unpacking) is deferred so that it happens after classification. This means you cannot classify clients using option 43 suboptions. The definition used to unpack option 43 is determined as follows:

- If defined at the global scope this definition is used
- If defined at client class scope and the packet belongs to this class the client class definition is used
- If not defined at global scope nor in a client class to which the packet belongs, the built-in last resort definition is used. This definition only says the sub-option space is "vendor-encapsulated-options-space"

The output definition selection is a bit simpler:

- If the packet belongs to a client class which defines the option 43 use this definition
- If defined at the global scope use this definition

- Otherwise use the built-in last resot definition.

Note as they use a specific/per vendor option space the sub-options are defined at the global scope.

Note: Option definitions in client classes are allowed only for this limited option set (codes 43 and from 224 to 254), and only for DHCPv4.

8.2.13 DHCPv4 Vendor-Specific Options

Currently there are two option spaces defined for the DHCPv4 daemon: "dhcp4" (for the top-level DHCPv4 options) and "vendor-encapsulated-options-space", which is empty by default but in which options can be defined. Those options are carried in the Vendor-Specific Information option (code 43). The following examples show how to define an option "foo" with code 1 that comprises an IPv4 address, an unsigned 16-bit integer, and a string. The "foo" option is conveyed in a Vendor-Specific Information option.

The first step is to define the format of the option:

```
"Dhcp4": {
    "option-def": [
        {
            "name": "foo",
            "code": 1,
            "space": "vendor-encapsulated-options-space",
            "type": "record",
            "array": false,
            "record-types": "ipv4-address, uint16, string",
            "encapsulate": ""
        }
    ],
    ...
}
```

(Note that the option space is set to `vendor-encapsulated-options-space`.) Once the option format is defined, the next step is to define actual values for that option:

```
"Dhcp4": {
    "option-data": [
        {
            "name": "foo",
            "space": "vendor-encapsulated-options-space",
            "code": 1,
            "csv-format": true,
            "data": "192.0.2.3, 123, Hello World"
        }
    ],
    ...
}
```

We also include the Vendor-Specific Information option, the option that conveys our suboption "foo". This is required; otherwise, the option will not be included in messages sent to the client.

```
"Dhcp4": {
    "option-data": [
        {
                "name": "vendor-encapsulated-options"
            }
        ],
        ...
}
```

Alternatively, the option can be specified using its code.

```
"Dhcp4": {
    "option-data": [
        {
            "code": 43
        }
    ],
    ...
}
```

Another popular option that is often somewhat imprecisely called "vendor option" is option 125. Its proper name is vendor-independent vendor-specific information option or vivso. The idea behind those options is that each vendor has its own unique set of options with their own custom formats. The vendor is identified by a 32-bit unsigned integer called enterprise-id or vendor-id. For example, vivso with vendor-id 4491 represents DOCSIS options, and they are often seen when dealing with cable modems.

In Kea each vendor is represented by its own vendor space. Since there are hundreds of vendors and sometimes they use different option definitions for different hardware, it's impossible for Kea to support them all out of the box. Fortunately, it's easy to define support for new vendor options. Let's take an example of the Genexis home gateway. This device requires sending the vivso 125 option with a suboption 2 that contains a string with the TFTP server URL. To support such a device, three steps are needed: first, we need to define option definitions that will explain how the option is supposed to be formed. Second, we will need to define option values. Third, we will need to tell Kea when to send those specific options. This last step will be accomplished with client classification.

An example snippet of a configuration could look similar to the following:

```
{
    // First, we need to define that the suboption 2 in vivso option for
    // vendor-id 25167 has a specific format (it's a plain string in this example).
    // After this definition, we can specify values for option tftp.
    "option-def": [
    {
        // We define a short name, so the option can be referenced by name.
        // The option has code 2 and resides within vendor space 25167.
        // Its data is a plain string.
        "name": "tftp",
        "code": 2,
        "space": "vendor-25167",
        "type": "string"
    } ],

    "client-classes": [
    {
        // We now need to tell Kea how to recognize when to use vendor space 25167.
        // Usually we can use a simple expression, such as checking if the device
        // sent a vivso option with specific vendor-id, e.g. "vendor[4491].exists".
        // Unfortunately, Genexis is a bit unusual in this aspect, because it
        // doesn't send vivso. In this case we need to look into the vendor class
        // (option code 60) and see if there's a specific string that identifies
        // the device.
        "name": "cpe_genexis",
        "test": "substring(option[60].hex,0,7) == 'HMC1000'",

        // Once the device is recognized, we want to send two options:
        // the vivso option with vendor-id set to 25167, and a suboption 2.
        "option-data": [
            {
                "name": "vivso-suboptions",
                "data": "25167"
            },

            // The suboption 2 value is defined as any other option. However,
            // we want to send this suboption 2, even when the client didn't
            // explicitly request it (often there is no way to do that for
            // vendor options). Therefore we use always-send to force Kea
            // to always send this option when 25167 vendor space is involved.
            {
                "name": "tftp",
                "space": "vendor-25167",
                "data": "tftp://192.0.2.1/genexis/HMC1000.v1.3.0-R.img",
                "always-send": true
            }
        ]
    } ]
}
```

By default Kea sends back only those options that are requested by a client, unless there are protocol rules that tell the DHCP server to always send an option. This approach works nicely for most cases and avoids problems with clients refusing responses with options they don't understand. Unfortunately, this is more complex when we consider vendor options. Some vendors (such as docsis, identified by vendor option 4491) have a mechanism to request specific vendor options and Kea is able to honor those. Unfortunately, for many other vendors, such as Genexis (25167) as discussed above, Kea does not have such a mechanism, so it can't send any sub-options on its own. To solve this issue, we came up with the concept of persistent options. Kea can be told to always send options, even if the client did not request them. This can be achieved by adding `"always-send": true` to the option definition. Note that in this particular case an option is defined in vendor space 25167. With the "always-send" enabled, the option will be sent every time there is a need to deal with vendor space 25167.

Another possibility is to redefine the option; see *DHCPv4 Private Options*.

8.2.14 Nested DHCPv4 Options (Custom Option Spaces)

It is sometimes useful to define a completely new option space, such as when a user creates a new option in the standard option space ("dhcp4") and wants this option to convey sub-options. Since they are in a separate space, sub-option codes will have a separate numbering scheme and may overlap with the codes of standard options.

Note that the creation of a new option space is not required when defining sub-options for a standard option, because one is created by default if the standard option is meant to convey any sub-options (see *DHCPv4 Vendor-Specific Options*).

Assume that we want to have a DHCPv4 option called "container" with code 222 that conveys two sub-options with codes 1 and 2. First we need to define the new sub-options:

```
"Dhcp4": {
    "option-def": [
        {
            "name": "subopt1",
            "code": 1,
            "space": "isc",
            "type": "ipv4-address",
            "record-types": "",
            "array": false,
            "encapsulate": ""
        },
        {
            "name": "subopt2",
            "code": 2,
            "space": "isc",
            "type": "string",
            "record-types": "",
            "array": false,
            "encapsulate": ""
        }
    ],
    ...
}
```

Note that we have defined the options to belong to a new option space (in this case, “isc”).

The next step is to define a regular DHCPv4 option with the desired code and specify that it should include options from the new option space:

```
"Dhcp4": {
    "option-def": [
        ...,
        {
            "name": "container",
            "code": 222,
            "space": "dhcp4",
            "type": "empty",
            "array": false,
            "record-types": "",
            "encapsulate": "isc"
        }
    ],
    ...
}
```

The name of the option space in which the sub-options are defined is set in the `encapsulate` field. The `type` field is set to `empty`, to indicate that this option does not carry any data other than sub-options.

Finally, we can set values for the new options:

```
"Dhcp4": {
    "option-data": [
        {
            "name": "subopt1",
            "code": 1,
            "space": "isc",
            "data": "192.0.2.3"
        },
        }
            "name": "subopt2",
            "code": 2,
            "space": "isc",
            "data": "Hello world"
        },
        {
            "name": "container",
            "code": 222,
            "space": "dhcp4"
        }
    ],
    ...
}
```

Note that it is possible to create an option which carries some data in addition to the sub-options defined in the encapsulated option space. For example, if the "container" option from the previous example were required to carry a uint16 value as well as the sub-options, the `type` value would have to be set to "uint16" in the option definition. (Such an option would then have the following data structure: DHCP header, uint16 value, sub-options.) The value specified with the `data` parameter — which should be a valid integer enclosed in quotes, e.g. "123" — would then be assigned to the uint16 field in the "container" option.

8.2.15 Unspecified Parameters for DHCPv4 Option Configuration

In many cases it is not required to specify all parameters for an option configuration, and the default values can be used. However, it is important to understand the implications of not specifying some of them, as it may result in configuration errors. The list below explains the behavior of the server when a particular parameter is not explicitly specified:

- `name` - the server requires either an option name or an option code to identify an option. If this parameter is unspecified, the option code must be specified.
- `code` - the server requires either an option name or an option code to identify an option. This parameter may be left unspecified if the `name` parameter is specified. However, this also requires that the particular option have a definition (either as a standard option or an administrator-created definition for the option using an 'option-def' structure), as the option definition associates an option with a particular name. It is possible to configure an option for which there is no definition (unspecified option format). Configuration of such options requires the use of the option code.
- `space` - if the option space is unspecified it will default to 'dhcp4', which is an option space holding standard DHCPv4 options.
- `data` - if the option data is unspecified it defaults to an empty value. The empty value is mostly used for the options which have no payload (boolean options), but it is legal to specify empty values for some options which carry variable-length data and for which the specification allows a length of 0. For such options, the data parameter may be omitted in the configuration.
- `csv-format` - if this value is not specified, the server will assume that the option data is specified as a list of comma-separated values to be assigned to individual fields of the DHCP option.

8.2.16 Stateless Configuration of DHCPv4 Clients

The DHCPv4 server supports the stateless client configuration whereby the client has an IP address configured (e.g. using manual configuration) and only contacts the server to obtain other configuration parameters, such as addresses of DNS servers. In order to obtain the stateless configuration parameters, the client sends the DHCPINFORM message to the server with the "ciaddr" set to the address that the client is currently using. The server unicasts the DHCPACK message to the client that includes the stateless configuration ("yiaddr" not set).

The server will respond to the DHCPINFORM when the client is associated with a subnet defined in the server's configuration. An example subnet configuration will look like this:

```
"Dhcp4": {
    "subnet4": [
        {
            "subnet": "192.0.2.0/24"
            "option-data": [ {
                "name": "domain-name-servers",
                "code": 6,
                "data": "192.0.2.200,192.0.2.201",
                "csv-format": true,
                "space": "dhcp4"
            } ]
        }
    ]
}
```

This subnet specifies the single option which will be included in the DHCPACK message to the client in response to DHCPINFORM. Note that the subnet definition does not require the address pool configuration if it will be used solely for the stateless configuration.

This server will associate the subnet with the client if one of the following conditions is met:

- The DHCPINFORM is relayed and the giaddr matches the configured subnet.
- The DHCPINFORM is unicast from the client and the ciaddr matches the configured subnet.
- The DHCPINFORM is unicast from the client and the ciaddr is not set, but the source address of the IP packet matches the configured subnet.
- The DHCPINFORM is not relayed and the IP address on the interface on which the message is received matches the configured subnet.

8.2.17 Client Classification in DHCPv4

The DHCPv4 server includes support for client classification. For a deeper discussion of the classification process see *Client Classification*.

In certain cases it is useful to configure the server to differentiate between DHCP client types and treat them accordingly. Client classification can be used to modify the behavior of almost any part of the DHCP message processing. Kea currently offers client classification via private options and option 43 deferred unpacking; subnet selection; pool selection; assignment of different options; and, for cable modems, specific options for use with the TFTP server address and the boot file field.

Kea can be instructed to limit access to given subnets based on class information. This is particularly useful for cases where two types of devices share the same link and are expected to be served from two different subnets. The primary use case for such a scenario is cable networks, where there are two classes of devices: the cable modem itself, which should be handed a lease from subnet A; and all other devices behind the modem, which should get a lease from subnet B. That segregation is essential to prevent overly curious users from playing with their cable modems. For details on how to set up class restrictions on subnets, see *Configuring Subnets With Class Information*.

When subnets belong to a shared network, the classification applies to subnet selection but not to pools; that is, a pool in a subnet limited to a particular class can still be used by clients which do not belong to the class, if the pool they are expected to use is exhausted. So the limit on access based on class information is also available at the pool level; see *Configuring Pools With Class Information*, within a subnet. This is useful when segregating clients belonging to the same subnet into different address ranges.

In a similar way, a pool can be constrained to serve only known clients, i.e. clients which have a reservation, using the built-in “KNOWN” or “UNKNOWN” classes. Addresses can be assigned to registered clients without giving a different address per reservation, for instance when there are not enough available addresses. The determination whether there is a reservation for a given client is made after a subnet is selected, so it is not possible to use "KNOWN"/"UNKNOWN" classes to select a shared network or a subnet.

The process of classification is conducted in five steps. The first step is to assess an incoming packet and assign it to zero or more classes. The second step is to choose a subnet, possibly based on the class information. When the incoming packet is in the special class, "DROP", it is dropped and an debug message logged. The next step is to evaluate class expressions depending on the built-in "KNOWN"/"UNKNOWN" classes after host reservation lookup, using them for pool selection and assigning classes from host reservations. The list of required classes is then built and each class of the list has its expression evaluated; when it returns "true" the packet is added as a member of the class. The last step is to assign options, again possibly based on the class information. More complete and detailed information is available in *Client Classification*.

There are two main methods of classification. The first is automatic and relies on examining the values in the vendor class options or the existence of a host reservation. Information from these options is extracted, and a class name is constructed from it and added to the class list for the packet. The second specifies an expression that is evaluated for each packet. If the result is "true", the packet is a member of the class.

Note: Care should be taken with client classification, as it is easy for clients that do not meet class criteria to be denied all service.

Setting Fixed Fields in Classification

It is possible to specify that clients belonging to a particular class should receive packets with specific values in certain fixed fields. In particular, three fixed fields are supported: `next-server` (conveys an IPv4 address, which is set in the siaddr field), `server-hostname` (conveys a server hostname, can be up to 64 bytes long, and is sent in the sname field) and `boot-file-name` (conveys the configuration file, can be up to 128 bytes long, and is sent using the file field).

Obviously, there are many ways to assign clients to specific classes, but for PXE clients the client architecture type option (code 93) seems to be particularly suited to make the distinction. The following example checks whether the client identifies itself as a PXE device with architecture EFI x86-64, and sets several fields if it does. See Section 2.1 of RFC 4578) or the client documentation for specific values.

```
"Dhcp4": {
    "client-classes": [
        {
            "name": "ipxe_efi_x64",
            "test": "option[93].hex == 0x0009",
            "next-server": "192.0.2.254",
            "server-hostname": "hal9000",
            "boot-file-name": "/dev/null"
        },
        ...
    ],
    ...
      }
```

If there are multiple classes defined and an incoming packet is matched to multiple classes, the class that is evaluated first is used.

Note: The classes are ordered as specified in the configuration.

Using Vendor Class Information in Classification

The server checks whether an incoming packet includes the vendor class identifier option (60). If it does, the content of that option is prepended with "VENDOR_CLASS_", and it is interpreted as a class. For example, modern cable modems will send this option with value "docsis3.0" and as a result the packet will belong to class "VENDOR_CLASS_docsis3.0".

Note: Certain special actions for clients in VENDOR_CLASS_docsis3.0 can be achieved by defining VENDOR_CLASS_docsis3.0 and setting its next-server and boot-file-name values appropriately.

This example shows a configuration using an automatically generated "VENDOR_CLASS_" class. The administrator of the network has decided that addresses from range 192.0.2.10 to 192.0.2.20 are going to be managed by the Dhcp4 server and only clients belonging to the docsis3.0 client class are allowed to use that pool.

```
"Dhcp4": {
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [ { "pool": "192.0.2.10 - 192.0.2.20" } ],
            "client-class": "VENDOR_CLASS_docsis3.0"
        }
    ],
    ...
}
```

Defining and Using Custom Classes

The following example shows how to configure a class using an expression and a subnet using that class. This configuration defines the class named "Client_foo". It is comprised of all clients whose client ids (option 61) start with the string "foo". Members of this class will be given addresses from 192.0.2.10 to 192.0.2.20 and the addresses of their DNS servers set to 192.0.2.1 and 192.0.2.2.

```
"Dhcp4": {
    "client-classes": [
        {
            "name": "Client_foo",
            "test": "substring(option[61].hex,0,3) == 'foo'",
            "option-data": [
                {
                    "name": "domain-name-servers",
                    "code": 6,
                    "space": "dhcp4",
                    "csv-format": true,
                    "data": "192.0.2.1, 192.0.2.2"
                }
            ]
        },
        ...
    ],
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [ { "pool": "192.0.2.10 - 192.0.2.20" } ],
            "client-class": "Client_foo"
        },
        ...
    ],
    ...
}
```

Required Classification

In some cases it is useful to limit the scope of a class to a shared network, subnet, or pool. There are two parameters which are used to limit the scope of the class by instructing the server to evaluate test expressions when required.

The first one is the per-class `only-if-required` flag, which is false by default. When it is set to `true`, the test expression of the class is not evaluated at the reception of the incoming packet but later, and only if the class evaluation is required.

The second is `require-client-classes`, which takes a list of class names and is valid in shared-network, subnet, and pool scope. Classes in these lists are marked as required and evaluated after selection of this specific shared network/subnet/pool and before output option processing.

In this example, a class is assigned to the incoming packet when the specified subnet is used:

```
"Dhcp4": {
    "client-classes": [
       {
           "name": "Client_foo",
           "test": "member('ALL')",
           "only-if-required": true
       },
       ...
    ],
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [ { "pool": "192.0.2.10 - 192.0.2.20" } ],
            "require-client-classes": [ "Client_foo" ],
            ...
        },
        ...
    ],
    ...
}
```

Required evaluation can be used to express complex dependencies like subnet membership. It can also be used to reverse the precedence; if an option-data is set in a subnet, it takes precedence over an option-data in a class. If the option-data is moved to a required class and required in the subnet, a class evaluated earlier may take precedence.

Required evaluation is also available at the shared-network and pool levels. The order in which required classes are considered is: shared-network, subnet, and pool, i.e. in the opposite order in which option-data is processed.

8.2.18 DDNS for DHCPv4

As mentioned earlier, kea-dhcp4 can be configured to generate requests to the DHCP-DDNS server, kea-dhcp-ddns, (referred to herein as “D2”) to update DNS entries. These requests are known as Name Change Requests or NCRs. Each NCR contains the following information:

1. Whether it is a request to add (update) or remove DNS entries
2. Whether the change requests forward DNS updates (A records), reverse DNS updates (PTR records), or both

3. The Fully Qualified Domain Name (FQDN), lease address, and DHCID (information identifying the client associated with the FQDN)

Prior to Kea 1.7.1, all parameters for controlling DDNS were within the global `dhcp-ddns` section of the kea-dhcp4. Beginning with Kea 1.7.1 DDNS related parameters were split into two groups:

1. Connectivity Parameters

   These are parameters which specify where and how kea-dhcp4 connects to and communicates with D2. These parameters can only be specified within the top-level `dhcp-ddns` section in the kea-dhcp4 configuration. The connectivity parameters are listed below:

   - `enable-updates`
   - `server-ip`
   - `server-port`
   - `sender-ip`
   - `sender-port`
   - `max-queue-size`
   - `ncr-protocol`
   - `ncr-format"`

2. Behavioral Parameters

   These parameters influence behavior such as how client host names and FQDN options are handled. They have been moved out of the `dhcp-ddns` section so that they may be specified at the global, shared-network, and/or subnet levels. Furthermore, they are inherited downward from global to shared-network to subnet. In other words, if a parameter is not specified at a given level, the value for that level comes from the level above it. The behavioral parameter as follows:

   - `ddns-send-updates`
   - `ddns-override-no-update`
   - `ddns-override-client-update`
   - `ddns-replace-client-name"`
   - `ddns-generated-prefix`
   - `ddns-qualifying-suffix`
   - `hostname-char-set`
   - `hostname-char-replacement`

Note: For backward compatibility, configuration parsing will still recognize the original behavioral parameters specified in `dhcp-ddns`. It will do so by translating the parameter into its global equivalent. If a parameter is specified both globally and in `dhcp-ddns`, the latter value will be ignored. In either case, a log will be emitted explaining what has occurred. Specifying these values within `dhcp-ddns` is deprecated and support for it will be removed at some future date.

The default configuration would appear as follows:

```
"Dhcp4": {
     "dhcp-ddns": {
        // Connectivity parameters
        "enable-updates": false,
        "server-ip": "127.0.0.1",
        "server-port":53001,
        "sender-ip":"",
        "sender-port":0,
        "max-queue-size":1024,
        "ncr-protocol":"UDP",
        "ncr-format":"JSON"
    },

    // Behavioral parameters (global)
    "ddns-send-updates": true,
    "ddns-override-no-update": false,
    "ddns-override-client-update": false,
    "ddns-replace-client-name": "never",
    "ddns-generated-prefix": "myhost",
    "ddns-qualifying-suffix": "",
    "hostname-char-set": "",
    "hostname-char-replacement": ""
    ...
}
```

As of Kea 1.7.1, there are two parameters which determine if kea-dhcp4 can generate DDNS requests to D2. The existing, `dhcp-ddns:enable-updates` parameter which now only controls whether kea-dhcp4 connects to D2. And the new behavioral parameter, `ddns-send-updates`, which determines if DDNS updates are enabled at a given level (i.e global, shared-network, or subnet). The following table shows how the two parameters function together:

Table 8.4: Enabling and Disabling DDNS Updates

dhcp-ddns: enable-updates	Global ddns-send-udpates	Outcome
false (default)	false	no updates at any scope
false	true (default)	no updates at any scope
true	false	updates only at scopes with a local value of true for ddns-enable-updates
true	true	updates at all scopes except those with a local value of false for ddns-enable-updates

DHCP-DDNS Server Connectivity

For NCRs to reach the D2 server, kea-dhcp4 must be able to communicate with it. kea-dhcp4 uses the following configuration parameters to control this communication:

- `enable-updates` - As of Kea 1.7.1, this parameter only enables connectivity to kea-dhcp-ddns such that DDNS updates can be constructed and sent. It must be true for NCRs to be generated and sent to D2. It defaults to false.
- `server-ip` - the IP address on which D2 listens for requests. The default is the local loopback interface at address 127.0.0.1. Either an IPv4 or IPv6 address may be specified.
- `server-port` - the port on which D2 listens for requests. The default value is 53001.
- `sender-ip` - the IP address which kea-dhcp4 uses to send requests to D2. The default value is blank, which instructs kea-dhcp4 to select a suitable address.
- `sender-port` - the port which kea-dhcp4 uses to send requests to D2. The default value of 0 instructs kea-dhcp4 to select a suitable port.

- `max-queue-size` - the maximum number of requests allowed to queue waiting to be sent to D2. This value guards against requests accumulating uncontrollably if they are being generated faster than they can be delivered. If the number of requests queued for transmission reaches this value, DDNS updating will be turned off until the queue backlog has been sufficiently reduced. The intent is to allow the kea-dhcp4 server to continue lease operations without running the risk that its memory usage grows without limit. The default value is 1024.
- `ncr-protocol` - the socket protocol to use when sending requests to D2. Currently only UDP is supported.
- `ncr-format` - the packet format to use when sending requests to D2. Currently only JSON format is supported.

By default, kea-dhcp-ddns is assumed to be running on the same machine as kea-dhcp4, and all of the default values mentioned above should be sufficient. If, however, D2 has been configured to listen on a different address or port, these values must be altered accordingly. For example, if D2 has been configured to listen on 192.168.1.10 port 900, the following configuration is required:

```
"Dhcp4": {
    "dhcp-ddns": {
        "server-ip": "192.168.1.10",
        "server-port": 900,
        ...
    },
    ...
}
```

When Does the kea-dhcp4 Server Generate a DDNS Request?

kea-dhcp4 follows the behavior prescribed for DHCP servers in RFC 4702. It is important to keep in mind that kea-dhcp4 makes the initial decision of when and what to update and forwards that information to D2 in the form of NCRs. Carrying out the actual DNS updates and dealing with such things as conflict resolution are within the purview of D2 itself (see *The DHCP-DDNS Server*). This section describes when kea-dhcp4 will generate NCRs and the configuration parameters that can be used to influence this decision. It assumes that both the connectivity parameter, `enable-updates` and the behavioral parameter `ddns-send-updates`, are true.

In general, kea-dhcp4 will generate DDNS update requests when:

1. A new lease is granted in response to a DHCPREQUEST;
2. An existing lease is renewed but the FQDN associated with it has changed; or
3. An existing lease is released in response to a DHCPRELEASE.

In the second case, lease renewal, two DDNS requests will be issued: one request to remove entries for the previous FQDN, and a second request to add entries for the new FQDN. In the last case, a lease release, a single DDNS request to remove its entries will be made.

As for the first case, the decisions involved when granting a new lease are more complex. When a new lease is granted, kea-dhcp4 will generate a DDNS update request if the DHCPREQUEST contains either the FQDN option (code 81) or the Host Name option (code 12). If both are present, the server will use the FQDN option. By default, kea-dhcp4 will respect the FQDN N and S flags specified by the client as shown in the following table:

Table 8.5: Default FQDN Flag Behavior

Client Flags:N-S	Client Intent	Server Response	Server Flags:N-S-O
0-0	Client wants to do forward updates, server should do reverse updates	Server generates reverse-only request	1-0-0
0-1	Server should do both forward and reverse updates	Server generates request to update both directions	0-1-0
1-0	Client wants no updates done	Server does not generate a request	1-0-0

The first row in the table above represents "client delegation." Here the DHCP client states that it intends to do the forward DNS updates and the server should do the reverse updates. By default, kea-dhcp4 will honor the client's wishes and generate a DDNS request to the D2 server to update only reverse DNS data. The parameter `ddns-override-client-update` can be used to instruct the server to override client delegation requests. When this parameter is "true", kea-dhcp4 will disregard requests for client delegation and generate a DDNS request to update both forward and reverse DNS data. In this case, the N-S-O flags in the server's response to the client will be 0-1-1 respectively.

(Note that the flag combination N=1, S=1 is prohibited according to RFC 4702. If such a combination is received from the client, the packet will be dropped by kea-dhcp4.)

To override client delegation, set the following values in the configuration file:

```
"Dhcp4": {
    ...
    "ddns-override-client-update": true,
    ...
}
```

The third row in the table above describes the case in which the client requests that no DNS updates be done. The parameter, `ddns-override-no-update`, can be used to instruct the server to disregard the client's wishes. When this parameter is true, kea-dhcp4 will generate DDNS update requests to kea-dhcp-ddns even if the client requests that no updates be done. The N-S-O flags in the server's response to the client will be 0-1-1.

To override client delegation, issue the following commands:

```
"Dhcp4": {
    ...
    "ddns-override-no-update": true,
    ...
}
```

kea-dhcp4 will always generate DDNS update requests if the client request only contains the Host Name option. In addition, it will include an FQDN option in the response to the client with the FQDN N-S-O flags set to 0-1-0 respectively. The domain name portion of the FQDN option will be the name submitted to D2 in the DDNS update request.

kea-dhcp4 Name Generation for DDNS Update Requests

Each Name Change Request must of course include the fully qualified domain name whose DNS entries are to be affected. kea-dhcp4 can be configured to supply a portion or all of that name, based upon what it receives from the client in the DHCPREQUEST.

The default rules for constructing the FQDN that will be used for DNS entries are:

1. If the DHCPREQUEST contains the client FQDN option, take the candidate name from there; otherwise, take it from the Host Name option.
2. If the candidate name is a partial (i.e. unqualified) name, then add a configurable suffix to the name and use the result as the FQDN.
3. If the candidate name provided is empty, generate an FQDN using a configurable prefix and suffix.
4. If the client provides neither option, then take no DNS action.

These rules can be amended by setting the `ddns-replace-client-name` parameter, which provides the following modes of behavior:

- `never` - use the name the client sent. If the client sent no name, do not generate one. This is the default mode.
- `always` - replace the name the client sent. If the client sent no name, generate one for the client.
- `when-present` - replace the name the client sent. If the client sent no name, do not generate one.
- `when-not-present` - use the name the client sent. If the client sent no name, generate one for the client.

Note: Note that in early versions of Kea, this parameter was a boolean and permitted only values of `true` and `false`. Boolean values have been deprecated and are no longer accepted. Administrators currently using booleans must replace them with the desired mode name. A value of `true` maps to `"when-present"`, while `false` maps to `"never"`.

For example, to instruct kea-dhcp4 to always generate the FQDN for a client, set the parameter `ddns-replace-client-name` to `always` as follows:

```
"Dhcp4": {
     ...
     "ddns-replace-client-name": "always",
     ...
}
```

The prefix used in the generation of an FQDN is specified by the `generated-prefix` parameter. The default value is “myhost”. To alter its value, simply set it to the desired string:

```
"Dhcp4": {
    ...
     "ddns-generated-prefix": "another.host",
    ...
}
```

The suffix used when generating an FQDN, or when qualifying a partial name, is specified by the `ddns-qualifying-suffix` parameter. It is strongly recommended that you supply a value for qualifying prefix when DDNS updates are enabled. For obvious reasons, we cannot supply a meaningful default.

```
"Dhcp4": {
    ...
    "ddns-qualifying-suffix": "foo.example.org",
    ...
}
```

When generating a name, kea-dhcp4 will construct the name in the format:

[**ddns-generated-prefix**]-[**address-text**].[**ddns-qualifying-suffix**].

where **address-text** is simply the lease IP address converted to a hyphenated string. For example, if the lease address is 172.16.1.10, the qualifying suffix "example.com", and the default value is used for `ddns-generated-prefix`, the generated FQDN is:

myhost-172-16-1-10.example.com.

Sanitizing Client Host Name and FQDN Names

Some DHCP clients may provide values in the Host Name option (option code 12) or FQDN option (option code 81) that contain undesirable characters. It is possible to configure kea-dhcp4 to sanitize these values. The most typical use case is ensuring that only characters that are permitted by RFC 1035 be included: A-Z, a-z, 0-9, and '-'. This may be accomplished with the following two parameters:

- `hostname-char-set` - a regular expression describing the invalid character set. This can be any valid, regular expression using POSIX extended expression syntax. Embedded nuls (0x00) will always be considered an invalid character to be replaced (or omitted).
- `hostname-char-replacement` - a string of zero or more characters with which to replace each invalid character in the host name. An empty string and will cause invalid characters to be OMITTED rather than replaced.

Note: Starting with Kea 1.7.5, the default values are as follows:

- "hostname-char-set": "[^A-Za-z0-9.-]",
- "hostname-char-replacement": ""

This enables sanitizing and will omit any character that is not a letter,digit, hyphen, dot or nul.

The following configuration will replace anything other than a letter, digit, hyphen, or dot with the letter 'x':

```
"Dhcp4": {
     ...
     "hostname-char-set": "[^A-Za-z0-9.-]",
     "hostname-char-replacement": "x",
     ...
}
```

Thus, a client-supplied value of "myhost-$[123.org" would become "myhost-xx123.org". Sanitizing is performed only on the portion of the name supplied by the client, and it is performed before applying a qualifying suffix (if one is defined and needed).

Note: The following are some considerations to keep in mind: Name sanitizing is meant to catch the more common cases of invalid characters through a relatively simple character-replacement scheme. It is difficult to devise a scheme that works well in all cases, for both Host Name and FQDN options. Administrators who find they have clients with odd corner cases of character combinations that cannot be readily handled with this mechanism should consider writing a hook that can carry out sufficiently complex logic to address their needs.

If clients include domain names in the Host Name option and the administrator wants these preserved, they will need to make sure that the dot, '.', is considered a valid character by the hostname-char-set expression, such as this: "[^A-Za-z0-9.-]". This will not affect dots in FQDN Option values. When scrubbing FQDNs, dots are treated as delimiters and used to separate the option value into individual domain labels that are scrubbed and then re-assembled.

If clients are sending values that differ only by characters considered as invalid by the hostname-char-set, be aware that scrubbing them will yield identical values. In such cases, DDNS conflict rules will permit only one of them to register the name.

Finally, given the latitude clients have in the values they send, it is virtually impossible to guarantee that a combination of these two parameters will always yield a name that is valid for use in DNS. For example, using an empty value for hostname-char-replacement could yield an empty domain label within a name, if that label consists only of invalid characters.

Note: Since the 1.6.0 Kea release it is possible to specify hostname-char-set and/or hostname-char-replacement at the global scope. This allows to sanitize host names without requiring a dhcp-ddns entry. When a hostname-char parameter is defined at the global scope and in a dhcp-ddns entry the second (local) value is used.

8.2.19 Next Server (siaddr)

In some cases, clients want to obtain configuration from a TFTP server. Although there is a dedicated option for it, some devices may use the siaddr field in the DHCPv4 packet for that purpose. That specific field can be configured using the `next-server` directive. It is possible to define it in the global scope or for a given subnet only. If both are defined, the subnet value takes precedence. The value in subnet can be set to 0.0.0.0, which means that `next-server` should not be sent. It may also be set to an empty string, which means the same as if it were not defined at all; that is, use the global value.

The `server-hostname` (which conveys a server hostname, can be up to 64 bytes long, and will be sent in the sname field) and `boot-file-name` (which conveys the configuration file, can be up to 128 bytes long, and will be sent using the file field) directives are handled the same way as `next-server`.

```
"Dhcp4": {
    "next-server": "192.0.2.123",
    "boot-file-name": "/dev/null",
    ...,
    "subnet4": [
        {
            "next-server": "192.0.2.234",
            "server-hostname": "some-name.example.org",
            "boot-file-name": "bootfile.efi",
            ...
        }
    ]
}
```

8.2.20 Echoing Client-ID (RFC 6842)

The original DHCPv4 specification (RFC 2131) states that the DHCPv4 server must not send back client-id options when responding to clients. However, in some cases that result confused clients that did not have a MAC address or client-id; see RFC 6842 for details. That behavior changed with the publication of RFC 6842, which updated RFC 2131. That update states that the server must send the client-id if the client sent it. That is Kea's default behavior. However, in some cases older devices that do not support RFC 6842 may refuse to accept responses that include the client-id option. To enable backward compatibility, an optional configuration parameter has been introduced. To configure it, use the following configuration statement:

```
"Dhcp4": {
    "echo-client-id": false,
    ...
}
```

8.2.21 Using Client Identifier and Hardware Address

The DHCP server must be able to identify the client from which it receives the message and distinguish it from other clients. There are many reasons why this identification is required; the most important ones are:

- When the client contacts the server to allocate a new lease, the server must store the client identification information in the lease database as a search key.
- When the client is trying to renew or release the existing lease, the server must be able to find the existing lease entry in the database for this client, using the client identification information as a search key.
- Some configurations use static reservations for the IP addresses and other configuration information. The server's administrator uses client identification information to create these static assignments.
- In dual-stack networks there is often a need to correlate the lease information stored in DHCPv4 and DHCPv6 servers for a particular host. Using common identification information by the DHCPv4 and DHCPv6 clients allows the network administrator to achieve this correlation and better administer the network.

DHCPv4 uses two distinct identifiers which are placed by the client in the queries sent to the server and copied by the server to its responses to the client: "chaddr" and "client identifier". The former was introduced as a part of the BOOTP specification and it is also used by DHCP to carry the hardware address of the interface used to send the query to the server (MAC address for the Ethernet). The latter is carried in the Client-identifier option, introduced in RFC 2132.

RFC 2131 indicates that the server may use both of these identifiers to identify the client but the "client identifier", if present, takes precedence over "chaddr". One of the reasons for this is that "client identifier" is independent from the hardware used by the client to communicate with the server. For example, if the client obtained the lease using one network card and then the network card is moved to another host, the server will wrongly identify this host as the one which obtained the lease. Moreover, RFC 4361 gives the recommendation to use a DUID (see RFC 8415, the DHCPv6 specification) carried as a "client identifier" when dual-stack networks are in use to provide consistent identification information for the client, regardless of the type of protocol it is using. Kea adheres to these specifications, and the "client identifier" by default takes precedence over the value carried in the "chaddr" field when the server searches, creates, updates, or removes the client's lease.

When the server receives a DHCPDISCOVER or DHCPREQUEST message from the client, it will try to find out if the client already has a lease in the database; if it does, the server will hand out that lease rather than allocate a new one. Each lease in the lease database is associated with the "client identifier" and/or "chaddr". The server will first use the "client identifier" (if present) to search for the lease. If the lease is found, the server will treat this lease as belonging to the client even if the current "chaddr" and the "chaddr" associated with the lease do not match. This facilitates the scenario when the network card on the client system has been replaced and thus the new MAC address appears in the messages sent by the DHCP client. If the server fails to find the lease using the "client identifier", it will perform another lookup using the "chaddr". If this lookup returns no result, the client is considered as not having a lease and a new lease will be created.

A common problem reported by network operators is that poor client implementations do not use stable client identifiers, instead generating a new "client identifier" each time the client connects to the network. Another well-known case is when the client changes its "client identifier" during the multi-stage boot process (PXE). In such cases, the MAC address of the client's interface remains stable, and using the "chaddr" field to identify the client guarantees that the particular system is considered to be the same client, even though its "client identifier" changes.

To address this problem, Kea includes a configuration option which enables client identification using "chaddr" only. This instructs the server to "ignore" the "client identifier" during lease lookups and allocations for a particular subnet. Consider the following simplified server configuration:

```
"Dhcp4": {
    ...
    "match-client-id": true,
    ...
    "subnet4": [
    {
        "subnet": "192.0.10.0/24",
        "pools": [ { "pool": "192.0.2.23-192.0.2.87" } ],
        "match-client-id": false
    },
    {
        "subnet": "10.0.0.0/8",
        "pools": [ { "pool": "10.0.0.23-10.0.2.99" } ],
    }
    ]
}
```

The `match-client-id` is a boolean value which controls this behavior. The default value of `true` indicates that the server will use the "client identifier" for lease lookups and "chaddr" if the first lookup returns no results. The `false` means that the server will only use the "chaddr" to search for the client's lease. Whether the DHCID for DNS updates is generated from the "client identifier" or "chaddr" is controlled through the same parameter.

The `match-client-id` parameter may appear both in the global configuration scope and/or under any subnet declaration. In the example shown above, the effective value of the `match-client-id` will be `false` for the subnet 192.0.10.0/24, because the subnet-specific setting of the parameter overrides the global value of the parameter. The effective value of the `match-client-id` for the subnet 10.0.0.0/8 will be set to `true` because the subnet declaration lacks this parameter and the global setting is by default used for this subnet. In fact, the global entry for this parameter could be omitted in this case, because `true` is the default value.

It is important to understand what happens when the client obtains its lease for one setting of the `match-client-id` and then renews it when the setting has been changed. First, consider the case when the client obtains the lease and the `match-client-id` is set to `true`. The server will store the lease information, including "client identifier" (if supplied) and "chaddr", in the lease database. When the setting is changed and the client renews the lease, the server will determine that it should use the "chaddr" to search for the existing lease. If the client hasn't changed its MAC address, the server should successfully find the existing lease. The "client identifier" associated with the returned lease will be ignored and the client will be allowed to use this lease. When the lease is renewed only the "chaddr" will be recorded for this lease, according to the new server setting.

In the second case the client has the lease with only a "chaddr" value recorded. When the `match-client-id` setting is changed to `true`, the server will first try to use the "client identifier" to find the existing client's lease. This will return no results because the "client identifier" was not recorded for this lease. The server will then use the "chaddr" and the lease will be found. If the lease appears to have no "client identifier" recorded, the server will assume that this lease belongs to the client and that it was created with the previous setting of the `match-client-id`. However, if the lease contains a "client identifier" which is different from the "client identifier" used by the client, the lease will be assumed to belong to another client and the new lease will be allocated.

8.2.22 Authoritative DHCPv4 Server Behavior

The original DHCPv4 specification (RFC 2131) states that if a client requests an address in the INIT-REBOOT state, of which the server has no knowledge, the server must remain silent, except if the server knows that the client has requested an IP address from the wrong network. By default, Kea follows the behavior of the ISC dhcpd daemon instead of the specification and also remains silent if the client requests an IP address from the wrong network, because configuration information about a given network segment is not known to be correct. Kea only rejects a client's DHCPREQUEST with a DHCPNAK message if it already has a lease for the client with a different IP address. Administrators can override this behavior through the boolean `authoritative` (`false` by default) setting.

In authoritative mode, `authoritative` set to `true`, Kea always rejects INIT-REBOOT requests from unknown clients with DHCPNAK messages. The `authoritative` setting can be specified in global, shared-network, and subnet configuration scope and is automatically inherited from the parent scope, if not specified. All subnets in a shared-network must have the same `authoritative` setting.

8.2.23 DHCPv4-over-DHCPv6: DHCPv4 Side

The support of DHCPv4-over-DHCPv6 transport is described in RFC 7341 and is implemented using cooperating DHCPv4 and DHCPv6 servers. This section is about the configuration of the DHCPv4 side (the DHCPv6 side is described in *DHCPv4-over-DHCPv6: DHCPv6 Side*).

Note: DHCPv4-over-DHCPv6 support is experimental and the details of the inter-process communication may change; both the DHCPv4 and DHCPv6 sides should be running the same version of Kea. For instance, the support of port relay (RFC 8357) introduced an incompatible change.

The `dhcp4o6-port` global parameter specifies the first of the two consecutive ports of the UDP sockets used for the communication between the DHCPv6 and DHCPv4 servers. The DHCPv4 server is bound to ::1 on `port` + 1 and connected to ::1 on `port`.

With DHCPv4-over-DHCPv6, the DHCPv4 server does not have access to several of the identifiers it would normally use to select a subnet. To address this issue, three new configuration entries have been added; the presence of any of these allows the subnet to be used with DHCPv4-over-DHCPv6. These entries are:

- `4o6-subnet`: takes a prefix (i.e., an IPv6 address followed by a slash and a prefix length) which is matched against the source address.
- `4o6-interface-id`: takes a relay interface ID option value.
- `4o6-interface`: takes an interface name which is matched against the incoming interface name.

The following configuration was used during some tests:

```
{

# DHCPv4 conf
"Dhcp4": {
    "interfaces-config": {
        "interfaces": [ "eno33554984" ]
    },

    "lease-database": {
        "type": "memfile",
        "name": "leases4"
    },

    "valid-lifetime": 4000,

    "subnet4": [ {
        "subnet": "10.10.10.0/24",
        "4o6-interface": "eno33554984",
        "4o6-subnet": "2001:db8:1:1::/64",
        "pools": [ { "pool": "10.10.10.100 - 10.10.10.199" } ]
    } ],

    "dhcp4o6-port": 6767,

    "loggers": [ {
        "name": "kea-dhcp4",
        "output_options": [ {
            "output": "/tmp/kea-dhcp4.log"
        } ],
        "severity": "DEBUG",
        "debuglevel": 0
    } ]
}

}
```

8.2.24 Sanity Checks in DHCPv4

An important aspect of a well-running DHCP system is an assurance that the data remain consistent. However, in some cases it may be convenient to tolerate certain inconsistent data. For example, a network administrator that temporarily removed a subnet from a configuration would not want all the leases associated with it to disappear from the lease database. Kea has a mechanism to control sanity checks such as this.

Kea supports a configuration scope called `sanity-checks`. It currently allows only a single parameter, called `lease-checks`, which governs the verification carried out when a new lease is loaded from a lease file. This mechanism permits Kea to attempt to correct inconsistent data.

Every subnet has a subnet-id value; this is how Kea internally identifies subnets. Each lease has a subnet-id parameter as well, which identifies which subnet it belongs to. However, if the configuration has changed, it is possible that a lease could exist with a subnet-id, but without any subnet that matches it. Also, it may be possible that the subnet's configuration has changed and the subnet-id now belongs to a subnet that does not match the lease. Kea's corrective algorithm first checks to see if there is a subnet with the subnet-id specified by the lease. If there is, it verifies whether the lease belongs to that subnet. If not, depending on the lease-checks setting, the lease is discarded, a warning is displayed, or a new subnet is selected for the lease that matches it topologically.

There are five levels which are supported:

- `none` - do no special checks; accept the lease as is.
- `warn` - if problems are detected display a warning, but accept the lease data anyway. This is the default value. If not explicitly configured to some other value, this level will be used.
- `fix` - if a data inconsistency is discovered, try to correct it. If the correction is not successful, the incorrect data will be inserted anyway.
- `fix-del` - if a data inconsistency is discovered, try to correct it. If the correction is not successful, reject the lease. This setting ensures the data's correctness, but some incorrect data may be lost. Use with care.
- `del` - this is the strictest mode. If any inconsistency is detected, reject the lease. Use with care.

This feature is currently implemented for the memfile backend.

An example configuration that sets this parameter looks as follows:

```
"Dhcp4": {
    "sanity-checks": {
        "lease-checks": "fix-del"
    },
    ...
}
```

8.2.25 Storing Extended Lease Information

In order to support such features as DHCP LeaseQuery (RFC 4388) it is necessary to store additional information with each lease. Because the amount of information stored for each lease has ramifications in terms of performance and system resource consumption, storing this additional information is configurable through the "store-extended-info" parameter. It defaults to false and may be set at the global, shared-network, and subnet levels.

```
"Dhcp4": {
    "store-extended-info": true,
    ...
}
```

When enabled, information relevant to the DHCPREQUEST asking for the lease is added into the lease's user-context as a map element labeled "ISC". Currently, the map will contain a single value, the relay-agent-info option (DHCP Option 82), when the DHCPREQUEST received contains it. Other values may be added at a future date. Since DHCPREQUESTs sent as renewals will likely not contain this information, the values taken from the last DHCPREQUEST that did contain it will be retained on the lease. The lease's user-context will look something like this:

```
{ "ISC": { "relay-agent-info": "0x52050104AABBCCDD" } }
```

Note: This feature is intended to be used in conjunction with an upcoming LeaseQuery hook library and at this time there is other use for this information within Kea.

Note: It is possible that other hook libraries are already making use of user-context. Enabling store-extended-info should not interfere with any other user-context content so long as it does not also use an element labled "ISC". In other words, user-context is intended to be a flexible container serving mulitple purposes. As long as no other purpose also writes an "ISC" element to user-context there should not be a conflict.

8.2.26 Multi-threading settings

The Kea server can be configured to process packets in parallel using multiple threads. These settings can be found under `multi-threading` structure and are represented by:

- `enable-multi-threading` - use multiple threads to process packets in parallel (default false).
- `thread-pool-size` - specify the number of threads to process packets in parallel. Supported values are: 0 (auto detect), any positive number sets thread count explicitly (default 0).
- `packet-queue-size` - specify the size of the queue used by the thread pool to process packets. Supported values are: 0 (unlimited), any positive number sets queue size explicitly (default 64).

An example configuration that sets these parameter looks as follows:

```
"Dhcp4": {
    "multi-threading": {
       "enable-multi-threading": true,
       "thread-pool-size": 4,
       "packet-queue-size": 16
    }
    ...
}
```

8.2.27 Multi-threading settings in different backends

Both kea-dhcp4 and kea-dhcp6 are tested internally to determine which settings gave the best performance. Although this section describes our results, those are just recommendations and are very dependent on the particular hardware that was used for testing. We strongly advice to run your own performance tests.

Full report of Kea 1.7 performance results can be found here. This include hardware description, test scenario descriptions and current results.

After enabling multi-threading, the number of threads is set by `thread-pool-size` parameter, and results from our tests show that best configurations for kea-dhcp4 are:

- `thread-pool-size`: 4 when using `memfile` for storing leases.
- `thread-pool-size`: 12 or more when using `mysql` for storing leases.
- `thread-pool-size`: 8 when using `postgresql`.

Another very important parameter is `packet-queue-size` and in our tests we used it as multiplier of `thread-pool-size`. So actual setting strongly depends on `thread-pool-size`.

Our tests reported best results when:

- `packet-queue-size`: 7 * `thread-pool-size` when using `memfile` for storing leases. In our case it's 7 * 4 = 28. This means that at any given time, up to 28 packets could be queued.
- `packet-queue-size`: 66 * `thread-pool-size` when using `mysql` for storing leases. In our case it's 66 * 12 = 792. This means that up to 792 packets could be queued.
- `packet-queue-size`: 11 * `thread-pool-size` when using `postgresql` for storing leases. In our case it's 11 * 8 = 88.

8.2.28 IPv6-only Preferred Networks

A recent Internet-Draft draft-ietf-dhc-v6only soon to be published by IETF as RFC specifies a DHCPv4 option to indicate that a host supports an IPv6-only mode and is willing to forgo obtaining an IPv4 address if the network provides IPv6 connectivity. The general idea is that a network administrator can enable this option to signal to compatible dual-stack devices that the IPv6 connectivity is available and they can shut down their IPv4 stack. The new option *v6-only-preferred* content is a 32 bit unsigned integer and specifies for how long the device should disable its stack for. The value is expressed in seconds.

The draft mentions V6ONLY_WAIT timer. This is implemented in Kea by setting the value of v6-only-preferred option. This follows the usual practice of setting options. You can specify the option value on pool, subnet, shared network, global levels or even specify it in host reservations.

Note there is no special processing involved. This follows the standard Kea option processing regime. The option will not be sent back, unless the client explicitly requests it. For example, to enable the option for the whole subnet, the following configuration can be used:

```
"subnet4": [
    {
        "pools": [ { "pool":  "192.0.2.1 - 192.0.2.200" } ],
        "subnet": "192.0.2.0/24",
        "option-data": [
            {
                // This will make the v6-only capable devices to disable their
                // v4 stack for half an hour and then try again
                "name": "v6-only-preferred",
                "data": "1800"
            }
        ]
    }
],
```

8.3 Host Reservation in DHCPv4

There are many cases where it is useful to provide a configuration on a per-host basis. The most obvious one is to reserve a specific, static address for exclusive use by a given client (host); the returning client will receive the same address from the server every time, and other clients will generally not receive that address. Another situation when host reservations are applicable is when a host has specific requirements, e.g. a printer that needs additional DHCP options. Yet another possible use case is to define unique names for hosts.

Note that there may be cases when a new reservation has been made for a client for an address currently in use by another client. We call this situation a "conflict." These conflicts get resolved automatically over time as described in subsequent sections. Once the conflict is resolved, the correct client will receive the reserved configuration when it renews.

Host reservations are defined as parameters for each subnet. Each host must have its own unique identifier, such as the hardware/MAC address. There is an optional `reservations` array in the `subnet4` structure; each element in that array is a structure that holds information about reservations for a single host. In particular, the structure must have a unique host identifier. In the DHCPv4 context, the identifier is usually a hardware or MAC address. In most cases an IP address will be specified. It is also possible to specify a hostname, host-specific options, or fields carried within the DHCPv4 message such as siaddr, sname, or file.

Note: Kea requires that reserved address must be within the subnet. Kea 1.7.10 is the last release that does not enforce this.

The following example shows how to reserve addresses for specific hosts in a subnet:

```
"subnet4": [
    {
        "pools": [ { "pool":  "192.0.2.1 - 192.0.2.200" } ],
        "subnet": "192.0.2.0/24",
        "interface": "eth0",
        "reservations": [
            {
                "hw-address": "1a:1b:1c:1d:1e:1f",
                "ip-address": "192.0.2.202"
            },
            {
                "duid": "0a:0b:0c:0d:0e:0f",
                "ip-address": "192.0.2.100",
                "hostname": "alice-laptop"
            },
            {
                "circuit-id": "'charter950'",
                "ip-address": "192.0.2.203"
            },
            {
                "client-id": "01:11:22:33:44:55:66",
                "ip-address": "192.0.2.204"
            }
        ]
    }
]
```

The first entry reserves the 192.0.2.202 address for the client that uses a MAC address of 1a:1b:1c:1d:1e:1f. The second entry reserves the address 192.0.2.100 and the hostname of alice-laptop for the client using a DUID 0a:0b:0c:0d:0e:0f. (Note that if DNS updates are planned, it is strongly recommended that the hostnames be unique.) The third example reserves address 192.0.3.203 for a client whose request would be relayed by a relay agent that inserts a circuit-id option with the value "charter950". The fourth entry reserves address 192.0.2.204 for a client that uses a client identifier with value 01:11:22:33:44:55:66.

The above example is used for illustrational purposes only; in actual deployments it is recommended to use as few types as possible (preferably just one). See *Fine-Tuning DHCPv4 Host Reservation* for a detailed discussion of this point.

Making a reservation for a mobile host that may visit multiple subnets requires a separate host definition in each subnet that host is expected to visit. It is not possible to define multiple host definitions with the same hardware address in a single subnet. Multiple host definitions with the same hardware address are valid if each is in a different subnet.

Adding host reservations incurs a performance penalty. In principle, when a server that does not support host reservation responds to a query, it needs to check whether there is a lease for a given address being considered for allocation or renewal. The server that does support host reservation has to perform additional checks: not only whether the address is currently used (i.e., if there is a lease for it), but also whether the address could be used by someone else (i.e., if there is a reservation for it). That additional check incurs extra overhead.

8.3.1 Address Reservation Types

In a typical scenario there is an IPv4 subnet defined, e.g. 192.0.2.0/24, with a certain part of it dedicated for dynamic allocation by the DHCPv4 server. That dynamic part is referred to as a dynamic pool or simply a pool. In principle, a host reservation can reserve any address that belongs to the subnet. The reservations that specify addresses that belong to configured pools are called "in-pool reservations." In contrast, those that do not belong to dynamic pools are called "out-of-pool reservations." There is no formal difference in the reservation syntax and both reservation types are handled uniformly.

Kea supports global host reservations. These are reservations that are specified at the global level within the configuration and that do not belong to any specific subnet. Kea will still match inbound client packets to a subnet as before, but when the subnet's reservation mode is set to `"global"`, Kea will look for host reservations only among the global reservations defined. Typically, such reservations would be used to reserve hostnames for clients which may move from one subnet to another.

Note: Global reservations, while useful in certain circumstances, have aspects that must be given due consideration when using them, please see *Conflicts in DHCPv4 Reservations* for more details.

8.3.2 Conflicts in DHCPv4 Reservations

As reservations and lease information are stored separately, conflicts may arise. Consider the following series of events: the server has configured the dynamic pool of addresses from the range of 192.0.2.10 to 192.0.2.20. Host A requests an address and gets 192.0.2.10. Now the system administrator decides to reserve address 192.0.2.10 for Host B. In general, reserving an address that is currently assigned to someone else is not recommended, but there are valid use cases where such an operation is warranted.

The server now has a conflict to resolve. If Host B boots up and requests an address, the server is not able to assign the reserved address 192.0.2.10. A naive approach would to be immediately remove the existing lease for Host A and create a new one for Host B. That would not solve the problem, though, because as soon as Host B gets the address, it will detect that the address is already in use (by Host A) and will send a DHCPDECLINE message. Therefore, in this situation, the server has to temporarily assign a different address from the dynamic pool (not matching what has been reserved) to Host B.

When Host A renews its address, the server will discover that the address being renewed is now reserved for another host - Host B. The server will inform Host A that it is no longer allowed to use it by sending a DHCPNAK message. The server will not remove the lease, though, as there's a small chance that the DHCPNAK may be lost if the network is lossy. If that happens, the client will not receive any responses, so it will retransmit its DHCPREQUEST packet.

Once the DHCPNAK is received by Host A, it will revert to server discovery and will eventually get a different address. Besides allocating a new lease, the server will also remove the old one. As a result, address 192.0.2.10 will become free. When Host B tries to renew its temporarily assigned address, the server will detect that it has a valid lease, but will note that there is a reservation for a different address. The server will send DHCPNAK to inform Host B that its address is no longer usable, but will keep its lease (again, the DHCPNAK may be lost, so the server will keep it until the client returns for a new address). Host B will revert to the server discovery phase and will eventually send a DHCPREQUEST message. This time the server will find that there is a reservation for that host and that the reserved address 192.0.2.10 is not used, so it will be granted. It will also remove the lease for the temporarily assigned address that Host B previously obtained.

This recovery will succeed, even if other hosts attempt to get the reserved address. If Host C requests the address 192.0.2.10 after the reservation is made, the server will either offer a different address (when responding to DHCPDISCOVER) or send DHCPNAK (when responding to DHCPREQUEST).

The recovery mechanism allows the server to fully recover from a case where reservations conflict with existing leases; however, this procedure will take roughly as long as the value set for renew-timer. The best way to avoid such recovery is not to define new reservations that conflict with existing leases. Another recommendation is to use out-of-pool reservations. If the reserved address does not belong to a pool, there is no way that other clients can get it.

Note: The conflict-resolution mechanism does not work for global reservations. Although the global address reservations feature may be useful in certain settings, it is generally recommended not to use global reservations for addresses. Administrators who do choose to use global reservations must manually ensure that the reserved addresses are not in dynamic pools.

8.3.3 Reserving a Hostname

When the reservation for a client includes the `hostname`, the server will return this hostname to the client in the Client FQDN or Hostname option. The server responds with the Client FQDN option only if the client has included the Client FQDN option in its message to the server. The server will respond with the Hostname option if the client included the Hostname option in its message to the server, or if the client requested the Hostname option using the Parameter Request List option. The server will return the Hostname option even if it is not configured to perform DNS updates. The reserved hostname always takes precedence over the hostname supplied by the client or the autogenerated (from the IPv4 address) hostname.

The server qualifies the reserved hostname with the value of the `ddns-qualifying-suffix` parameter. For example, the following subnet configuration:

```
{
    "subnet4": [ {
        "subnet": "10.0.0.0/24",
        "pools": [ { "pool": "10.0.0.10-10.0.0.100" } ],
        "ddns-qualifying-suffix": "example.isc.org.",
        "reservations": [
           {
             "hw-address": "aa:bb:cc:dd:ee:ff",
             "hostname": "alice-laptop"
           }
        ]
     }],
    "dhcp-ddns": {
        "enable-updates": true,
    }
}
```

will result in assigning the "alice-laptop.example.isc.org." hostname to the client using the MAC address "aa:bb:cc:dd:ee:ff". If the `ddns-qualifying-suffix` is not specified, the default (empty) value will be used, and in this case the value specified as a `hostname` will be treated as a fully qualified name. Thus, by leaving the `ddns-qualifying-suffix` empty it is possible to qualify hostnames for different clients with different domain names:

```
{
    "subnet4": [ {
        "subnet": "10.0.0.0/24",
        "pools": [ { "pool": "10.0.0.10-10.0.0.100" } ],
        "reservations": [
           {
             "hw-address": "aa:bb:cc:dd:ee:ff",
             "hostname": "alice-laptop.isc.org."
           },
           {
             "hw-address": "12:34:56:78:99:AA",
             "hostname": "mark-desktop.example.org."
           }

        ]
     }],
    "dhcp-ddns": {
        "enable-updates": true,
    }
}
```

8.3.4 Including Specific DHCPv4 Options in Reservations

Kea offers the ability to specify options on a per-host basis. These options follow the same rules as any other options. These can be standard options (see *Standard DHCPv4 Options*), custom options (see *Custom DHCPv4 Options*), or vendor-specific options (see *DHCPv4 Vendor-Specific Options*). The following example demonstrates how standard options can be defined.

```
{
    "subnet4": [ {
        "reservations": [
        {
            "hw-address": "aa:bb:cc:dd:ee:ff",
            "ip-address": "192.0.2.1",
            "option-data": [
            {
                "name": "cookie-servers",
                "data": "10.1.1.202,10.1.1.203"
            },
            {
                "name": "log-servers",
                "data": "10.1.1.200,10.1.1.201"
            } ]
        } ]
    } ]
}
```

Vendor-specific options can be reserved in a similar manner:

```
{
    "subnet4": [ {
        "reservations": [
        {
            "hw-address": "aa:bb:cc:dd:ee:ff",
            "ip-address": "10.0.0.7",
            "option-data": [
            {
                "name": "vivso-suboptions",
                "data": "4491"
            },
            {
                "name": "tftp-servers",
                "space": "vendor-4491",
                "data": "10.1.1.202,10.1.1.203"
            } ]
        } ]
    } ]
}
```

Options defined at host level have the highest priority. In other words, if there are options defined with the same type on global, subnet, class, and host levels, the host-specific values will be used.

8.3.5 Reserving Next Server, Server Hostname, and Boot File Name

BOOTP/DHCPv4 messages include "siaddr", "sname", and "file" fields. Even though DHCPv4 includes corresponding options, such as option 66 and option 67, some clients may not support these options. For this reason, server administrators often use the "siaddr", "sname", and "file" fields instead.

With Kea, it is possible to make static reservations for these DHCPv4 message fields:

```
{
    "subnet4": [ {
        "reservations": [
        {
            "hw-address": "aa:bb:cc:dd:ee:ff",
            "next-server": "10.1.1.2",
            "server-hostname": "server-hostname.example.org",
            "boot-file-name": "/tmp/bootfile.efi"
        } ]
    } ]
}
```

Note that those parameters can be specified in combination with other parameters for a reservation, such as a reserved IPv4 address. These parameters are optional; a subset of them can be specified, or all of them can be omitted.

8.3.6 Reserving Client Classes in DHCPv4

Using Expressions in Classification explains how to configure the server to assign classes to a client, based on the content of the options that this client sends to the server. Host reservations mechanisms also allow for the static assignment of classes to clients. The definitions of these classes are placed in the Kea configuration or a database. The following configuration snippet shows how to specify that a client belongs to classes `reserved-class1` and `reserved-class2`. Those classes are associated with specific options sent to the clients which belong to them.

```
{
    "client-classes": [
    {
       "name": "reserved-class1",
       "option-data": [
       {
           "name": "routers",
           "data": "10.0.0.200"
       }
       ]
    },
    {
       "name": "reserved-class2",
       "option-data": [
       {
           "name": "domain-name-servers",
           "data": "10.0.0.201"
       }
       ]
    }
    ],
    "subnet4": [ {
        "subnet": "10.0.0.0/24",
        "pools": [ { "pool": "10.0.0.10-10.0.0.100" } ],
        "reservations": [
        {
            "hw-address": "aa:bb:cc:dd:ee:ff",

            "client-classes": [ "reserved-class1", "reserved-class2" ]

        }
        ]
    } ]
}
```

In some cases the host reservations can be used in conjuction with client classes specified within the Kea configuration. In particular, when a host reservation exists for a client within a given subnet, the “KNOWN” built-in class is assigned to the client. Conversely, when there is no static assignment for the client, the “UNKNOWN” class is assigned to the client. Class expressions within the Kea configuration file can refer to “KNOWN” or “UNKNOWN” classes using using the “member” operator. For example:

```
{
    "client-classes": [
        {
            "name": "dependent-class",
            "test": "member('KNOWN')",
            "only-if-required": true
        }
    ]
}
```

Note that the `only-if-required` parameter is needed here to force evaluation of the class after the lease has been allocated and thus the reserved class has been also assigned.

Note: Be aware that the classes specified in non global host reservations are assigned to the processed packet after all classes with the `only-if-required` parameter set to `false` have been evaluated. This has an implication that these classes must not depend on the statically assigned classes from the host reservations. If there is a need to create such dependency, the `only-if-required` must be set to `true` for the dependent classes. Such classes are evaluated after the static classes have been assigned to the packet. This, however, imposes additional configuration overhead, because all classes marked as `only-if-required` must be listed in the `require-client-classes` list for every subnet where they are used.

Note: Client classes specified within the Kea configuration file may depend on the classes specified within the global host reservations. In such case the `only-if-required` parameter is not needed. Refer to the *Pool Selection with Client Class Reservations* and *Subnet Selection with Client Class Reservations* for the specific use cases.

8.3.7 Storing Host Reservations in MySQL, PostgreSQL, or Cassandra

It is possible to store host reservations in MySQL, PostgreSQL, or Cassandra. See *Hosts Storage* for information on how to configure Kea to use reservations stored in MySQL, PostgreSQL, or Cassandra. Kea provides a dedicated hook for managing reservations in a database; section *host_cmds: Host Commands* provides detailed information. The Kea wiki provides some examples of how to conduct common host reservation operations.

Note: In Kea, the maximum length of an option specified per-host is arbitrarily set to 4096 bytes.

8.3.8 Fine-Tuning DHCPv4 Host Reservation

The host reservation capability introduces additional restrictions for the allocation engine (the component of Kea that selects an address for a client) during lease selection and renewal. In particular, three major checks are necessary. First, when selecting a new lease, it is not sufficient for a candidate lease to simply not be in use by another DHCP client; it also must not be reserved for another client. Second, when renewing a lease, an additional check must be performed to see whether the address being renewed is reserved for another client. Finally, when a host renews an address, the server must check whether there is a reservation for this host, so the existing (dynamically allocated) address should be revoked and the reserved one be used instead.

Some of those checks may be unnecessary in certain deployments, and not performing them may improve performance. The Kea server provides the `reservation-mode` configuration parameter to select the types of reservations allowed for a particular subnet. Each reservation type has different constraints for the checks to be performed by the server when allocating or renewing a lease for the client. Allowed values are:

- `all` - enables both in-pool and out-of-pool host reservation types. This setting is the default value, and is the safest and most flexible. However, as all checks are conducted, it is also the slowest. It does not check against global reservations.
- `out-of-pool` - allows only out-of- pool host reservations. With this setting in place, the server may assume that all host reservations are for addresses that do not belong to the dynamic pool. Therefore, it can skip the reservation checks when dealing with in-pool addresses, thus improving performance. Do not use this mode if any reservations use in-pool addresses. Caution is advised when using this setting; Kea does not sanity-check the reservations against `reservation-mode` and misconfiguration may cause problems.
- `global` - allows only global host reservations. With this setting in place, the server searches for reservations for a client only among the defined global reservations. If an address is specified, the server skips the reservation checks carried out when dealing in other modes, thus improving performance. Caution is advised when using this setting; Kea does not sanity-check the reservations when `global` and misconfiguration may cause problems.

- `disabled` - host reservation support is disabled. As there are no reservations, the server will skip all checks. Any reservations defined will be completely ignored. As the checks are skipped, the server may operate faster in this mode.

The parameter can be specified at global, subnet, and shared-network levels.

An example configuration that disables reservation looks as follows:

```
"Dhcp4": {
    "subnet4": [
    {
        "subnet": "192.0.2.0/24",
        "reservation-mode": "disabled",
        ...
    }
    ]
}
```

An example configuration using global reservations is shown below:

```
"Dhcp4": {


    "reservation-mode": "global",
    "reservations": [
       {
        "hw-address": "01:bb:cc:dd:ee:ff",
        "hostname": "host-one"
       },
       {
        "hw-address": "02:bb:cc:dd:ee:ff",
        "hostname": "host-two"
       }
    ],

    "subnet4": [
    {
        "subnet": "192.0.2.0/24",
        ...
    }
    ]
}
```

For more details regarding global reservations, see *Global Reservations in DHCPv4*.

Another aspect of host reservations is the different types of identifiers. Kea currently supports four types of identifiers: hw-address, duid, client-id, and circuit-id. This is beneficial from a usability perspective; however, there is one drawback. For each incoming packet, Kea has to extract each identifier type and then query the database to see if there is a reservation by this particular identifier. If nothing is found, the next identifier is extracted and the next query is issued. This process continues until either a reservation is found or all identifier types have been checked. Over time, with an increasing number of supported identifier types, Kea would become slower and slower.

To address this problem, a parameter called `host-reservation-identifiers` is available. It takes a list of identifier types as a parameter. Kea will check only those identifier types enumerated in host-reservation-identifiers. From a performance perspective, the number of identifier types should be kept to a minimum, ideally one. If the deployment uses several reservation types, please enumerate them from most- to least-frequently used, as this increases the chances of Kea finding the reservation using the fewest queries. An example of host reservation identifiers looks as follows:

```
"host-reservation-identifiers": [ "circuit-id", "hw-address", "duid", "client-id" ],
"subnet4": [
    {
        "subnet": "192.0.2.0/24",
        ...
    }
]
```

If not specified, the default value is:

```
"host-reservation-identifiers": [ "hw-address", "duid", "circuit-id", "client-id" ]
```

8.3.9 Global Reservations in DHCPv4

In some deployments, such as mobile, clients can roam within the network and certain parameters must be specified regardless of the client's current location. To facilitate such a need, a global reservation mechanism has been implemented. The idea behind it is that regular host reservations are tied to specific subnets, by using a specific subnet-id. Kea can specify a global reservation that can be used in every subnet that has global reservations enabled.

This feature can be used to assign certain parameters, such as hostname or other dedicated, host-specific options. It can also be used to assign addresses. However, global reservations that assign addresses bypass the whole topology determination provided by DHCP logic implemented in Kea. It is very easy to misuse this feature and get a configuration that is inconsistent. To give a specific example, imagine a global reservation for address 192.0.2.100 and two subnets 192.0.2.0/24 and 192.0.5.0/24. If global reservations are used in both subnets and a device matching global host reservations visits part of the network that is serviced by 192.0.5.0/24, it will get an IP address 192.0.2.100, a subnet 192.0.5.0 and a default router 192.0.5.1. Obviously, such a configuration is unusable, as the client will not be able to reach its default gateway.

To use global host reservations, a configuration similar to the following can be used:

```
"Dhcp4:" {
    # This specifies global reservations. They will apply to all subnets that
    # have global reservations enabled.

    "reservations": [
    {
       "hw-address": "aa:bb:cc:dd:ee:ff",
       "hostname": "hw-host-dynamic"
    },
    {
       "hw-address": "01:02:03:04:05:06",
       "hostname": "hw-host-fixed",

       # Use of IP address in global reservation is risky. If used outside of
       # a matching subnet, such as 192.0.1.0/24, it will result in a broken
       # configuration being handed to the client.
       "ip-address": "192.0.1.77"
    },
    {
       "duid": "01:02:03:04:05",
       "hostname": "duid-host"
    },
    {
       "circuit-id": "'charter950'",
       "hostname": "circuit-id-host"
    },
    {
       "client-id": "01:11:22:33:44:55:66",
       "hostname": "client-id-host"
    }
    ],
    "valid-lifetime": 600,
    "subnet4": [ {
        "subnet": "10.0.0.0/24",
        "reservation-mode": "global",
        "pools": [ { "pool": "10.0.0.10-10.0.0.100" } ]
    } ]
}
```

When using database backends, the global host reservations are distinguished from regular reservations by using a subnet-id value of zero.

8.3.10 Pool Selection with Client Class Reservations

Client classes can be specified both in the Kea configuration file and/or host reservations. The classes specified in the Kea configuration file are evaluated immediately after receiving the DHCP packet and therefore can be used to influence subnet selection using the `client-class` parameter specified in the subnet scope. The classes specified within the host reservations are fetched and assigned to the packet after the server has already selected a subnet for the client. This means that the client class specified within a host reservation cannot be used to influence subnet assignment for this client, unless the subnet belongs to a shared network. If the subnet belongs to a shared network, the server may dynamically change the subnet assignment while trying to allocate a lease. If the subnet does not belong to a shared network, once selected, the subnet is not changed.

If the subnet does not belong to a shared network, it is possible to use host reservation based client classification to select an address pool within the subnet as follows:

```
"Dhcp4": {
    "client-classes": [
        {
            "name": "reserved_class"
        },
        {
            "name": "unreserved_class",
            "test": "not member('reserved_class')"
        }
    ],
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "reservations": [{"
                "hw-address": "aa:bb:cc:dd:ee:fe",
                "client-classes": [ "reserved_class" ]
             }],
            "pools": [
                {
                    "pool": "192.0.2.10-192.0.2.20",
                    "client-class": "reserved_class"
                },
                {
                    "pool": "192.0.2.30-192.0.2.40",
                    "client-class": "unreserved_class"
                }
            ]
        }
    ]
}
```

The `reserved_class` is declared without the `test` parameter because it may be only assigned to the client via host reservation mechanism. The second class, `unreserved_class`, is assigned to the clients which do not belong to the `reserved_class`. The first pool within the subnet is only used for the clients having a reservation for the `reserved_class`. The second pool is used for the clients not having such reservation. The configuration snippet includes one host reservation which causes the client having the MAC address of aa:bb:cc:dd:ee:fe to be assigned to the `reserved_class`. Thus, this client will be given an IP address from the first address pool.

8.3.11 Subnet Selection with Client Class Reservations

There is one specific use case when subnet selection may be influenced by client classes specified within host reservations. This is the case when the client belongs to a shared network. In such case it is possible to use classification to select a subnet within this shared network. Consider the following example:

```
"Dhcp4": {
    "client-classes": [
        {
            "name": "reserved_class"
        },
        {
            "name: "unreserved_class",
            "test": "not member('reserved_class")
        }
    ],
    "reservations": [{"
        "hw-address": "aa:bb:cc:dd:ee:fe",
        "client-classes": [ "reserved_class" ]
    }],
    "reservation-mode": "global",
    "shared-networks": [{
        "subnet4": [
            {
                "subnet": "192.0.2.0/24",
                "pools": [
                    {
                        "pool": "192.0.2.10-192.0.2.20",
                        "client-class": "reserved_class"
                    }
                ]
            },
            {
                "subnet": "192.0.3.0/24",
                "pools": [
                    {
                        "pool": "192.0.3.10-192.0.3.20",
                        "client-class": "unreserved_class"
                    }
                ]
            }
        ]
    }]
}
```

This is similar to the example described in the *Pool Selection with Client Class Reservations*. This time, however, there are two subnets, each of them having a pool associated with a different class. The clients which don't have a reservation for the `reserved_class` will be assigned an address from the subnet 192.0.3.0/24. Clients having a reservation for the `reserved_class` will be assigned an address from the subnet 192.0.2.0/24. The subnets must belong to the same shared network. In addition, the reservation for the client class must be specified at the global scope (global reservation) and the `reservation-mode` must be set to `global`.

In the example above the `client-class` could also be specified at the subnet level rather than pool level yielding the same effect.

8.4 Shared Networks in DHCPv4

DHCP servers use subnet information in two ways. First, it is used to determine the point of attachment, or where the client is connected to the network. Second, the subnet information is used to group information pertaining to a specific location in the network. This approach works well in general, but there are scenarios where the boundaries are blurred. Sometimes it is useful to have more than one logical IP subnet deployed on the same physical link. Understanding that two or more subnets are used on the same link requires additional logic in the DHCP server. This capability is called "shared networks" in the Kea and ISC DHCP projects. (It is sometimes also called "shared subnets"; in Microsoft's nomenclature it is called "multinet.")

There are many use cases where the feature is useful; this paragraph explains just a handful of the most common ones. The first and by far the most common use case is an existing network that has grown and is running out of available address space. Rather than migrating all devices to a new, larger subnet, it is easier to simply configure additional subnets on top of the existing one. Sometimes, due to address space fragmentation (e.g. only many disjointed /24s are available), this is the only choice. Also, configuring additional subnets has the advantage of not disrupting the operation of existing devices.

Another very frequent use case comes from cable networks. There are two types of devices in cable networks: cable modems and the end-user devices behind them. It is a common practice to use different subnets for cable modems to prevent users from tinkering with them. In this case, the distinction is based on the type of device, rather than on address-space exhaustion.

A client connected to a shared network may be assigned an address from any of the pools defined within the subnets belonging to the shared network. Internally, the server selects one of the subnets belonging to a shared network and tries to allocate an address from this subnet. If the server is unable to allocate an address from the selected subnet (e.g., due to address-pool exhaustion), it will use another subnet from the same shared network and will try to allocate an address from this subnet, etc. Therefore, the server will typically allocate all addresses available in a given subnet before it starts allocating addresses from other subnets belonging to the same shared network. However, in certain situations the client can be allocated an address from another subnet before the address pools in the first subnet get exhausted; this sometimes occurs when the client provides a hint that belongs to another subnet, or the client has reservations in a subnet other than the default.

Note: Deployments should not assume that Kea waits until it has allocated all the addresses from the first subnet in a shared network before allocating addresses from other subnets.

In order to define a shared network an additional configuration scope is introduced:

```
{
"Dhcp4": {
    "shared-networks": [
        {
            # Name of the shared network. It may be an arbitrary string
            # and it must be unique among all shared networks.
            "name": "my-secret-lair-level-1",
            # The subnet selector can be specified at the shared network level.
            # Subnets from this shared network will be selected for directly
            # connected clients sending requests to server's "eth0" interface.
            "interface": "eth0",

            # This starts a list of subnets in this shared network.
            # There are two subnets in this example.
            "subnet4": [
                {
                    "subnet": "10.0.0.0/8",
                    "pools": [ { "pool":  "10.0.0.1 - 10.0.0.99" } ],
                },
                {
                    "subnet": "192.0.2.0/24",
                    "pools": [ { "pool":  "192.0.2.100 - 192.0.2.199" } ]
                }
            ],
        } ], # end of shared-networks

    # It is likely that in the network there will be a mix of regular,
    # "plain" subnets and shared networks. It is perfectly valid to mix
    # them in the same configuration file.
    #
    # This is a regular subnet. It is not part of any shared network.
    "subnet4": [
        {
            "subnet": "192.0.3.0/24",
            "pools": [ { "pool":  "192.0.3.1 - 192.0.3.200" } ],
            "interface": "eth1"
        }
    ]

} # end of Dhcp4
}
```

As demonstrated in the example, it is possible to mix shared and regular (“plain”) subnets. Each shared network must have a unique name. This is similar to the ID for subnets, but gives administrators more flexibility. It is used for logging, but also internally for identifying shared networks.

In principle it makes sense to define only shared networks that consist of two or more subnets. However, for testing purposes, an empty subnet or a network with just a single subnet is allowed. This is not a recommended practice in production networks, as the shared network logic requires additional processing and thus lowers the server’s performance. To avoid unnecessary performance degradation, the shared subnets should only be defined when required by the deployment.

Shared networks provide an ability to specify many parameters in the shared network scope that apply to all subnets within it. If necessary, it is possible to specify a parameter in the shared network scope and then override its value in the subnet scope. For example:

```
"shared-networks": [
    {
        "name": "lab-network3",

        "interface": "eth0",

        # This applies to all subnets in this shared network, unless
        # values are overridden on subnet scope.
        "valid-lifetime": 600,

        # This option is made available to all subnets in this shared
        # network.
        "option-data": [ {
            "name": "log-servers",
            "data": "1.2.3.4"
        } ],

        "subnet4": [
            {
                "subnet": "10.0.0.0/8",
                "pools": [ { "pool":  "10.0.0.1 - 10.0.0.99" } ],

                # This particular subnet uses different values.
                "valid-lifetime": 1200,
                "option-data": [
                {
                    "name": "log-servers",
                    "data": "10.0.0.254"
                },
                {
                    "name": "routers",
                    "data": "10.0.0.254"
                } ]
            },
            {
                 "subnet": "192.0.2.0/24",
                 "pools": [ { "pool":  "192.0.2.100 - 192.0.2.199" } ],

                 # This subnet does not specify its own valid-lifetime value,
                 # so it is inherited from shared network scope.
                 "option-data": [
                 {
                     "name": "routers",
                     "data": "192.0.2.1"
                 } ]
            }
        ]
    } ]
```

In this example, there is a log-servers option defined that is available to clients in both subnets in this shared network. Also, the valid lifetime is set to 10 minutes (600s). However, the first subnet overrides some of the values (valid lifetime is 20 minutes, different IP address for log-servers), but also adds its own option (router address). Assuming a client asking for router and log servers options is assigned a lease from this subnet, it will get a lease for 20 minutes and a log-servers and routers value of 10.0.0.254. If the same client is assigned to the second subnet, it will get a 10-minute lease, a log-servers value of 1.2.3.4, and routers set to 192.0.2.1.

8.4.1 Local and Relayed Traffic in Shared Networks

It is possible to specify an interface name at the shared network level to tell the server that this specific shared network is reachable directly (not via relays) using the local network interface. As all subnets in a shared network are expected to be used on the same physical link, it is a configuration error to attempt to define a shared network using subnets that are reachable over different interfaces. In other words, all subnets within the shared network must have the same value of the "interface" parameter. The following configuration is wrong.

```
"shared-networks": [
    {
        "name": "office-floor-2",
        "subnet4": [
            {
                "subnet": "10.0.0.0/8",
                "pools": [ { "pool":  "10.0.0.1 - 10.0.0.99" } ],
                "interface": "eth0"
            },
            {
                 "subnet": "192.0.2.0/24",
                 "pools": [ { "pool":  "192.0.2.100 - 192.0.2.199" } ],

                 # Specifying the different interface name is a configuration
                 # error. This value should rather be "eth0" or the interface
                 # name in the other subnet should be "eth1".
                 "interface": "eth1"
            }
        ]
    } ]
```

To minimize the chance of the configuration errors, it is often more convenient to simply specify the interface name once, at the shared network level, like shown in the example below.

```
"shared-networks": [
    {
        "name": "office-floor-2",

        # This tells Kea that the whole shared network is reachable over a
        # local interface. This applies to all subnets in this network.
        "interface": "eth0",

        "subnet4": [
            {
                "subnet": "10.0.0.0/8",
                "pools": [ { "pool":  "10.0.0.1 - 10.0.0.99" } ],
            },
            {
                 "subnet": "192.0.2.0/24",
                 "pools": [ { "pool":  "192.0.2.100 - 192.0.2.199" } ]
            }
        ]
    } ]
```

In case of the relayed traffic, the subnets are typically selected using the relay agents' addresses. If the subnets are used independently (not grouped within a shared network) it is allowed to specify different relay address for each of these subnets. When multiple subnets belong to a shared network they must be selected via the same relay address and, similarly to the case of the local traffic described above, it is a configuration error to specify different relay addresses for the respective subnets in the shared network. The following configuration is wrong.

```
"shared-networks": [
    {
        "name": "kakapo",
        "subnet4": [
            {
                "subnet": "192.0.2.0/26",
                "relay": {
                    "ip-addresses": [ "192.1.1.1" ]
                },
                "pools": [ { "pool": "192.0.2.63 - 192.0.2.63" } ]
            },
            {
                "subnet": "10.0.0.0/24",
                "relay": {
                    # Specifying a different relay address for this
                    # subnet is a configuration error. In this case
                    # it should be 192.1.1.1 or the relay address
                    # in the previous subnet should be 192.2.2.2.
                    "ip-addresses": [ "192.2.2.2" ]
                },
                "pools": [ { "pool": "10.0.0.16 - 10.0.0.16" } ]
            }
        ]
    }
]
```

Again, it is better to specify the relay address at the shared network level and this value will be inherited by all subnets belonging to the shared network.

```
"shared-networks": [
    {
        "name": "kakapo",
        "relay": {
             # This relay address is inherited by both subnets.
            "ip-addresses": [ "192.1.1.1" ]
        },
        "subnet4": [
            {
                "subnet": "192.0.2.0/26",
                "pools": [ { "pool": "192.0.2.63 - 192.0.2.63" } ]
            },
            {
                "subnet": "10.0.0.0/24",
                "pools": [ { "pool": "10.0.0.16 - 10.0.0.16" } ]
            }
        ]
    }
]
```

Even though it is technically possible to configure two (or more) subnets within the shared network to use different relay addresses, this will almost always lead to a different behavior than what the user would expect. In this case, the Kea server will initially select one of the subnets by matching the relay address in the client's packet with the subnet's conifguration. However, it MAY end up using the other subnet (even though it does not match the relay address) if the client already has a lease in this subnet, has a host reservation in this subnet or simply the initially selected subnet has no more addresses available. Therefore, it is strongly recommended to always specify subnet selectors (interface or a relay address) at shared network level if the subnets belong to a shared network, as it is rarely useful to specify them at the subnet level and it may lead to the configurtion errors described above.

8.4.2 Client Classification in Shared Networks

Sometimes it is desirable to segregate clients into specific subnets based on certain properties. This mechanism is called client classification and is described in *Client Classification*. Client classification can be applied to subnets belonging to shared networks in the same way as it is used for subnets specified outside of shared networks. It is important to understand how the server selects subnets for clients when client classification is in use, to ensure that the desired subnet is selected for a given client type.

If a subnet is associated with a class, only the clients belonging to this class can use this subnet. If there are no classes specified for a subnet, any client connected to a given shared network can use this subnet. A common mistake is to assume that the subnet including a client class is preferred over subnets without client classes. Consider the following example:

```
{
    "client-classes": [
        {
            "name": "b-devices",
            "test": "option[93].hex == 0x0002"
        }
    ],
    "shared-networks": [
        {
            "name": "galah",
            "interface": "eth0",
            "subnet4": [
                {
                    "subnet": "192.0.2.0/26",
                    "pools": [ { "pool": "192.0.2.1 - 192.0.2.63" } ],
                },
                {
                    "subnet": "10.0.0.0/24",
                    "pools": [ { "pool": "10.0.0.2 - 10.0.0.250" } ],
                    "client-class": "b-devices"
                }
            ]
        }
    ]
}
```

If the client belongs to the “b-devices” class (because it includes option 93 with a value of 0x0002), that does not guarantee that the subnet 10.0.0.0/24 will be used (or preferred) for this client. The server can use either of the two subnets, because the subnet 192.0.2.0/26 is also allowed for this client. The client classification used in this case should be perceived as a way to restrict access to certain subnets, rather than a way to express subnet preference. For example, if the client does not belong to the “b-devices” class it may only use the subnet 192.0.2.0/26 and will never use the subnet 10.0.0.0/24.

A typical use case for client classification is in a cable network, where cable modems should use one subnet and other devices should use another subnet within the same shared network. In this case it is necessary to apply classification on all subnets. The following example defines two classes of devices, and the subnet selection is made based on option 93 values.

```
{
    "client-classes": [
        {

            "name": "a-devices",
            "test": "option[93].hex == 0x0001"
        },
        {
            "name": "b-devices",
            "test": "option[93].hex == 0x0002"
        }
    ],
    "shared-networks": [
        {
            "name": "galah",
            "interface": "eth0",
            "subnet4": [
                {
                    "subnet": "192.0.2.0/26",
                    "pools": [ { "pool": "192.0.2.1 - 192.0.2.63" } ],
                    "client-class": "a-devices"
                },
                {
                    "subnet": "10.0.0.0/24",
                    "pools": [ { "pool": "10.0.0.2 - 10.0.0.250" } ],
                    "client-class": "b-devices"
                }
            ]
        }
    ]
}
```

In this example each class has its own restriction. Only clients that belong to class “a-devices” will be able to use subnet 192.0.2.0/26 and only clients belonging to “b-devices” will be able to use subnet 10.0.0.0/24. Care should be taken not to define too-restrictive classification rules, as clients that are unable to use any subnets will be refused service. However, this may be a desired outcome if one wishes to provide service only to clients with known properties (e.g. only VoIP phones allowed on a given link).

Note that it is possible to achieve an effect similar to the one presented in this section without the use of shared networks. If the subnets are placed in the global subnets scope, rather than in the shared network, the server will still use classification rules to pick the right subnet for a given class of devices. The major benefit of placing subnets within the shared network is that common parameters for the logically grouped subnets can be specified once, in the shared network scope, e.g. the “interface” or “relay” parameter. All subnets belonging to this shared network will inherit those parameters.

8.4.3 Host Reservations in Shared Networks

Subnets that are part of a shared network allow host reservations, similar to regular subnets:

```
{
    "shared-networks": [
    {
        "name": "frog",
        "interface": "eth0",
        "subnet4": [
            {
                "subnet": "192.0.2.0/26",
                "id": 100,
                "pools": [ { "pool": "192.0.2.1 - 192.0.2.63" } ],
                "reservations": [
                    {
                        "hw-address": "aa:bb:cc:dd:ee:ff",
                        "ip-address": "192.0.2.28"
                    }
                ]
            },
            {
                "subnet": "10.0.0.0/24",
                "id": 101,
                "pools": [ { "pool": "10.0.0.1 - 10.0.0.254" } ],
                "reservations": [
                    {
                        "hw-address": "11:22:33:44:55:66",
                        "ip-address": "10.0.0.29"
                    }
                ]
            }
        ]
    }
    ]
}
```

It is worth noting that Kea conducts additional checks when processing a packet if shared networks are defined. First, instead of simply checking whether there's a reservation for a given client in its initially selected subnet, Kea looks through all subnets in a shared network for a reservation. This is one of the reasons why defining a shared network may impact performance. If there is a reservation for a client in any subnet, that particular subnet will be picked for the client. Although it is technically not an error, it is considered a bad practice to define reservations for the same host in multiple subnets belonging to the same shared network.

While not strictly mandatory, it is strongly recommended to use explicit "id" values for subnets if database storage will be used for host reservations. If an ID is not specified, the values for it are autogenerated, i.e. Kea assigns increasing integer values starting from 1. Thus, the autogenerated IDs are not stable across configuration changes.

8.5 Server Identifier in DHCPv4

The DHCPv4 protocol uses a "server identifier" to allow clients to discriminate between several servers present on the same link; this value is an IPv4 address of the server. The server chooses the IPv4 address of the interface on which the message from the client (or relay) has been received. A single server instance will use multiple server identifiers if it is receiving queries on multiple interfaces.

It is possible to override the default server identifier values by specifying the "dhcp-server-identifier" option. This option is only supported at the global, shared network, and subnet levels; it must not be specified on the client class or host reservation levels.

The following example demonstrates how to override the server identifier for a subnet:

```
"subnet4": [
    {
        "subnet": "192.0.2.0/24",
        "option-data": [
            {
                "name": "dhcp-server-identifier",
                "data": "10.2.5.76"
            }
        ],
        ...
    }
]
```

8.6 How the DHCPv4 Server Selects a Subnet for the Client

The DHCPv4 server differentiates between directly connected clients, clients trying to renew leases, and clients sending their messages through relays. For directly connected clients, the server will check the configuration for the interface on which the message has been received and, if the server configuration doesn't match any configured subnet, the message is discarded.

Assuming that the server's interface is configured with the IPv4 address 192.0.2.3, the server will only process messages received through this interface from a directly connected client if there is a subnet configured to which this IPv4 address belongs, such as 192.0.2.0/24. The server will use this subnet to assign an IPv4 address for the client.

The rule above does not apply when the client unicasts its message, i.e. is trying to renew its lease. Such a message is accepted through any interface. The renewing client sets ciaddr to the currently used IPv4 address, and the server uses this address to select the subnet for the client (in particular, to extend the lease using this address).

If the message is relayed it is accepted through any interface. The giaddr set by the relay agent is used to select the subnet for the client.

It is also possible to specify a relay IPv4 address for a given subnet. It can be used to match incoming packets into a subnet in uncommon configurations, e.g. shared networks. See *Using a Specific Relay Agent for a Subnet* for details.

Note: The subnet selection mechanism described in this section is based on the assumption that client classification is not used. The classification mechanism alters the way in which a subnet is selected for the client, depending on the classes to which the client belongs.

Note: Starting with Kea 1.7.9, the order used to find a subnet which matches required conditions to be selected is the ascending subnet identifier order. When the selected subnet is a member of a shared network the whole shared network is selected.

8.6.1 Using a Specific Relay Agent for a Subnet

A relay must have an interface connected to the link on which the clients are being configured. Typically the relay has an IPv4 address configured on that interface, which belongs to the subnet from which the server will assign addresses. Normally, the server is able to use the IPv4 address inserted by the relay (in the giaddr field of the DHCPv4 packet) to select the appropriate subnet.

However, that is not always the case. In certain uncommon — but valid — deployments, the relay address may not match the subnet. This usually means that there is more than one subnet allocated for a given link. The two most common examples where this is the case are long-lasting network renumbering (where both old and new address space is still being used) and a cable network. In a cable network, both cable modems and the devices behind them are physically connected to the same link, yet they use distinct addressing. In such a case, the DHCPv4 server needs additional information (the IPv4 address of the relay) to properly select an appropriate subnet.

The following example assumes that there is a subnet 192.0.2.0/24 that is accessible via a relay that uses 10.0.0.1 as its IPv4 address. The server is able to select this subnet for any incoming packets that come from a relay that has an address in the 192.0.2.0/24 subnet. It also selects that subnet for a relay with address 10.0.0.1.

```
"Dhcp4": {
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [ { "pool": "192.0.2.10 - 192.0.2.20" } ],
            "relay": {
                "ip-addresses": [ "10.0.0.1" ]
            },
            ...
        }
    ],
    ...
}
```

If “relay” is specified, the “ip-addresses” parameter within it is mandatory.

Note: The current version of Kea uses the “ip-addresses” parameter, which supports specifying a list of addresses.

8.6.2 Segregating IPv4 Clients in a Cable Network

In certain cases, it is useful to mix relay address information (introduced in *Using a Specific Relay Agent for a Subnet*), with client classification (explained in *Client Classification*). One specific example is in a cable network, where modems typically get addresses from a different subnet than all the devices connected behind them.

Let us assume that there is one CMTS (Cable Modem Termination System) with one CM MAC (a physical link that modems are connected to). We want the modems to get addresses from the 10.1.1.0/24 subnet, while everything connected behind the modems should get addresses from another subnet (192.0.2.0/24). The CMTS that acts as a relay uses address 10.1.1.1. The following configuration can serve that configuration:

```
"Dhcp4": {
    "subnet4": [
        {
            "subnet": "10.1.1.0/24",
            "pools":  [ { "pool": "10.1.1.2 - 10.1.1.20" } ],
            "client-class" "docsis3.0",
            "relay": {
                "ip-addresses": [ "10.1.1.1 ]"
            }
        },
        {
            "subnet": "192.0.2.0/24",
            "pools": [ { "pool": "192.0.2.10 - 192.0.2.20" } ],
            "relay": {
                "ip-addresses": [ "10.1.1.1" ]
            }
        }
    ],
    ...
}
```

8.7 Duplicate Addresses (DHCPDECLINE Support)

The DHCPv4 server is configured with a certain pool of addresses that it is expected to hand out to DHCPv4 clients. It is assumed that the server is authoritative and has complete jurisdiction over those addresses. However, for various reasons, such as misconfiguration or a faulty client implementation that retains its address beyond the valid lifetime, there may be devices connected that use those addresses without the server’s approval or knowledge.

Such an unwelcome event can be detected by legitimate clients (using ARP or ICMP Echo Request mechanisms) and reported to the DHCPv4 server using a DHCPDECLINE message. The server will do a sanity check (to see whether the client declining an address really was supposed to use it), and then will conduct a clean-up operation. Any DNS entries related to that address will be removed, the fact will be logged, and hooks will be triggered. After that is complete, the address will be marked as declined (which indicates that it is used by an unknown entity and thus not available for assignment) and a probation time will be set on it. Unless otherwise configured, the probation period lasts 24 hours; after that period, the server will recover the lease (i.e. put it back into the available state) and the address will be available for assignment again. It should be noted that if the underlying issue of a misconfigured device is not resolved, the duplicate-address scenario will repeat. If reconfigured correctly, this mechanism provides an opportunity to recover from such an event automatically, without any system administrator intervention.

To configure the decline probation period to a value other than the default, the following syntax can be used:

```
  "Dhcp4": {
    "decline-probation-period": 3600,
    "subnet4": [ ... ],
    ...
}
```

The parameter is expressed in seconds, so the example above will instruct the server to recycle declined leases after one hour.

There are several statistics and hook points associated with the Decline handling procedure. The lease4_decline hook is triggered after the incoming DHCPDECLINE message has been sanitized and the server is about to decline the lease. The declined-addresses statistic is increased after the hook returns (both global and subnet-specific variants). (See *Statistics in the DHCPv4 Server* and *Hooks Libraries* for more details on DHCPv4 statistics and Kea hook points.)

Once the probation time elapses, the declined lease is recovered using the standard expired-lease reclamation procedure, with several additional steps. In particular, both declined-addresses statistics (global and subnet-specific) are decreased. At the same time, reclaimed-declined-addresses statistics (again in two variants, global and subnet-specific) are increased.

A note about statistics: the server does not decrease the assigned-addresses statistics when a DHCPDECLINE is received and processed successfully. While technically a declined address is no longer assigned, the primary usage of the assigned-addresses statistic is to monitor pool utilization. Most people would forget to include declined-addresses in the calculation, and simply use assigned-addresses/total-addresses. This would cause a bias towards under-representing pool utilization. As this has a potential for major issues, ISC decided not to decrease assigned-addresses immediately after receiving DHCPDECLINE, but to do it later when Kea recovers the address back to the available pool.

8.8 Statistics in the DHCPv4 Server

The DHCPv4 server supports the following statistics:

Table 8.6: DHCPv4 Statistics

Statistic	Data Type	Description
pkt4-received	integer	Number of DHCPv4 packets received. This includes all packets: valid, bogus, corrupted, rejected, etc. This statistic is expected to grow rapidly.
pkt4-discover-received	integer	Number of DHCPDISCOVER packets received. This statistic is expected to grow; its increase means that clients that just booted started their configuration process and their initial packets reached the Kea server.

Continued on next page

Table 8.6 – continued from previous page

Statistic	Data Type	Description
pkt4-offer-received	integer	Number of DHCPOFFER packets received. This statistic is expected to remain zero at all times, as DHCPOFFER packets are sent by the server and the server is never expected to receive them. A non-zero value indicates an error. One likely cause would be a misbehaving relay agent that incorrectly forwards DHCPOFFER messages towards the server, rather than back to the clients.
pkt4-request-received	integer	Number of DHCPREQUEST packets received. This statistic is expected to grow. Its increase means that clients that just booted received the server's response (DHCPOFFER) and accepted it, and are now requesting an address (DHCPREQUEST).
pkt4-ack-received	integer	Number of DHCPACK packets received. This statistic is expected to remain zero at all times, as DHCPACK packets are sent by the server and the server is never expected to receive them. A non-zero value indicates an error. One likely cause would be a misbehaving relay agent that incorrectly forwards DHCPACK messages towards the server, rather than back to the clients.
pkt4-nak-received	integer	Number of DHCPNAK packets received. This statistic is expected to remain zero at all times, as DHCPNAK packets are sent by the server and the server is never expected to receive them. A non-zero value indicates an error. One likely cause would be a misbehaving relay agent that incorrectly forwards DHCPNAK messages towards the server, rather than back to the clients.
pkt4-release-received	integer	Number of DHCPRELEASE packets received. This statistic is expected to grow. Its increase means that clients that had an address are shutting down or ceasing to use their addresses.
pkt4-decline-received	integer	Number of DHCPDECLINE packets received. This statistic is expected to remain close to zero. Its increase means that a client leased an address, but discovered that the address is currently used by an unknown device in the network.
pkt4-inform-received	integer	Number of DHCPINFORM packets received. This statistic is expected to grow. Its increase means that there are clients that either do not need an address or already have an address and are interested only in getting additional configuration parameters.
pkt4-unknown-received	integer	Number of packets received of an unknown type. A non-zero value of this statistic indicates that the server received a packet that it wasn't able to recognize, either with an unsupported type or possibly malformed (without message type option).
pkt4-sent	integer	Number of DHCPv4 packets sent. This statistic is expected to grow every time the server transmits a packet. In general, it should roughly match pkt4-received, as most incoming packets cause the server to respond. There are exceptions (e.g. DHCPRELEASE), so do not worry if it is less than pkt4-received.
pkt4-offer-sent	integer	Number of DHCPOFFER packets sent. This statistic is expected to grow in most cases after a DHCPDISCOVER is processed. There are certain uncommon, but valid, cases where incoming DHCPDISCOVER packets are dropped, but in general this statistic is expected to be close to pkt4-discover-received.
pkt4-ack-sent	integer	Number of DHCPACK packets sent. This statistic is expected to grow in most cases after a DHCPREQUEST is processed. There are certain cases where DHCPNAK is sent instead. In general, the sum of pkt4-ack-sent and pkt4-nak-sent should be close to pkt4-request-received.
pkt4-nak-sent	integer	Number of DHCPNAK packets sent. This statistic is expected to grow when the server chooses not to honor the address requested by a client. In general, the sum of pkt4-ack-sent and pkt4-nak-sent should be close to pkt4-request-received.
pkt4-parse-failed	integer	Number of incoming packets that could not be parsed. A non-zero value of this statistic indicates that the server received a malformed or truncated packet. This may indicate problems in the network, faulty clients, or a bug in the server.

Continued on next page

Table 8.6 – continued from previous page

Statistic	Data Type	Description
pkt4-receive-drop	integer	Number of incoming packets that were dropped. The exact reason for dropping packets is logged, but the most common reasons may be: an unacceptable packet type, direct responses are forbidden, or the server-id sent by the client does not match the server's server-id.
subnet[id].total-addresses	integer	Total number of addresses available for DHCPv4 management; in other words, this is the sum of all addresses in all configured pools. This statistic changes only during configuration changes. Note it does not take into account any addresses that may be reserved due to host reservation. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.
cumulative-assigned-addresses	integer	Cumulative number of addresses that have been assigned since server startup. It is incremented each time an address is assigned and is not reset when the server is reconfigured.
subnet[id].cumulative-assigned-addresses	integer	Cumulative number of assigned addresses in a given subnet. It increases every time a new lease is allocated (as a result of receiving a DHCPREQUEST message) and never decreased. The *id* is the subnet-id of the subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.
subnet[id].assigned-addresses	integer	Number of assigned addresses in a given subnet. It increases every time a new lease is allocated (as a result of receiving a DHCPREQUEST message) and is decreased every time a lease is released (a DHCPRELEASE message is received) or expires. The *id* is the subnet-id of the subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.
reclaimed-leases	integer	Number of expired leases that have been reclaimed since server startup. It is incremented each time an expired lease is reclaimed. This statistic never decreases. It can be used as a long-term indicator of how many actual leases have been reclaimed. This is a global statistic that covers all subnets.
subnet[id].reclaimed-leases	integer	Number of expired leases associated with a given subnet (*id* is the subnet-id) that have been reclaimed since server startup. It is incremented each time an expired lease is reclaimed. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately.
declined-addresses	integer	Number of IPv4 addresses that are currently declined; a count of the number of leases currently unavailable. Once a lease is recovered, this statistic will be decreased; ideally, this statistic should be zero. If this statistic is non-zero or increasing, a network administrator should investigate whether there is a misbehaving device in the network. This is a global statistic that covers all subnets.
subnet[id].declined-addresses	integer	Number of IPv4 addresses that are currently declined in a given subnet; a count of the number of leases currently unavailable. Once a lease is recovered, this statistic will be decreased; ideally, this statistic should be zero. If this statistic is non-zero or increasing, a network administrator should investigate whether there is a misbehaving device in the network. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately.
reclaimed-declined-addresses	integer	Number of IPv4 addresses that were declined, but have now been recovered. Unlike declined-addresses, this statistic never decreases. It can be used as a long-term indicator of how many actual valid Declines were processed and recovered from. This is a global statistic that covers all subnets.
subnet[id].reclaimed-declined-addresses	integer	Number of IPv4 addresses that were declined, but have now been recovered. Unlike declined-addresses, this statistic never decreases. It can be used as a long-term indicator of how many actual valid Declines were processed and recovered from. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately.

Continued on next page

Table 8.6 – continued from previous page

Statistic	Data Type	Description
pkt4-lease-query-received	integer	Number of IPv4 DHCPLEASEQUERY packets received. (Only exists if Leasequery hook library is loaded.)
pkt4-lease-query-response-unknown-sent	integer	Number of IPv4 DHCPLEASEUNKNOWN responses sent. (Only exists if Leasequery hook library is loaded.)
pkt4-lease-query-response-unassigned-sent	integer	Number of IPv4 DHCPLEASEUNASSIGNED responses sent. (Only exists if Leasequery hook library is loaded.)
pkt4-lease-query-response-active-sent	integer	Number of IPv4 DHCPLEASEACTIVE responses sent. (Only exists if Leasequery hook library is loaded.)

Note: This section describes DHCPv4-specific statistics. For a general overview and usage of statistics, see *Statistics*.

Beginning with Kea 1.7.7 the DHCPv4 server provides two global parameters to control statistics default sample limits:

- `statistic-default-sample-count` - determines the default maximum number of samples which will be kept. The special value of zero means to use a default maximum age.
- `statistic-default-sample-age` - determines the default maximum age in seconds of samples which will be kept.

For instance to reduce the statistic keeping overhead you can set the default maximum sample count to 1 so only one sample will be kept by:

```
  "Dhcp4": {
    "statistic-default-sample-count": 1,
    "subnet4": [ ... ],
    ...
}
```

Statistics can be retrieved periodically to gain more insight into Kea operations. One tool that leverages that capability is ISC Stork. See *Monitoring Kea with Stork* for details.

8.9 Management API for the DHCPv4 Server

The management API allows the issuing of specific management commands, such as statistics retrieval, reconfiguration, or shutdown. For more details, see *Management API*. Currently, the only supported communication channel type is UNIX stream socket. By default there are no sockets open; to instruct Kea to open a socket, the following entry in the configuration file can be used:

```
"Dhcp4": {
    "control-socket": {
        "socket-type": "unix",
        "socket-name": "/path/to/the/unix/socket"
    },

    "subnet4": [
        ...
    ],
    ...
}
```

The length of the path specified by the `socket-name` parameter is restricted by the maximum length for the UNIX socket name on the administrator's operating system, i.e. the size of the `sun_path` field in the `sockaddr_un` structure, decreased by 1. This value varies on different operating systems between 91 and 107 characters. Typical values are 107 on Linux and 103 on FreeBSD.

Communication over the control channel is conducted using JSON structures. See the Control Channel section in the Kea Developer's Guide for more details.

The DHCPv4 server supports the following operational commands:

- build-report
- config-get
- config-reload
- config-set
- config-test
- config-write
- dhcp-disable
- dhcp-enable
- leases-reclaim
- list-commands
- shutdown
- status-get
- version-get

as described in *Commands Supported by Both the DHCPv4 and DHCPv6 Servers*. In addition, it supports the following statistics-related commands:

- statistic-get
- statistic-reset
- statistic-remove
- statistic-get-all
- statistic-reset-all
- statistic-remove-all
- statistic-sample-age-set
- statistic-sample-age-set-all
- statistic-sample-count-set
- statistic-sample-count-set-all

as described in *Commands for Manipulating Statistics*.

8.10 User Contexts in IPv4

Kea allows loading hook libraries that can sometimes benefit from additional parameters. If such a parameter is specific to the whole library, it is typically defined as a parameter for the hook library. However, sometimes there is a need to specify parameters that are different for each pool.

See *Comments and User Context* for additional background regarding the user context idea. See *User Contexts in Hooks* for a discussion from the hooks perspective.

User contexts can be specified at global scope, shared network, subnet, pool, client class, option data, or definition level, and via host reservation. One other useful feature is the ability to store comments or descriptions.

Let's consider an imaginary case of devices that have colored LED lights. Depending on their location, they should glow red, blue, or green. It would be easy to write a hook library that would send specific values as maybe a vendor option. However, the server has to have some way to specify that value for each pool. This need is addressed by user contexts. In essence, any user data can be specified in the user context as long as it is a valid JSON map. For example, the aforementioned case of LED devices could be configured in the following way:

```
"Dhcp4": {
    "subnet4": [{
        "subnet": "192.0.2.0/24",
        "pools": [{
            "pool": "192.0.2.10 - 192.0.2.20",
            # This is pool specific user context
            "user-context": { "color": "red" }
        }],

        # This is a subnet-specific user context. Any type
        # of information can be entered here as long as it is valid JSON.
        "user-context": {
            "comment": "network on the second floor",
            "last-modified": "2017-09-04 13:32",
            "description": "you can put anything you like here",
            "phones": [ "x1234", "x2345" ],
            "devices-registered": 42,
            "billing": false
        }
    }],
}
```

Kea does not interpret or use the user context information; it simply stores it and makes it available to the hook libraries. It is up to each hook library to extract that information and use it. The parser translates a "comment" entry into a user context with the entry, which allows a comment to be attached inside the configuration itself.

8.11 Supported DHCP Standards

The following standards are currently supported:

- *BOOTP Vendor Information Extensions*, RFC 1497: This requires the open source BOOTP hook to be loaded.
- *Dynamic Host Configuration Protocol*, RFC 2131: Supported messages are DHCPDISCOVER (1), DHCPOFFER (2), DHCPREQUEST (3), DHCPRELEASE (7), DHCPINFORM (8), DHCPACK (5), and DHCPNAK(6).
- *DHCP Options and BOOTP Vendor Extensions*, RFC 2132: Supported options are: PAD (0), END(255), Message Type(53), DHCP Server Identifier (54), Domain Name (15), DNS Servers (6), IP Address Lease Time (51), Subnet mask (1), and Routers (3).

- *DHCP Relay Agent Information Option*, RFC 3046: Relay Agent Information option is supported.
- *Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol version 4*, RFC 3925: Vendor-Identifying Vendor Class and Vendor-Identifying Vendor-Specific Information options are supported.
- *The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option*, RFC 4702: The Kea server is able to handle the Client FQDN option. Also, it is able to use kea-dhcp-ddns component do initiate appropriate DNS Update operations.
- *Resolution of Fully Qualified Domain Name (FQDN) Conflicts among Dynamic Host Configuration Protocol (DHCP) Clients*, RFC 4703: The DHCPv6 server uses DHCP-DDNS server to resolve conflicts.
- *Client Identifier Option in DHCP Server Replies*, RFC 6842: Server by default sends back client-id option. That capability may be disabled. See *Echoing Client-ID (RFC 6842)* for details.
- *Generalized UDP Source Port for DHCP Relay*, RFC 8357: The Kea server is able to handle Relay Agent Information Source Port suboption in a received message, remembers the UDP port and sends back reply to the same relay agent using this UDP port.
- *IPv6-Only-Preferred Option for DHCPv4*, draft-ietf-dhc-v6only-08: The Kea server is able to designate its pools and subnets as v6only-preferred and send back the *v6-only-preferred* option to clients that requested it.

8.12 DHCPv4 Server Limitations

These are the current limitations of the DHCPv4 server software. Most of them are reflections of the current stage of development and should be treated as “not implemented yet,” rather than as actual limitations. However, some of them are implications of the design choices made. Those are clearly marked as such.

- BOOTP (RFC 951) is not supported. This is a design choice; historic BOOTP support is not planned.
- BOOTP with vendor information extensions (RFC 1497) is supported by the BOOTP hooks library; see *BOOTP support* for details.
- On Linux and BSD system families the DHCP messages are sent and received over the raw sockets (using LPF and BPF) and all packet headers (including data link layer, IP, and UDP headers) are created and parsed by Kea, rather than by the system kernel. Currently, Kea can only parse the data link layer headers with a format adhering to the IEEE 802.3 standard and assumes this data link layer header format for all interfaces. Thus, Kea will fail to work on interfaces which use different data link layer header formats (e.g. Infiniband).
- The DHCPv4 server does not verify that an assigned address is unused. According to RFC 2131, the allocating server should verify that an address is not used by sending an ICMP echo request.

8.13 Kea DHCPv4 Server Examples

A collection of simple-to-use examples for the DHCPv4 component of Kea is available with the source files, located in the doc/examples/kea4 directory.

8.14 Configuration Backend in DHCPv4

In the *Kea Configuration Backend* section we have described the Configuration Backend feature, its applicability, and its limitations. This section focuses on the usage of the CB with the DHCPv4 server. It lists the supported parameters, describes limitations, and gives examples of the DHCPv4 server configuration to take advantage of the CB. Please also refer to the sibling section *Configuration Backend in DHCPv6* for the DHCPv6-specific usage of the CB.

8.14.1 Supported Parameters

The ultimate goal for the CB is to serve as a central configuration repository for one or multiple Kea servers connected to the database. In the future it will be possible to store most of the server's configuration in the database and reduce the configuration file to a bare minimum; the only mandatory parameter will be `config-control`, which includes the necessary information to connect to the database. In the Kea 1.6.0 release, however, only a subset of the DHCPv4 server parameters can be stored in the database. All other parameters must be specified in the JSON configuration file, if required.

The following table lists DHCPv4 specific parameters supported by the Configuration Backend, with an indication on which level of the hierarchy it is currently supported. "n/a" is used in cases when a given parameter is not applicable on a particular level of the hierarchy, or in cases when the parameter is not supported by the server at this level of the hierarchy. "no" is used when the parameter is supported by the server on the given level of the hierarchy, but is not configurable via the Configuration Backend.

All supported parameters can be configured via the `cb_cmds` hooks library described in the *cb_cmds: Configuration Backend Commands* section. The general rule is that the scalar global parameters are set using `remote-global-parameter4-set`; the shared network-specific parameters are set using `remote-network4-set`; and the subnet- and pool-level parameters are set using `remote-subnet4-set`. Whenever there is an exception to this general rule, it is highlighted in the table. The non-scalar global parameters have dedicated commands; for example, the global DHCPv4 options (`option-data`) are modified using `remote-option4-global-set`.

The *Configuration Sharing and Server Tags* explains the concept of shareable and non-shareable configuration elements and the limitations for sharing them between multiple servers. In the DHCP configuration (both DHCPv4 and DHCPv6) the shareable configuration elements are: subnets and shared networks. Thus, they can be explicitly associated with multiple server tags. The global parameters, option definitions and global options are non-shareable and they can be associated with only one server tag. This rule does not apply to the configuration elements associated with "all" servers. Any configuration element associated with "all" servers (using "all" keyword as a server tag) is used by all servers connecting to the configuration database.

Table 8.7: List of DHCPv4 Parameters Supported by the Configuration Backend

Parameter	Global	Shared Network	Subnet	Pool
4o6-interface	n/a	n/a	yes	n/a
4o6-interface-id	n/a	n/a	yes	n/a
4o6-subnet	n/a	n/a	yes	n/a
boot-file-name	yes	yes	yes	n/a
calculate-tee-times	yes	yes	yes	n/a
client-class	n/a	yes	yes	yes
ddns-send-update	yes	yes	yes	n/a
ddns-override-no-update	yes	yes	yes	n/a
ddns-override-client-update	yes	yes	yes	n/a
ddns-replace-client-name	yes	yes	yes	n/a
ddns-generated-prefix	yes	yes	yes	n/a
ddns-qualifying-suffix	yes	yes	yes	n/a
decline-probation-period	yes	n/a	n/a	n/a
dhcp4o6-port	yes	n/a	n/a	n/a
echo-client-id	yes	n/a	n/a	n/a
hostname-char-set	no	no	no	n/a
hostname-char-replacement	no	no	no	n/a
interface	n/a	yes	yes	n/a
match-client-id	yes	yes	yes	n/a

Continued on next page

Table 8.7 – continued from previous page

Parameter	Global	Shared Network	Subnet	Pool
next-server	yes	yes	yes	n/a
option-data	yes (via remote-option4-global-set)	yes	yes	yes
option-def	yes (via remote-option-def4-set)	n/a	n/a	n/a
rebind-timer	yes	yes	yes	n/a
renew-timer	yes	yes	yes	n/a
server-hostname	yes	yes	yes	n/a
valid-lifetime	yes	yes	yes	n/a
relay	n/a	yes	yes	n/a
require-client-classes	no	yes	yes	yes
reservation-mode	yes	yes	yes	n/a
t1-percent	yes	yes	yes	n/a
t2-percent	yes	yes	yes	n/a

8.14.2 Enabling Configuration Backend

Consider the following configuration snippet:

```
"Dhcp4": {
    "server-tag": "my DHCPv4 server",
    "config-control": {
        "config-databases": [{
            "type": "mysql",
            "name": "kea",
            "user": "kea",
            "password": "kea",
            "host": "192.0.2.1",
            "port": 3302
        }],
        "config-fetch-wait-time": 20
    },
    "hooks-libraries": [{
        "library": "/usr/local/lib/kea/hooks/libdhcp_mysql_cb.so"
    }, {
        "library": "/usr/local/lib/kea/hooks/libdhcp_cb_cmds.so"
    }],
}
```

The `config-control` command contains two parameters. `config-databases` is a list which contains one element comprising database type, location, and the credentials to be used to connect to this database. (Note that the parameters specified here correspond to the database specification for the lease database backend and hosts database backend.) Currently only one database connection can be specified on the `config-databases` list. The server will connect to this database during the startup or reconfiguration, and will fetch the configuration available for this server from the database. This configuration is merged into the configuration read from the configuration file.

Note: Whenever there is a conflict between the parameters specified in the configuration file and the database, the parameters from the database take precedence. We strongly recommend avoiding the duplication of parameters in the file and the database, but this recommendation is not enforced by the Kea servers. In particular, if the subnets' configuration is sourced from the database, we recommend that all subnets be specified in the database and that no subnets be specified in the configuration file. It is possible to specify the subnets in both places, but the subnets in the configuration file with overlapping ids and/or prefixes with the subnets from the database will be superseded by those from the database.

Once the Kea server is configured, it starts periodically polling for the configuration changes in the database. The frequency of polling is controlled by the `config-fetch-wait-time` parameter, expressed in seconds; it is the period between the time when the server completed last polling (and possibly the local configuration update) and the time when it will begin polling again. In the example above, this period is set to 20 seconds. This means that after adding a new configuration into the database (e.g. adding new subnet), it will take up to 20 seconds (plus the time needed to fetch and apply the new configuration) before the server starts using this subnet. The lower the `config-fetch-wait-time` value, the shorter the time for the server to react to the incremental configuration updates in the database. On the other hand, polling the database too frequently may impact the DHCP server's performance, because the server needs to make at least one query to the database to discover the pending configuration updates. The default value of the `config-fetch-wait-time` is 30 seconds.

The `config-backend-pull` command can be used to force the server to immediately poll the configuration changes from the database and avoid waiting for the next fetch cycle. The command was added in 1.7.1 Kea release for DHCPv4 and DHCPv6 servers.

Finally, in the configuration example above, two hooks libraries are loaded. The first, `libdhcp_mysql_cb.so`, is the implementation of the Configuration Backend for MySQL. It must be always present when the server uses MySQL as the configuration repository. Failing to load this library will result in an error during the server configuration if the "mysql" database is selected with the `config-control` parameter.

The second hooks library, `libdhcp_cb_cmds.so`, is optional. It should be loaded when the Kea server instance is to be used for managing the configuration in the database. See the *cb_cmds: Configuration Backend Commands* section for details. Note that this hooks library is only available to ISC customers with a support contract.

CHAPTER

NINE

THE DHCPV6 SERVER

9.1 Starting and Stopping the DHCPv6 Server

It is recommended that the Kea DHCPv6 server be started and stopped using `keactrl` (described in *Managing Kea with keactrl*); however, it is also possible to run the server directly. It accepts the following command-line switches:

- `-c file` - specifies the configuration file. This is the only mandatory switch.
- `-d` - specifies whether the server logging should be switched to debug/verbose mode. In verbose mode, the logging severity and debuglevel specified in the configuration file are ignored; "debug" severity and the maximum debuglevel (99) are assumed. The flag is convenient for temporarily switching the server into maximum verbosity, e.g. when debugging.
- `-p server-port` - specifies the local UDP port on which the server will listen. This is only useful during testing, as a DHCPv6 server listening on ports other than the standard ones will not be able to handle regular DHCPv6 queries.
- `-P client-port` - specifies the remote UDP port to which the server will send all responses. This is only useful during testing, as a DHCPv6 server sending responses to ports other than the standard ones will not be able to handle regular DHCPv6 queries.
- `-t file` - specifies a configuration file to be tested. Kea-dhcp6 will load it, check it, and exit. During the test, log messages are printed to standard output and error messages to standard error. The result of the test is reported through the exit code (0 = configuration looks ok, 1 = error encountered). The check is not comprehensive; certain checks are possible only when running the server.
- `-v` - displays the Kea version and exits.
- `-V` - displays the Kea extended version with additional parameters and exits. The listing includes the versions of the libraries dynamically linked to Kea.
- `-W` - displays the Kea configuration report and exits. The report is a copy of the `config.report` file produced by `./configure`; it is embedded in the executable binary.

On startup, the server will detect available network interfaces and will attempt to open UDP sockets on all interfaces mentioned in the configuration file. Since the DHCPv6 server opens privileged ports, it requires root access. This daemon must be run as root.

During startup, the server will attempt to create a PID file of the form: [**runstatedir**]/kea/[**conf name**].kea-dhcp6.pid where:

- `runstatedir`: The value as passed into the build configure script; it defaults to "/usr/local/var/run". Note that this value may be overridden at runtime by setting the environment variable KEA_PIDFILE_DIR, although this is intended primarily for testing purposes.
- `conf name`: The configuration file name used to start the server, minus all preceding paths and the file extension. For example, given a pathname of "/usr/local/etc/kea/myconf.txt", the portion used would be "myconf".

If the file already exists and contains the PID of a live process, the server will issue a DHCP6_ALREADY_RUNNING log message and exit. It is possible, though unlikely, that the file is a remnant of a system crash and the process to which the PID belongs is unrelated to Kea. In such a case it would be necessary to manually delete the PID file.

The server can be stopped using the `kill` command. When running in a console, the server can also be shut down by pressing ctrl-c. It detects the key combination and shuts down gracefully.

9.2 DHCPv6 Server Configuration

9.2.1 Introduction

This section explains how to configure the DHCPv6 server using a configuration file. Before DHCPv6 is started, its configuration file must be created. The basic configuration is as follows:

```
{
# DHCPv6 configuration starts on the next line
"Dhcp6": {

# First we set up global values
    "valid-lifetime": 4000,
    "renew-timer": 1000,
    "rebind-timer": 2000,
    "preferred-lifetime": 3000,

# Next we set up the interfaces to be used by the server.
    "interfaces-config": {
        "interfaces": [ "eth0" ]
    },

# And we specify the type of lease database
    "lease-database": {
        "type": "memfile",
        "persist": true,
        "name": "/var/lib/kea/dhcp6.leases"
    },

# Finally, we list the subnets from which we will be leasing addresses.
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",
            "pools": [
                 {
                     "pool": "2001:db8:1::1-2001:db8:1::ffff"
                 }
             ]
        }
    ]
# DHCPv6 configuration ends with the next line
}

}
```

The following paragraphs provide a brief overview of the parameters in the above example, along with their format. Subsequent sections of this chapter go into much greater detail for these and other parameters.

The lines starting with a hash (#) are comments and are ignored by the server; they do not impact its operation in any way.

The configuration starts in the first line with the initial opening curly bracket (or brace). Each configuration must contain an object specifying the configuration of the Kea module using it. In the example above this object is called `Dhcp6`.

Note: In the current Kea release it is possible to specify configurations of multiple modules within a single configuration file, but this is not recommended and support for it was removed in 1.7.10 release, including the `Logging` object: its previous content, the list of loggers, must now be inside the `Dhcp6` object.

The Dhcp6 configuration starts with the `"Dhcp6": {` line and ends with the corresponding closing brace (in the above example, the brace after the last comment). Everything defined between those lines is considered to be the Dhcp6 configuration.

In general, the order in which those parameters appear does not matter, but there are two caveats. The first one is to remember that the configuration file must be well-formed JSON. That means that the parameters for any given scope must be separated by a comma, and there must not be a comma after the last parameter. When reordering a configuration file, keep in mind that moving a parameter to or from the last position in a given scope may also require moving the comma. The second caveat is that it is uncommon — although legal JSON — to repeat the same parameter multiple times. If that happens, the last occurrence of a given parameter in a given scope is used, while all previous instances are ignored. This is unlikely to cause any confusion as there are no real-life reasons to keep multiple copies of the same parameter in the configuration file.

The first few DHCPv6 configuration elements define some global parameters. `valid-lifetime` defines how long the addresses (leases) given out by the server are valid. If nothing changes, a client that got an address is allowed to use it for 4000 seconds. (Note that integer numbers are specified as is, without any quotes around them.) The address will become deprecated in 3000 seconds, i.e. clients are allowed to keep old connections, but can't use this address for creating new connections. `renew-timer` and `rebind-timer` are values (also in seconds) that define T1 and T2 timers that govern when the client will begin the renewal and rebind procedures.

The `interfaces-config` map specifies the server configuration concerning the network interfaces on which the server should listen to the DHCP messages. The `interfaces` parameter specifies a list of network interfaces on which the server should listen. Lists are opened and closed with square brackets, with elements separated by commas. To listen on two interfaces, the `interfaces-config` should look like this:

```
"interfaces-config": {
    "interfaces": [ "eth0", "eth1" ]
},
```

The next couple of lines define the lease database, the place where the server stores its lease information. This particular example tells the server to use `memfile`, which is the simplest (and fastest) database backend. It uses an in-memory database and stores leases on disk in a CSV (comma-separated values) file. This is a very simple configuration; usually the lease database configuration is more extensive and contains additional parameters. Note that `lease-database` is an object and opens up a new scope, using an opening brace. Its parameters (just one in this example: `type`) follow. If there were more than one, they would be separated by commas. This scope is closed with a closing brace. As more parameters for the Dhcp6 definition follow, a trailing comma is present.

Finally, we need to define a list of IPv6 subnets. This is the most important DHCPv6 configuration structure, as the server uses that information to process clients' requests. It defines all subnets from which the server is expected to receive DHCP requests. The subnets are specified with the `subnet6` parameter. It is a list, so it starts and ends with square brackets. Each subnet definition in the list has several attributes associated with it, so it is a structure and is opened and closed with braces. At a minimum, a subnet definition has to have at least two parameters: `subnet` (which defines the whole subnet) and `pools` (which is a list of dynamically allocated pools that are governed by the DHCP server).

The example contains a single subnet. If more than one were defined, additional elements in the `subnet6` parameter would be specified and separated by commas. For example, to define two subnets, the following syntax would be used:

```
"subnet6": [
    {
        "pools": [ { "pool": "2001:db8:1::/112" } ],
        "subnet": "2001:db8:1::/64"
    },
    {
        "pools": [ { "pool": "2001:db8:2::1-2001:db8:2::ffff" } ],
        "subnet": "2001:db8:2::/64"
    }
]
```

Note that indentation is optional and is used for aesthetic purposes only. In some cases in may be preferable to use more compact notation.

After all the parameters are specified, we have two contexts open: global and Dhcp6; thus, we need two closing curly brackets to close them.

9.2.2 Lease Storage

All leases issued by the server are stored in the lease database. Currently there are four database backends available: memfile (which is the default backend), MySQL, PostgreSQL, and Cassandra.

Memfile - Basic Storage for Leases

The server is able to store lease data in different repositories. Larger deployments may elect to store leases in a database. *Lease Database Configuration* describes this option. In typical smaller deployments, though, the server will store lease information in a CSV file rather than a database. As well as requiring less administration, an advantage of using a file for storage is that it eliminates a dependency on third-party database software.

The configuration of the file backend (memfile) is controlled through the Dhcp6/lease-database parameters. The `type` parameter is mandatory and it specifies which storage for leases the server should use. The value of `"memfile"` indicates that the file should be used as the storage. The following list gives additional optional parameters that can be used to configure the memfile backend.

- `persist`: controls whether the new leases and updates to existing leases are written to the file. It is strongly recommended that the value of this parameter be set to `true` at all times during the server's normal operation. Not writing leases to disk means that if a server is restarted (e.g. after a power failure), it will not know which addresses have been assigned. As a result, it may assign new clients addresses that are already in use. The value of `false` is mostly useful for performance-testing purposes. The default value of the `persist` parameter is `true`, which enables writing lease updates to the lease file.
- `name`: specifies an absolute location of the lease file in which new leases and lease updates will be recorded. The default value for this parameter is `"[kea-install-dir]/var/lib/kea/kea-leases6.csv"`.
- `lfc-interval`: specifies the interval, in seconds, at which the server will perform a lease file cleanup (LFC). This removes redundant (historical) information from the lease file and effectively reduces the lease file size. The cleanup process is described in more detail later in this section. The default value of the `lfc-interval` is `3600`. A value of 0 disables the LFC.
- `max-row-errors`: when the server loads a lease file, it is processed row by row, each row contaning a single lease. If a row is flawed and cannot be processed correctly the server will log it, discard the row, and go on to the next row. This parameter can be used to set a limit on the number of such discards that may occur after which the server will abandon the effort and exit. The default value of 0 disables the limit and allows the server to process the entire file, regardless of how many rows are discarded.

An example configuration of the memfile backend is presented below:

```
"Dhcp6": {
    "lease-database": {
        "type": "memfile",
        "persist": true,
        "name": "/tmp/kea-leases6.csv",
        "lfc-interval": 1800,
        "max-row-errors": 100
    }
}
```

This configuration selects the `/tmp/kea-leases6.csv` as the storage for lease information and enables persistence (writing lease updates to this file). It also configures the backend to perform a periodic cleanup of the lease file every 30 minutes and sets the maximum number of row errors to 100.

It is important to know how the lease file contents are organized to understand why the periodic lease file cleanup is needed. Every time the server updates a lease or creates a new lease for the client, the new lease information must be recorded in the lease file. For performance reasons, the server does not update the existing client's lease in the file, as this would potentially require rewriting the entire file. Instead, it simply appends the new lease information to the end of the file; the previous lease entries for the client are not removed. When the server loads leases from the lease file, e.g. at the server startup, it assumes that the latest lease entry for the client is the valid one. The previous entries are discarded, meaning that the server can re-construct the accurate information about the leases even though there may be many lease entries for each client. However, storing many entries for each client results in a bloated lease file and impairs the performance of the server's startup and reconfiguration, as it needs to process a larger number of lease entries.

Lease file cleanup (LFC) removes all previous entries for each client and leaves only the latest ones. The interval at which the cleanup is performed is configurable, and it should be selected according to the frequency of lease renewals initiated by the clients. The more frequent the renewals, the smaller the value of `lfc-interval` should be. Note, however, that the LFC takes time and thus it is possible (although unlikely) that, if the `lfc-interval` is too short, a new cleanup may be started while the previous one is still running. The server would recover from this by skipping the new cleanup when it detected that the previous cleanup was still in progress. But it implies that the actual cleanups will be triggered more rarely than configured. Moreover, triggering a new cleanup adds overhead to the server, which will not be able to respond to new requests for a short period of time when the new cleanup process is spawned. Therefore, it is recommended that the `lfc-interval` value be selected in a way that allows the LFC to complete the cleanup before a new cleanup is triggered.

Lease file cleanup is performed by a separate process (in the background) to avoid a performance impact on the server process. To avoid conflicts between two processes both using the same lease files, the LFC process starts with Kea opening a new lease file; the actual LFC process operates on the lease file that is no longer used by the server. There are also other files created as a side effect of the lease file cleanup. The detailed description of the LFC process is located later in this Kea Administrator's Reference Manual: *The LFC Process*.

Lease Database Configuration

Note: Lease database access information must be configured for the DHCPv6 server, even if it has already been configured for the DHCPv4 server. The servers store their information independently, so each server can use a separate database or both servers can use the same database.

Lease database configuration is controlled through the Dhcp6/lease-database parameters. The database type must be set to "memfile", "mysql", "postgresql", or "cql", e.g.:

```
"Dhcp6": { "lease-database": { "type": "mysql", ... }, ... }
```

Next, the name of the database to hold the leases must be set; this is the name used when the database was created (see *First-Time Creation of the MySQL Database*, *First-Time Creation of the PostgreSQL Database*, or *First-Time Creation of the Cassandra Database*).

```
"Dhcp6": { "lease-database": { "name": "database-name" , ... }, ... }
```

For Cassandra:

```
"Dhcp6": { "lease-database": { "keyspace": "database-name" , ... }, ... }
```

If the database is located on a different system from the DHCPv6 server, the database host name must also be specified:

```
"Dhcp6": { "lease-database": { "host": "remote-host-name", ... }, ... }
```

(It should be noted that this configuration may have a severe impact on server performance.)

For Cassandra, multiple contact points can be provided:

```
"Dhcp6": { "lease-database": { "contact-points": "remote-host-name[, ...]" , ... }, ..
↪. }
```

Normally, the database will be on the same machine as the DHCPv6 server. In this case, set the value to the empty string:

```
"Dhcp6": { "lease-database": { "host" : "", ... }, ... }
```

For Cassandra:

```
"Dhcp6": { "lease-database": { "contact-points": "", ... }, ... }
```

Should the database use a port other than the default, it may be specified as well:

```
"Dhcp6": { "lease-database": { "port" : 12345, ... }, ... }
```

Should the database be located on a different system, the administrator may need to specify a longer interval for the connection timeout:

```
"Dhcp6": { "lease-database": { "connect-timeout" : timeout-in-seconds, ... }, ... }
```

The default value of five seconds should be more than adequate for local connections. If a timeout is given, though, it should be an integer greater than zero.

The maximum number of times the server will automatically attempt to reconnect to the lease database after connectivity has been lost may be specified:

```
"Dhcp6": { "lease-database": { "max-reconnect-tries" : number-of-tries, ... }, ... }
```

If the server is unable to reconnect to the database after making the maximum number of attempts, the server will exit. A value of zero (the default) disables automatic recovery and the server will exit immediately upon detecting a loss of connectivity (MySQL and PostgreSQL only).

The number of milliseconds the server will wait between attempts to reconnect to the lease database after connectivity has been lost may also be specified:

```
"Dhcp6": { "lease-database": { "reconnect-wait-time" : number-of-milliseconds, ... },␣
↪... }
```

The default value for MySQL and PostgreSQL is 0, which disables automatic recovery and causes the server to exit immediately upon detecting the loss of connectivity. The default value for Cassandra is 2000 ms.

Note: Automatic reconnection to database backends is configured individually per backend. This allows users to tailor the recovery parameters to each backend they use. We do suggest that users enable it either for all backends or none, so behavior is consistent. Losing connectivity to a backend for which reconnect is disabled will result in the server shutting itself down. This includes cases when the lease database backend and the hosts database backend are connected to the same database instance.

Note: Note that the host parameter is used by the MySQL and PostgreSQL backends. Cassandra has a concept of contact points that can be used to contact the cluster, instead of a single IP or hostname. It takes a list of comma-separated IP addresses, which may be specified as:

```
"Dhcp6": { "lease-database": { "contact-points" : "192.0.2.1,192.0.2.2", ... }, ... }
```

Finally, the credentials of the account under which the server will access the database should be set:

```
"Dhcp6": { "lease-database": { "user": "user-name",
                               "password": "password",
                              ... },
           ... }
```

If there is no password to the account, set the password to the empty string “”. (This is also the default.)

Cassandra-Specific Parameters

The parameters are the same for both DHCPv4 and DHCPv6. See *Cassandra-Specific Parameters* for details.

9.2.3 Hosts Storage

Kea is also able to store information about host reservations in the database. The hosts database configuration uses the same syntax as the lease database. In fact, a Kea server opens independent connections for each purpose, be it lease or hosts information. This arrangement gives the most flexibility. Kea can keep leases and host reservations separately, but can also point to the same database. Currently the supported hosts database types are MySQL, PostgreSQL, and Cassandra.

For example, the following configuration can be used to configure a connection to MySQL:

```
"Dhcp6": {
    "hosts-database": {
        "type": "mysql",
        "name": "kea",
        "user": "kea",
        "password": "secret123",
        "host": "localhost",
        "port": 3306
    }
}
```

Note that depending on the database configuration, many of the parameters may be optional.

Please note that usage of hosts storage is optional. A user can define all host reservations in the configuration file, and that is the recommended way if the number of reservations is small. However, when the number of reservations grows, it is more convenient to use host storage. Please note that both storage methods (configuration file and one of the supported databases) can be used together. If hosts are defined in both places, the definitions from the configuration file are checked first and external storage is checked later, if necessary.

In fact, host information can be placed in multiple stores. Operations are performed on the stores in the order they are defined in the configuration file, although this leads to a restriction in ordering in the case of a host reservation addition; read-only stores must be configured after a (required) read-write store, or the addition will fail.

DHCPv6 Hosts Database Configuration

Hosts database configuration is controlled through the Dhcp6/hosts-database parameters. If enabled, the type of database must be set to “mysql” or “postgresql”.

```
"Dhcp6": { "hosts-database": { "type": "mysql", ... }, ... }
```

Next, the name of the database to hold the reservations must be set; this is the name used when the lease database was created (see *Supported Backends* for instructions on how to set up the desired database type):

```
"Dhcp6": { "hosts-database": { "name": "database-name" , ... }, ... }
```

If the database is located on a different system than the DHCPv6 server, the database host name must also be specified:

```
"Dhcp6": { "hosts-database": { "host": remote-host-name, ... }, ... }
```

(Again, it should be noted that this configuration may have a severe impact on server performance.)

Normally, the database will be on the same machine as the DHCPv6 server. In this case, set the value to the empty string:

```
"Dhcp6": { "hosts-database": { "host" : "", ... }, ... }
```

```
"Dhcp6": { "hosts-database": { "port" : 12345, ... }, ... }
```

The maximum number of times the server will automatically attempt to reconnect to the host database after connectivity has been lost may be specified:

```
"Dhcp6": { "host-database": { "max-reconnect-tries" : number-of-tries, ... }, ... }
```

If the server is unable to reconnect to the database after making the maximum number of attempts, the server will exit. A value of zero (the default) disables automatic recovery and the server will exit immediately upon detecting a loss of connectivity (MySQL and PostgreSQL only). For Cassandra, Kea uses a Cassandra interface that connects to all nodes in a cluster at the same time. Any connectivity issues should be handled by internal Cassandra mechanisms.

The number of milliseconds the server will wait between attempts to reconnect to the host database after connectivity has been lost may also be specified:

```
"Dhcp6": { "hosts-database": { "reconnect-wait-time" : number-of-milliseconds, ... },␣
↪... }
```

The default value for MySQL and PostgreSQL is 0, which disables automatic recovery and causes the server to exit immediately upon detecting the loss of connectivity. The default value for Cassandra is 2000 ms.

Note: Automatic reconnection to database backends is configured individually per backend. This allows users to tailor the recovery parameters to each backend they use. We do suggest that users enable it either for all backends or none, so behavior is consistent. Losing connectivity to a backend for which reconnect is disabled will result in the server shutting itself down. This includes cases when the lease database backend and the hosts database backend are connected to the same database instance.

Finally, the credentials of the account under which the server will access the database should be set:

```
"Dhcp6": { "hosts-database": { "user": "user-name",
                               "password": "password",
                              ... },
           ... }
```

If there is no password to the account, set the password to the empty string "". (This is also the default.)

The multiple storage extension uses a similar syntax; a configuration is placed into a "hosts-databases" list instead of into a "hosts-database" entry, as in:

```
"Dhcp6": { "hosts-databases": [ { "type": "mysql", ... }, ... ], ... }
```

For additional Cassandra-specific parameters, see *Cassandra-Specific Parameters*.

Using Read-Only Databases for Host Reservations with DHCPv6

In some deployments the database user whose name is specified in the database backend configuration may not have write privileges to the database. This is often required by the policy within a given network to secure the data from being unintentionally modified. In many cases administrators have deployed inventory databases, which contain substantially more information about the hosts than just the static reservations assigned to them. The inventory database can be used to create a view of a Kea hosts database and such a view is often read-only.

Kea host database backends operate with an implicit configuration to both read from and write to the database. If the database user does not have write access to the host database, the backend will fail to start and the server will refuse to start (or reconfigure). However, if access to a read-only host database is required for retrieving reservations for clients and/or assigning specific addresses and options, it is possible to explicitly configure Kea to start in "read-only" mode. This is controlled by the `readonly` boolean parameter as follows:

```
"Dhcp6": { "hosts-database": { "readonly": true, ... }, ... }
```

Setting this parameter to `false` configures the database backend to operate in "read-write" mode, which is also the default configuration if the parameter is not specified.

Note: The `readonly` parameter is currently only supported for MySQL and PostgreSQL databases.

9.2.4 Interface Configuration

The DHCPv6 server must be configured to listen on specific network interfaces. The simplest network interface configuration tells the server to listen on all available interfaces:

```
"Dhcp6": {
    "interfaces-config": {
        "interfaces": [ "*" ]
    }
    ...
}
```

The asterisk plays the role of a wildcard and means "listen on all interfaces." However, it is usually a good idea to explicitly specify interface names:

```
"Dhcp6": {
    "interfaces-config": {
        "interfaces": [ "eth1", "eth3" ]
    },
    ...
}
```

It is possible to use a wildcard interface name (asterisk) concurrently with explicit interface names:

```
"Dhcp6": {
    "interfaces-config": {
        "interfaces": [ "eth1", "eth3", "*" ]
    },
    ...
}
```

It is anticipated that this form of usage will only be used when it is desired to temporarily override a list of interface names and listen on all interfaces.

As with the DHCPv4 server, binding to specific addresses and disabling re-detection of interfaces are supported. But `dhcp-socket-type` is not supported, because DHCPv6 uses UDP/IPv6 sockets only. The following example shows how to disable the interface detection:

```
"Dhcp6": {
    "interfaces-config": {
        "interfaces": [ "eth1", "eth3" ],
        "re-detect": false
    },
    ...
}
```

The loopback interfaces (i.e. the "lo" or "lo0" interface) are not configured by default, unless explicitly mentioned in the configuration. Note that Kea requires a link-local address (which does not exist on all systems) or a specified unicast address, as in:

```
"Dhcp6": {
    "interfaces-config": {
        "interfaces": [ "enp0s2/2001:db8::1234:abcd" ]
    },
    ...
}
```

9.2.5 IPv6 Subnet Identifier

The subnet identifier is a unique number associated with a particular subnet. In principle, it is used to associate clients' leases with their respective subnets. When a subnet identifier is not specified for a subnet being configured, it will be automatically assigned by the configuration mechanism. The identifiers are assigned from 1 and are monotonically increased for each subsequent subnet: 1, 2, 3 ....

If there are multiple subnets configured with auto-generated identifiers and one of them is removed, the subnet identifiers may be renumbered. For example: if there are four subnets and the third is removed, the last subnet will be assigned the identifier that the third subnet had before removal. As a result, the leases stored in the lease database for subnet 3 are now associated with subnet 4, something that may have unexpected consequences. The only remedy for this issue at present is to manually specify a unique identifier for each subnet.

Note: Subnet IDs must be greater than zero and less than 4294967295.

The following configuration will assign the specified subnet identifier to a newly configured subnet:

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",
            "id": 1024,
            ...
        }
    ]
}
```

This identifier will not change for this subnet unless the "id" parameter is removed or set to 0. The value of 0 forces auto-generation of the subnet identifier.

9.2.6 IPv6 Subnet Prefix

The subnet prefix is the second way to identify a subnet. It does not need to have the address part to match the prefix length, for instance this configuration is accepted:

```
"Dhcp6": {
   "subnet6": [
       {
            "subnet": "2001:db8:1::1/64",
            ...
       }
    ]
}
```

Even there is another subnet with the "2001:db8:1::/64" prefix: only the textual form of subnets are compared to avoid duplicates.

Note: Abuse of this feature can lead to incorrect subnet selection (see *IPv6 Subnet Selection*).

9.2.7 Unicast Traffic Support

When the DHCPv6 server starts, by default it listens to the DHCP traffic sent to multicast address ff02::1:2 on each interface that it is configured to listen on (see *Interface Configuration*). In some cases it is useful to configure a server to handle incoming traffic sent to global unicast addresses as well; the most common reason for this is to have relays send their traffic to the server directly. To configure the server to listen on a specific unicast address, add a slash after the interface name, followed by the global unicast address on which the server should listen. The server will listen to this address in addition to normal link-local binding and listening on the ff02::1:2 address. The sample configuration below shows how to listen on 2001:db8::1 (a global address) configured on the eth1 interface.

```
"Dhcp6": {
    "interfaces-config": {
        "interfaces": [ "eth1/2001:db8::1" ]
    },
    ...
    "option-data": [
        {
            "name": "unicast",
            "data": "2001:db8::1"
        } ],
    ...
}
```

This configuration will cause the server to listen on eth1 on the link-local address, the multicast group (ff02::1:2), and 2001:db8::1.

Usually unicast support is associated with a server unicast option which allows clients to send unicast messages to the server. The example above includes a server unicast option specification which will cause the client to send messages to the specified unicast address.

It is possible to mix interface names, wildcards, and interface names/addresses in the list of interfaces. It is not possible, however, to specify more than one unicast address on a given interface.

Care should be taken to specify proper unicast addresses. The server will attempt to bind to the addresses specified without any additional checks. This approach was selected on purpose, to allow the software to communicate over uncommon addresses if so desired.

9.2.8 Configuration of IPv6 Address Pools

The main role of a DHCPv6 server is address assignment. For this, the server must be configured with at least one subnet and one pool of dynamic addresses to be managed. For example, assume that the server is connected to a network segment that uses the 2001:db8:1::/64 prefix. The administrator of that network decides that addresses from range 2001:db8:1::1 to 2001:db8:1::ffff are going to be managed by the Dhcp6 server. Such a configuration can be achieved in the following way:

```
"Dhcp6": {
    "subnet6": [
       {
           "subnet": "2001:db8:1::/64",
           "pools": [
               {
                   "pool": "2001:db8:1::1-2001:db8:1::ffff"
               }
           ],
           ...
       }
    ]
}
```

Note that `subnet` is defined as a simple string, but the `pools` parameter is actually a list of pools; for this reason, the pool definition is enclosed in square brackets, even though only one range of addresses is specified.

Each `pool` is a structure that contains the parameters that describe a single pool. Currently there is only one parameter, `pool`, which gives the range of addresses in the pool.

It is possible to define more than one pool in a subnet; continuing the previous example, further assume that 2001:db8:1:0:5::/80 should also be managed by the server. It could be written as 2001:db8:1:0:5:: to 2001:db8:1::5:ffff:ffff:ffff, but typing so many ‘f’s is cumbersome. It can be expressed more simply as 2001:db8:1:0:5::/80. Both formats are supported by Dhcp6 and can be mixed in the pool list. For example, one could define the following pools:

```
"Dhcp6": {
    "subnet6": [
    {
        "subnet": "2001:db8:1::/64",
        "pools": [
            { "pool": "2001:db8:1::1-2001:db8:1::ffff" },
            { "pool": "2001:db8:1:05::/80" }
        ],
        ...
    }
    ]
}
```

White space in pool definitions is ignored, so spaces before and after the hyphen are optional. They can be used to improve readability.

The number of pools is not limited, but for performance reasons it is recommended to use as few as possible.

The server may be configured to serve more than one subnet. To add a second subnet, use a command similar to the following:

```
"Dhcp6": {
    "subnet6": [
    {
        "subnet": "2001:db8:1::/64",
        "pools": [
            { "pool": "2001:db8:1::1-2001:db8:1::ffff" }
        ]
    },
    {
        "subnet": "2001:db8:2::/64",
        "pools": [
            { "pool": "2001:db8:2::/64" }
        ]
    },

        ...
    ]
}
```

In this example, we allow the server to dynamically assign all addresses available in the whole subnet. Although rather wasteful, it is certainly a valid configuration to dedicate the whole /64 subnet for that purpose. Note that the Kea server does not preallocate the leases, so there is no danger in using gigantic address pools.

When configuring a DHCPv6 server using prefix/length notation, please pay attention to the boundary values. When specifying that the server can use a given pool, it will also be able to allocate the first (typically a network address) address from that pool. For example, for pool 2001:db8:2::/64, the 2001:db8:2:: address may be assigned as well. To avoid this, use the “min-max” notation.

9.2.9 Subnet and Prefix Delegation Pools

Subnets may also be configured to delegate prefixes, as defined in RFC 8415, section 6.3. A subnet may have one or more prefix delegation pools. Each pool has a prefixed address, which is specified as a prefix (`prefix`) and a prefix length (`prefix-len`), as well as a delegated prefix length (`delegated-len`). The delegated length must not be shorter than (that is, it must be numerically greater than or equal to) the prefix length. If both the delegated and prefix lengths are equal, the server will be able to delegate only one prefix. The delegated prefix does not have to match the subnet prefix.

Below is a sample subnet configuration which enables prefix delegation for the subnet:

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "2001:d8b:1::/64",
            "pd-pools": [
                {
                    "prefix": "3000:1::",
                    "prefix-len": 64,
                    "delegated-len": 96
                }
            ]
        }
    ],
    ...
}
```

9.2.10 Prefix Exclude Option

For each delegated prefix, the delegating router may choose to exclude a single prefix out of the delegated prefix as specified in RFC 6603. The requesting router must not assign the excluded prefix to any of its downstream interfaces, and it is intended to be used on a link through which the delegating router exchanges DHCPv6 messages with the requesting router. The configuration example below demonstrates how to specify an excluded prefix within a prefix pool definition. The excluded prefix “2001:db8:1:8000:cafe:80::/72” will be sent to a requesting router which includes the Prefix Exclude option in the Option Request option (ORO), and which is delegated a prefix from this pool.

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "2001:db8:1::/48",
            "pd-pools": [
                {
                    "prefix": "2001:db8:1:8000::",
                    "prefix-len": 48,
                    "delegated-len": 64,
                    "excluded-prefix": "2001:db8:1:8000:cafe:80::",
                    "excluded-prefix-len": 72
                }
            ]
        }
    ]
}
```

9.2.11 Standard DHCPv6 Options

One of the major features of the DHCPv6 server is the ability to provide configuration options to clients. Although there are several options that require special behavior, most options are sent by the server only if the client explicitly requests them. The following example shows how to configure the addresses of DNS servers, one of the most frequently used options. Options specified in this way are considered global and apply to all configured subnets.

```
"Dhcp6": {
    "option-data": [
        {
           "name": "dns-servers",
            "code": 23,
            "space": "dhcp6",
            "csv-format": true,
            "data": "2001:db8::cafe, 2001:db8::babe"
        },
        ...
    ]
}
```

The `option-data` line creates a new entry in the option-data table. This table contains information on all global options that the server is supposed to configure in all subnets. The `name` line specifies the option name. (For a complete list of currently supported names, see *List of Standard DHCPv6 Options*.) The next line specifies the option code, which must match one of the values from that list. The line beginning with `space` specifies the option space, which must always be set to “dhcp6” as these are standard DHCPv6 options. For other name spaces, including custom option spaces, see *Nested DHCPv6 Options (Custom Option Spaces)*. The following line specifies the format in which the data will be entered; use of CSV (comma-separated values) is recommended. Finally, the `data` line gives the actual value to be sent to clients. The data parameter is specified as normal text, with values separated by commas if more than one value is allowed.

Options can also be configured as hexadecimal values. If “csv-format” is set to false, the option data must be specified as a hexadecimal string. The following commands configure the DNS-SERVERS option for all subnets with the following addresses: 2001:db8:1::cafe and 2001:db8:1::babe.

```
"Dhcp6": {
    "option-data": [
        {
            "name": "dns-servers",
            "code": 23,
            "space": "dhcp6",
            "csv-format": false,
            "data": "20 01 0D B8 00 01 00 00 00 00 00 00 00 00 CA FE
                     20 01 0D B8 00 01 00 00 00 00 00 00 00 00 BA BE"
        },
        ...
    ]
}
```

Note: The value for the setting of the “data” element is split across two lines in this example for clarity; when entering the command, the whole string should be entered on the same line.

Kea supports the following formats when specifying hexadecimal data:

- `Delimited octets` - one or more octets separated by either colons or spaces (‘:’ or ‘ ‘). While each octet may contain one or two digits, we strongly recommend always using two digits. Valid examples are “ab:cd:ef” and “ab cd ef”.
- `String of digits` - a continuous string of hexadecimal digits with or without a “0x” prefix. Valid examples are “0xabcdef” and “abcdef”.

Care should be taken to use proper encoding when using hexadecimal format; Kea’s ability to validate data correctness in hexadecimal is limited.

As of Kea 1.6.0, it is also possible to specify data for binary options as a single-quoted text string within double quotes as shown (note that `csv-format` must be set to false):

```
"Dhcp6": {
    "option-data": [
        {
            "name": "subscriber-id",
            "code": 38,
            "space": "dhcp6",
            "csv-format": false,
            "data": "'convert this text to binary'"
        },
        ...
    ],
    ...
}
```

Most of the parameters in the "option-data" structure are optional and can be omitted in some circumstances, as discussed in *Unspecified Parameters for DHCPv6 Option Configuration*. Only one of name or code is required; it is not necessary to specify both. Space has a default value of "dhcp6", so this can be skipped as well if a regular (not encapsulated) DHCPv6 option is defined. Finally, csv-format defaults to "true", so it too can be skipped, unless the option value is specified as hexstring. Therefore, the above example can be simplified to:

```
"Dhcp6": {
    "option-data": [
        {
           "name": "dns-servers",
           "data": "2001:db8::cafe, 2001:db8::babe"
        },
        ...
    ]
}
```

Defined options are added to the response when the client requests them, as well as any options required by a protocol. An administrator can also specify that an option is always sent, even if a client did not specifically request it. To enforce the addition of a particular option, set the "always-send" flag to true as in:

```
"Dhcp6": {
    "option-data": [
        {
           "name": "dns-servers",
           "data": "2001:db8::cafe, 2001:db8::babe",
           "always-send": true
        },
        ...
    ]
}
```

The effect is the same as if the client added the option code in the Option Request option (or its equivalent for vendor options), as in:

```
"Dhcp6": {
    "option-data": [
        {
           "name": "dns-servers",
           "data": "2001:db8::cafe, 2001:db8::babe",
           "always-send": true
        },
        ...
    ],
    "subnet6": [
        {
           "subnet": "2001:db8:1::/64",
           "option-data": [
               {
                   "name": "dns-servers",
                   "data": "2001:db8:1::cafe, 2001:db8:1::babe"
               },
               ...
           ],
           ...
        },
        ...
    ],
    ...
}
```

The DNS servers option is always added to responses (the always-send is "sticky"), but the value is the subnet one when the client is localized in the subnet.

It is possible to override options on a per-subnet basis. If clients connected to most subnets are expected to get the same values of a given option, administrators should use global options; it is possible to override specific values for a small number of subnets. On the other hand, if different values are used in each subnet, it does not make sense to specify global option values; rather, only subnet-specific ones should be set.

The following commands override the global DNS servers option for a particular subnet, setting a single DNS server with address 2001:db8:1::3.

```
"Dhcp6": {
    "subnet6": [
        {
            "option-data": [
                {
                    "name": "dns-servers",
                    "code": 23,
                    "space": "dhcp6",
                    "csv-format": true,
                    "data": "2001:db8:1::3"
                },
                ...
            ],
            ...
        },
        ...
    ],
    ...
}
```

In some cases it is useful to associate some options with an address or prefix pool from which a client is assigned a lease. Pool-specific option values override subnet-specific and global option values. If the client is assigned multiple leases from different pools, the server will assign options from all pools from which the leases have been obtained. However, if the particular option is specified in multiple pools from which the client obtains the leases, only one instance of this option will be handed out to the client. The server's administrator must not try to prioritize assignment of pool-specific options by trying to order pools declarations in the server configuration.

The following configuration snippet demonstrates how to specify the DNS servers option, which will be assigned to a client only if the client obtains an address from the given pool:

```
"Dhcp6": {
    "subnet6": [
        {
            "pools": [
                {
                    "pool": "2001:db8:1::100-2001:db8:1::300",
                    "option-data": [
                        {
                            "name": "dns-servers",
                            "data": "2001:db8:1::10"
                        }
                    ]
                }
            ]
        },
        ...
    ],
    ...
}
```

Options can also be specified in class or host reservation scope. The current Kea options precedence order is (from most important): host reservation, pool, subnet, shared network, class, global.

The currently supported standard DHCPv6 options are listed in *List of Standard DHCPv6 Options*. "Name" and "Code" are the values that should be used as a name/code in the option-data structures. "Type" designates the format of the data; the meanings of the various types are given in *List of Standard DHCP Option Types*.

When a data field is a string and that string contains the comma (,; U+002C) character, the comma must be escaped with two backslashes (; U+005C). This double escape is required because both the routine splitting CSV data into fields and JSON use the same escape character; a single escape (,) would make the JSON invalid. For example, the string "EST5EDT4,M3.2.0/02:00,M11.1.0/02:00" must be represented as:

```
"Dhcp6": {
    "subnet6": [
        {
            "pools": [
                {
                    "option-data": [
                        {
                            "name": "new-posix-timezone",
                            "data": "EST5EDT4\\,M3.2.0/02:00\\,M11.1.0/02:00"
                        }
                    ]
                },
                ...
            ],
            ...
        },
        ...
    ],
    ...
}
```

Some options are designated as arrays, which means that more than one value is allowed in such an option. For example, the option dns-servers allows the specification of more than one IPv6 address, enabling clients to obtain the addresses of multiple DNS servers.

Custom DHCPv6 Options describes the configuration syntax to create custom option definitions (formats). Creation of custom definitions for standard options is generally not permitted, even if the definition being created matches the actual option format defined in the RFCs. There is an exception to this rule for standard options for which Kea currently does not provide a definition. In order to use such options, a server administrator must create a definition as described in *Custom DHCPv6 Options* in the 'dhcp6' option space. This definition should match the option format described in the relevant RFC, but the configuration mechanism will allow any option format as it currently has no means to validate it.

Table 9.1: List of Standard DHCPv6 Options

Name	Code	Type	Array?
preference	7	uint8	false
unicast	12	ipv6-address	false
vendor-opts	17	uint32	false
sip-server-dns	21	fqdn	true
sip-server-addr	22	ipv6-address	true
dns-servers	23	ipv6-address	true
domain-search	24	fqdn	true
nis-servers	27	ipv6-address	true
nisp-servers	28	ipv6-address	true
nis-domain-name	29	fqdn	true
nisp-domain-name	30	fqdn	true
sntp-servers	31	ipv6-address	true
information-refresh-time	32	uint32	false
bcmcs-server-dns	33	fqdn	true
bcmcs-server-addr	34	ipv6-address	true
geoconf-civic	36	record (uint8, uint16, binary)	false
remote-id	37	record (uint32, binary)	false
subscriber-id	38	binary	false
client-fqdn	39	record (uint8, fqdn)	false
pana-agent	40	ipv6-address	true
new-posix-timezone	41	string	false
new-tzdb-timezone	42	string	false
ero	43	uint16	true
lq-query (1)	44	record (uint8, ipv6-address)	false
client-data (1)	45	empty	false
clt-time (1)	46	uint32	false
lq-relay-data (1)	47	record (ipv6-address, binary)	false
lq-client-link (1)	48	ipv6-address	true
v6-lost	51	fqdn	false
capwap-ac-v6	52	ipv6-address	true
relay-id	53	binary	false
v6-access-domain	57	fqdn	false
sip-ua-cs-list	58	fqdn	true
bootfile-url	59	string	false
bootfile-param	60	tuple	true
client-arch-type	61	uint16	true
nii	62	record (uint8, uint8, uint8)	false
aftr-name	64	fqdn	false
erp-local-domain-name	65	fqdn	false
rsoo	66	empty	false
pd-exclude	67	binary	false
rdnss-selection	74	record (ipv6-address, uint8, fqdn)	true

Continued on next page

Table 9.1 – continued from previous page

Name	Code	Type	Array?
client-linklayer-addr	79	binary	false
link-address	80	ipv6-address	false
solmax-rt	82	uint32	false
inf-max-rt	83	uint32	false
dhcp4o6-server-addr	88	ipv6-address	true
s46-rule	89	record (uint8, uint8, uint8, ipv4-address, ipv6-prefix)	false
s46-br	90	ipv6-address	false
s46-dmr	91	ipv6-prefix	false
s46-v4v6bind	92	record (ipv4-address, ipv6-prefix)	false
s46-portparams	93	record(uint8, psid)	false
s46-cont-mape	94	empty	false
s46-cont-mapt	95	empty	false
s46-cont-lw	96	empty	false
v6-captive-portal	103	string	false
ipv6-address-andsf	143	ipv6-address	true

Options marked with (1) have option definitions, but the logic behind them is not implemented. That means that, technically, Kea knows how to parse them in incoming messages or how to send them if configured to do so, but not what to do with them. Since the related RFCs require certain processing, the support for those options is non-functional. However, it may be useful in some limited lab testing; hence the definition formats are listed here.

9.2.12 Common Softwire46 Options

Softwire46 options are involved in IPv4 over IPv6 provisioning by means of tunneling or translation as specified in RFC 7598. The following sections provide configuration examples of these options.

Softwire46 Container Options

Softwire46 (S46) container options group rules and optional port parameters for a specified domain. There are three container options specified in the "dhcp6" (top-level) option space: the MAP-E Container option, the MAP-T Container option, and the S46 Lightweight 4over6 Container option. These options only contain the encapsulated options specified below; they do not include any data fields.

To configure the server to send a specific container option along with all encapsulated options, the container option must be included in the server configuration as shown below:

```
"Dhcp6": {
    ...
    "option-data": [
        {
            "name": "s46-cont-mape"
        } ],
    ...
}
```

This configuration will cause the server to include the MAP-E Container option to the client. Use "s46-cont-mapt" or "s46-cont-lw" for the MAP-T Container and S46 Lightweight 4over6 Container options, respectively.

All remaining Softwire options described below are included in one of the container options. Thus, they must be included in appropriate option spaces by selecting a "space" name, which specifies in which option they are supposed to be included.

S46 Rule Option

The S46 Rule option is used for conveying the Basic Mapping Rule (BMR) and Forwarding Mapping Rule (FMR).

```
{
    "space": "s46-cont-mape-options",
    "name": "s46-rule",
    "data": "128, 0, 24, 192.0.2.0, 2001:db8:1::/64"
}
```

Another possible "space" value is "s46-cont-mapt-options".

The S46 Rule option conveys a number of parameters:

- `flags` - an unsigned 8-bit integer, with currently only the most-significant bit specified. It denotes whether the rule can be used for forwarding (128) or not (0).
- `ea-len` - an 8-bit-long Embedded Address length. Allowed values range from 0 to 48.
- `IPv4 prefix length` - 8 bits long; expresses the prefix length of the Rule IPv4 prefix specified in the ipv4-prefix field. Allowed values range from 0 to 32.
- `IPv4 prefix` - a fixed-length 32-bit field that specifies the IPv4 prefix for the S46 rule. The bits in the prefix after a specific number of bits (defined in prefix4-len) are reserved, and MUST be initialized to zero by the sender and ignored by the receiver.
- `IPv6 prefix` - in prefix/length notation that specifies the IPv6 domain prefix for the S46 rule. The field is padded on the right with zero bits up to the nearest octet boundary, when prefix6-len is not evenly divisible by 8.

S46 BR Option

The S46 BR option is used to convey the IPv6 address of the Border Relay. This option is mandatory in the MAP-E Container option and is not permitted in the MAP-T and S46 Lightweight 4over6 Container options.

```
{
    "space": "s46-cont-mape-options",
    "name": "s46-br",
    "data": "2001:db8:cafe::1",
}
```

Another possible "space" value is "s46-cont-lw-options".

S46 DMR Option

The S46 DMR option is used to convey values for the Default Mapping Rule (DMR). This option is mandatory in the MAP-T container option and is not permitted in the MAP-E and S46 Lightweight 4over6 Container options.

```
{
    "space": "s46-cont-mapt-options",
    "name": "s46-dmr",
    "data": "2001:db8:cafe::/64",
}
```

This option must not be included in other containers.

S46 IPv4/IPv6 Address Binding Option

The S46 IPv4/IPv6 Address Binding option may be used to specify the full or shared IPv4 address of the Customer Edge (CE). The IPv6 prefix field is used by the CE to identify the correct prefix to use for the tunnel source.

```
{
    "space": "s46-cont-lw",
    "name": "s46-v4v6bind",
    "data": "192.0.2.3, 2001:db8:1:cafe::/64"
}
```

This option must not be included in other containers.

S46 Port Parameters

The S46 Port Parameters option specifies optional port-set information that MAY be provided to CEs.

```
{
    "space": "s46-rule-options",
    "name": "s46-portparams",
    "data": "2, 3/4",
}
```

Another possible "space" value is "s46-v4v6bind", to include this option in the S46 IPv4/IPv6 Address Binding option.

Note that the second value in the example above specifies the PSID and PSID-length fields in the format of PSID/PSID length. This is equivalent to the values of PSID-len=4 and PSID=12288 conveyed in the S46 Port Parameters option.

9.2.13 Custom DHCPv6 Options

Kea supports custom (non-standard) DHCPv6 options. Assume that we want to define a new DHCPv6 option called "foo" which will have code 100 and which will convey a single, unsigned, 32-bit integer value. We can define such an option by putting the following entry in the configuration file:

```
"Dhcp6": {
    "option-def": [
        {
            "name": "foo",
            "code": 100,
            "type": "uint32",
            "array": false,
            "record-types": "",
            "space": "dhcp6",
            "encapsulate": ""
        }, ...
    ],
    ...
}
```

The `false` value of the `array` parameter determines that the option does NOT comprise an array of "uint32" values but is, instead, a single value. Two other parameters have been left blank: `record-types` and `encapsulate`. The former specifies the comma-separated list of option data fields, if the option comprises a record of data fields. The `record-types` value should be non-empty if `type` is set to "record"; otherwise it must be left blank. The latter parameter specifies the name of the option space being encapsulated by the particular option. If the particular option does not encapsulate any option space, the parameter should be left blank. Note that the `option-def` configuration statement only defines the format of the new option and does not set its value(s).

The `name`, `code`, and `type` parameters are required; all others are optional. The `array` default value is `false`. The `record-types` and `encapsulate` default values are blank (i.e. “”). The default `space` is “dhcp6”.

Once the new option format is defined, its value is set in the same way as for a standard option. For example, the following commands set a global value that applies to all subnets.

```
"Dhcp6": {
    "option-data": [
        {
            "name": "foo",
            "code": 100,
            "space": "dhcp6",
            "csv-format": true,
            "data": "12345"
        }, ...
    ],
    ...
}
```

New options can take more complex forms than simple use of primitives (uint8, string, ipv6-address, etc.); it is possible to define an option comprising a number of existing primitives.

For example, assume we want to define a new option that will consist of an IPv6 address, followed by an unsigned 16-bit integer, followed by a boolean value, followed by a text string. Such an option could be defined in the following way:

```
"Dhcp6": {
    "option-def": [
        {
            "name": "bar",
            "code": 101,
            "space": "dhcp6",
            "type": "record",
            "array": false,
            "record-types": "ipv6-address, uint16, boolean, string",
            "encapsulate": ""
        }, ...
    ],
    ...
}
```

The `type` is set to “record” to indicate that the option contains multiple values of different types. These types are given as a comma-separated list in the `record-types` field and should be ones from those listed in *List of Standard DHCP Option Types*.

The values of the options are set in an `option-data` statement as follows:

```
"Dhcp6": {
    "option-data": [
        {
            "name": "bar",
            "space": "dhcp6",
            "code": 101,
            "csv-format": true,
            "data": "2001:db8:1::10, 123, false, Hello World"
        }
    ],
    ...
}
```

`csv-format` is set to `true` to indicate that the `data` field comprises a comma-separated list of values. The values in `data` must correspond to the types set in the `record-types` field of the option definition.

When `array` is set to `true` and `type` is set to "record", the last field is an array, i.e. it can contain more than one value, as in:

```
"Dhcp6": {
    "option-def": [
        {
            "name": "bar",
            "code": 101,
            "space": "dhcp6",
            "type": "record",
            "array": true,
            "record-types": "ipv6-address, uint16",
            "encapsulate": ""
        }, ...
    ],
    ...
}
```

The new option content is one IPv6 address followed by one or more 16-bit unsigned integers.

Note: In general, boolean values are specified as `true` or `false`, without quotes. Some specific boolean parameters may accept also `"true"`, `"false"`, `0`, `1`, `"0"`, and `"1"`.

9.2.14 DHCPv6 Vendor-Specific Options

Vendor options in DHCPv6 are carried in the Vendor-Specific Information option (code 17). The following examples show how to define an option "foo" with code 1 that consists of an IPv6 address, an unsigned 16-bit integer, and a string. The "foo" option is conveyed in a Vendor-Specific Information option, which comprises a single uint32 value that is set to "12345". The sub-option "foo" follows the data field holding this value.

The first step is to define the format of the option:

```
"Dhcp6": {
    "option-def": [
        {
            "name": "foo",
            "code": 1,
            "space": "vendor-12345",
            "type": "record",
            "array": false,
            "record-types": "ipv6-address, uint16, string",
            "encapsulate": ""
        }
    ],
    ...
}
```

(Note that the option space is set to `vendor-12345`.) Once the option format is defined, the next step is to define actual values for that option:

```
"Dhcp6": {
    "option-data": [
        {
            "name": "foo",
            "space": "vendor-12345",
            "data": "2001:db8:1::10, 123, Hello World"
        },
        ...
    ],
    ...
}
```

We should also define a value (enterprise-number) for the Vendor-Specific Information option, that conveys our option "foo".

```
"Dhcp6": {
    "option-data": [
        ...,
        {
            "name": "vendor-opts",
            "data": "12345"
        }
    ],
    ...
}
```

Alternatively, the option can be specified using its code.

```
"Dhcp6": {
    "option-data": [
        ...,
        {
            "code": 17,
            "data": "12345"
        }
    ],
    ...
}
```

A common configuration is to set the always-send flag to true so the vendor option is sent even when the client did not specify it in the query.

9.2.15 Nested DHCPv6 Options (Custom Option Spaces)

It is sometimes useful to define completely new option spaces, such as when a user creates a new option to convey sub-options that use a separate numbering scheme, for example sub-options with codes 1 and 2. Those option codes conflict with standard DHCPv6 options, so a separate option space must be defined.

Note that the creation of a new option space is not required when defining sub-options for a standard option, because one is created by default if the standard option is meant to convey any sub-options (see *DHCPv6 Vendor-Specific Options*).

Assume that we want to have a DHCPv6 option called "container" with code 102 that conveys two sub-options with codes 1 and 2. First we need to define the new sub-options:

```
"Dhcp6": {
    "option-def": [
        {
            "name": "subopt1",
            "code": 1,
            "space": "isc",
            "type": "ipv6-address",
            "record-types": "",
            "array": false,
            "encapsulate": ""
        },
        {
            "name": "subopt2",
            "code": 2,
            "space": "isc",
            "type": "string",
            "record-types": "",
            "array": false
            "encapsulate": ""
        }
    ],
    ...
}
```

Note that we have defined the options to belong to a new option space (in this case, “isc”).

The next step is to define a regular DHCPv6 option with the desired code and specify that it should include options from the new option space:

```
"Dhcp6": {
    "option-def": [
        ...,
        {
            "name": "container",
            "code": 102,
            "space": "dhcp6",
            "type": "empty",
            "array": false,
            "record-types": "",
            "encapsulate": "isc"
        }
    ],
    ...
}
```

The name of the option space in which the sub-options are defined is set in the `encapsulate` field. The `type` field is set to `empty`, which limits this option to only carrying data in sub-options.

Finally, we can set values for the new options:

```
"Dhcp6": {
    "option-data": [
        {
            "name": "subopt1",
            "code": 1,
            "space": "isc",
            "data": "2001:db8::abcd"
        },
        }
            "name": "subopt2",
            "code": 2,
            "space": "isc",
            "data": "Hello world"
        },
        {
            "name": "container",
            "code": 102,
            "space": "dhcp6"
        }
    ],
    ...
}
```

Note that it is possible to create an option which carries some data in addition to the sub-options defined in the encapsulated option space. For example, if the “container” option from the previous example were required to carry a uint16 value as well as the sub-options, the `type` value would have to be set to “uint16” in the option definition. (Such an option would then have the following data structure: DHCP header, uint16 value, sub-options.) The value specified with the `data` parameter — which should be a valid integer enclosed in quotes, e.g. “123” — would then be assigned to the uint16 field in the “container” option.

9.2.16 Unspecified Parameters for DHCPv6 Option Configuration

In many cases it is not required to specify all parameters for an option configuration, and the default values can be used. However, it is important to understand the implications of not specifying some of them, as it may result in configuration errors. The list below explains the behavior of the server when a particular parameter is not explicitly specified:

- `name` - the server requires an option name or an option code to identify an option. If this parameter is unspecified, the option code must be specified.
- `code` - the server requires either an option name or an option code to identify an option. This parameter may be left unspecified if the `name` parameter is specified. However, this also requires that the particular option have a definition (either as a standard option or an administrator-created definition for the option using an ‘option-def’ structure), as the option definition associates an option with a particular name. It is possible to configure an option for which there is no definition (unspecified option format). Configuration of such options requires the use of the option code.
- `space` - if the option space is unspecified it will default to ‘dhcp6’, which is an option space holding standard DHCPv6 options.
- `data` - if the option data is unspecified it defaults to an empty value. The empty value is mostly used for the options which have no payload (boolean options), but it is legal to specify empty values for some options which carry variable-length data and for which the specification allows a length of 0. For such options, the data parameter may be omitted in the configuration.
- `csv-format` - if this value is not specified, the server will assume that the option data is specified as a list of comma-separated values to be assigned to individual fields of the DHCP option.

9.2.17 Controlling the Values Sent for T1 and T2 Times

According to RFC 8415, section 21.4, the recommended T1 and T2 values are 50% and 80% of the preferred lease time, respectively. Kea can be configured to send values that are specified explicitly or that are calculated as percentages of the preferred lease time. The server’s behavior is governed by a combination of configuration parameters, two of which have already been mentioned.

Beginning with Kea 1.6.0 lease preferred and valid lifetime are extended from single values to triplets with minimum, default and maximum values using:

- `min-preferred-lifetime` - specifies the minimum preferred lifetime (optional).

- `preferred-lifetime` - specifies the default preferred lifetime.
- `max-preferred-lifetime` - specifies the maximum preferred lifetime (optional).
- `min-valid-lifetime` - specifies the minimum valid lifetime (optional).
- `valid-lifetime` - specifies the default valid lifetime.
- `max-valid-lifetime` - specifies the maximum valid lifetime (optional).

When the client does not specify lifetimes the default is used. When it specifies a lifetime using IAADDR or IAPREFIX sub option with not zero values these values are used when they are between configured minimum (lower values are round up) and maximal (larger values are round down) bounds.

To send specific, fixed values use the following two parameters:

- `renew-timer` - specifies the value of T1 in seconds.
- `rebind-timer` - specifies the value of T2 in seconds.

Any value greater than or equal to zero may be specified for T2. When specifying T1 it must be less than T2. This flexibility is allowed to support a use case where administrators want to suppress client renewals and rebinds by deferring them beyond the lifespan of the lease. This should cause the lease to expire, rather than get renewed by clients. If T1 is specified as larger than T2, T1 will be set to zero in the outbound IA.

In the great majority of cases the values should follow this rule: T1 < T2 < preferred lifetime < valid lifetime. Alternatively, both T1 and T2 values can be configured to 0, which is a signal to DHCPv6 clients that they may renew at their own discretion. However, there are known broken client implementations in use that will start renewing immediately. Administrators who plan to use T1=T2=0 values should test first and make sure their clients behave rationally.

In some rare cases there may be a need to disable a client's ability to renew addresses. This is undesired from a protocol perspective and should be avoided if possible. However, if necessary, administrators can configure the T1 and T2 values to be equal or greater to the valid lifetime. Be advised that this will cause clients to occasionally lose their addresses, which is generally perceived as poor service. However, there may be some rare business cases when this is desired (e.g. when it is desirable to intentionally break long-lasting connections).

Calculation of the values is controlled by the following three parameters:

- `calculate-tee-times` - when true, T1 and T2 will be calculated as percentages of the valid lease time. It defaults to true.
- `t1-percent` - the percentage of the valid lease time to use for T1. It is expressed as a real number between 0.0 and 1.0 and must be less than t2-percent. The default value is 0.5 per RFC 8415.
- `t2-percent` - the percentage of the valid lease time to use for T2. It is expressed as a real number between 0.0 and 1.0 and must be greater than t1-percent. The default value is 0.8 per RFC 8415.

Note: In the event that both explicit values are specified and calculate-tee-times is true, the server will use the explicit values. Administrators with a setup where some subnets or share-networks will use explicit values and some will use calculated values must not define the explicit values at any level higher than where they will be used. Inheriting them from too high a scope, such as global, will cause them to have values at every level underneath (shared-networks and subnets), effectively disabling calculated values.

9.2.18 IPv6 Subnet Selection

The DHCPv6 server may receive requests from local (connected to the same subnet as the server) and remote (connected via relays) clients. As the server may have many subnet configurations defined, it must select an appropriate subnet for a given request.

In IPv4, the server can determine which of the configured subnets are local, as there is a reasonable expectation that the server will have a (global) IPv4 address configured on the interface. That assumption is not true in IPv6; the DHCPv6 server must be able to operate while only using link-local addresses. Therefore, an optional `interface` parameter is available within a subnet definition to designate that a given subnet is local, i.e. reachable directly over the specified interface. For example, a server that is intended to serve a local subnet over eth0 may be configured as follows:

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "2001:db8:beef::/48",
            "pools": [
                 {
                     "pool": "2001:db8:beef::/48"
                 }
             ],
            "interface": "eth0"
        }
    ],
    ...
}
```

9.2.19 Rapid Commit

The Rapid Commit option, described in RFC 8415, is supported by the Kea DHCPv6 server. However, support is disabled by default. It can be enabled on a per-subnet basis using the `rapid-commit` parameter as shown below:

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "2001:db8:beef::/48",
            "rapid-commit": true,
            "pools": [
                 {
                     "pool": "2001:db8:beef::1-2001:db8:beef::10"
                 }
             ],
        }
    ],
    ...
}
```

This setting only affects the subnet for which `rapid-commit` is set to `true`. For clients connected to other subnets, the server will ignore the Rapid Commit option sent by the client and will follow the 4-way exchange procedure, i.e. respond with an Advertise for a Solicit containing a Rapid Commit option.

9.2.20 DHCPv6 Relays

A DHCPv6 server with multiple subnets defined must select the appropriate subnet when it receives a request from a client. For clients connected via relays, two mechanisms are used:

The first uses the linkaddr field in the RELAY_FORW message. The name of this field is somewhat misleading in that it does not contain a link-layer address; instead, it holds an address (typically a global address) that is used to identify a link. The DHCPv6 server checks to see whether the address belongs to a defined subnet and, if it does, that subnet is selected for the client's request.

The second mechanism is based on interface-id options. While forwarding a client's message, relays may insert an interface-id option into the message that identifies the interface on the relay that received the message. (Some relays allow configuration of that parameter, but it is sometimes hardcoded and may range from the very simple (e.g. "vlan100") to the very cryptic; one example seen on real hardware was "ISAM144|299|ipv6|nt:vp:1:110"). The server can use this information to select the appropriate subnet. The information is also returned to the relay, which then knows the interface to use to transmit the response to the client. For this to work successfully, the relay interface IDs must be unique within the network and the server configuration must match those values.

When configuring the DHCPv6 server, it should be noted that two similarly named parameters can be configured for a subnet:

- `interface` defines which local network interface can be used to access a given subnet.
- `interface-id` specifies the content of the interface-id option used by relays to identify the interface on the relay to which the response packet is sent.

The two are mutually exclusive; a subnet cannot be reachable both locally (direct traffic) and via relays (remote traffic). Specifying both is a configuration error and the DHCPv6 server will refuse such a configuration.

The following example configuration shows how to specify an interface-id with a value of "vlan123":

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "2001:db8:beef::/48",
            "pools": [
                 {
                     "pool": "2001:db8:beef::/48"
                 }
             ],
            "interface-id": "vlan123"
        }
    ],
    ...
}
```

9.2.21 Relay-Supplied Options

RFC 6422 defines a mechanism called Relay-Supplied DHCP Options. In certain cases relay agents are the only entities that may have specific information, and they can insert options when relaying messages from the client to the server. The server will then do certain checks and copy those options to the response sent to the client.

There are certain conditions that must be met for the option to be included. First, the server must not provide the option itself; in other words, if both relay and server provide an option, the server always takes precedence. Second, the option must be RSOO-enabled. (RSOO is the "Relay Supplied Options option.") IANA maintains a list of RSOO-enabled options here. However, there may be cases when system administrators want to echo other options. Kea can be instructed to treat other options as RSOO-enabled. For example, to mark options 110, 120, and 130 as RSOO-enabled, the following syntax should be used:

```
"Dhcp6": {
    "relay-supplied-options": [ "110", "120", "130" ],
    ...
}
```

As of February 2019, only option 65 is RSOO-enabled by IANA. This option will always be treated as such, so there is no need to explicitly mark it. Also, when enabling standard options, it is possible to use their names rather than their option code, e.g. use `dns-servers` instead of `23`. See ref:*dhcp6-std-options-list* for the names. In certain cases this may also work for custom options, but due to the nature of the parser code this may be unreliable and should be avoided.

9.2.22 Client Classification in DHCPv6

The DHCPv6 server includes support for client classification. For a deeper discussion of the classification process see *Client Classification*.

In certain cases it is useful to configure the server to differentiate between DHCP client types and treat them accordingly. Client classification can be used to modify the behavior of almost any part of the DHCP message processing. Kea currently offers three mechanisms that take advantage of client classification in DHCPv6: subnet selection, address pool selection, and DHCP options assignment.

Kea can be instructed to limit access to given subnets based on class information. This is particularly useful for cases where two types of devices share the same link and are expected to be served from two different subnets. The primary use case for such a scenario is cable networks, where there are two classes of devices: the cable modem itself, which should be handed a lease from subnet A; and all other devices behind the modem, which should get a lease from subnet B. That segregation is essential to prevent overly curious users from playing with their cable modems. For details on how to set up class restrictions on subnets, see *Configuring Subnets With Class Information*.

When subnets belong to a shared network, the classification applies to subnet selection but not to pools; that is, a pool in a subnet limited to a particular class can still be used by clients which do not belong to the class, if the pool they are expected to use is exhausted. So the limit on access based on class information is also available at the address/prefix pool level; see *Configuring Pools With Class Information*, within a subnet. This is useful when segregating clients belonging to the same subnet into different address ranges.

In a similar way, a pool can be constrained to serve only known clients, i.e. clients which have a reservation, using the built-in “KNOWN” or “UNKNOWN” classes. Addresses can be assigned to registered clients without giving a different address per reservation, for instance when there are not enough available addresses. The determination whether there is a reservation for a given client is made after a subnet is selected, so it is not possible to use “KNOWN”/”UNKNOWN” classes to select a shared network or a subnet.

The process of classification is conducted in five steps. The first step is to assess an incoming packet and assign it to zero or more classes. The second step is to choose a subnet, possibly based on the class information. When the incoming packet is in the special class, “DROP, it is dropped and an debug message logged. The next step is to evaluate class expressions depending on the built-in “KNOWN”/”UNKNOWN” classes after host reservation lookup, using them for pool/pd-pool selection and assigning classes from host reservations. The list of required classes is then built and each class of the list has its expression evaluated; when it returns “true” the packet is added as a member of the class. The last step is to assign options, again possibly based on the class information. More complete and detailed information is available in *Client Classification*.

There are two main methods of classification. The first is automatic and relies on examining the values in the vendor class options or the existence of a host reservation. Information from these options is extracted, and a class name is constructed from it and added to the class list for the packet. The second specifies an expression that is evaluated for each packet. If the result is “true”, the packet is a member of the class.

Note: Care should be taken with client classification, as it is easy for clients that do not meet class criteria to be denied all service.

Defining and Using Custom Classes

The following example shows how to configure a class using an expression and a subnet using that class. This configuration defines the class named “Client_enterprise”. It is comprised of all clients whose client identifiers start with the given hex string (which would indicate a DUID based on an enterprise id of 0xAABBCCDD). Members of this class will be given an address from 2001:db8:1::0 to 2001:db8:1::FFFF and the addresses of their DNS servers set to 2001:db8:0::1 and 2001:db8:2::1.

```
"Dhcp6": {
    "client-classes": [
        {
            "name": "Client_enterprise",
            "test": "substring(option[1].hex,0,6) == 0x0002AABBCCDD",
            "option-data": [
                {
                    "name": "dns-servers",
                    "code": 23,
                    "space": "dhcp6",
                    "csv-format": true,
                    "data": "2001:db8:0::1, 2001:db8:2::1"
                }
            ]
        },
        ...
    ],
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",
            "pools": [ { "pool": "2001:db8:1::-2001:db8:1::ffff" } ],
            "client-class": "Client_enterprise"
        }
    ],
    ...
}
```

This example shows a configuration using an automatically generated "VENDOR_CLASS_" class. The administrator of the network has decided that addresses in the range 2001:db8:1::1 to 2001:db8:1::ffff are to be managed by the DHCP6 server and that only clients belonging to the eRouter1.0 client class are allowed to use that pool.

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",
            "pools": [
                 {
                     "pool": "2001:db8:1::-2001:db8:1::ffff"
                 }
             ],
            "client-class": "VENDOR_CLASS_eRouter1.0"
        }
    ],
    ...
}
```

Required Classification

In some cases it is useful to limit the scope of a class to a shared network, subnet, or pool. There are two parameters which are used to limit the scope of the class by instructing the server to evaluate test expressions when required.

The first one is the per-class `only-if-required` flag, which is false by default. When it is set to `true`, the test expression of the class is not evaluated at the reception of the incoming packet but later, and only if the class evaluation is required.

The second is `require-client-classes`, which takes a list of class names and is valid in shared-network, subnet, and pool scope. Classes in these lists are marked as required and evaluated after selection of this specific shared-network/subnet/pool and before output option processing.

In this example, a class is assigned to the incoming packet when the specified subnet is used:

```
"Dhcp6": {
    "client-classes": [
       {
           "name": "Client_foo",
           "test": "member('ALL')",
           "only-if-required": true
       },
       ...
    ],
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64"
            "pools": [
                 {
                     "pool": "2001:db8:1::-2001:db8:1::ffff"
                 }
             ],
            "require-client-classes": [ "Client_foo" ],
            ...
        },
        ...
    ],
    ...
}
```

Required evaluation can be used to express complex dependencies like subnet membership. It can also be used to reverse the precedence; if an option-data is set in a subnet it takes precedence over an option-data in a class. When option-data is moved to a required class and required in the subnet, a class evaluated earlier may take precedence.

Required evaluation is also available at shared-network and pool/pd-pool levels. The order in which required classes are considered is: shared-network, subnet, and (pd-)pool, i.e. in the opposite order in which option-data is processed.

9.2.23 DDNS for DHCPv6

As mentioned earlier, kea-dhcp6 can be configured to generate requests to the DHCP-DDNS server (referred to here as "D2") to update DNS entries. These requests are known as Name Change Requests or NCRs. Each NCR contains the following information:

1. Whether it is a request to add (update) or remove DNS entries
2. Whether the change requests forward DNS updates (AAAA records), reverse DNS updates (PTR records), or both
3. The Fully Qualified Domain Name (FQDN), lease address, and DHCID (information identifying the client associated with the FQDN)

Prior to Kea 1.7.1, all parameters for controlling DDNS were within the global `dhcp-ddns` section of the kea-dhcp6. Beginning with Kea 1.7.1 DDNS related parameters were split into two groups:

1. Connectivity Parameters

    > These are parameters which specify where and how kea-dhcp6 connects to and communicates with

D2. These parameters can only be specified within the top-level `dhcp-ddns` section in the kea-dhcp6 configuration. The connectivity parameters are listed below:

- `enable-updates`
- `server-ip`
- `server-port`
- `sender-ip`
- `sender-port`
- `max-queue-size`
- `ncr-protocol`
- `ncr-format"`

2. Behavioral Parameters

   These parameters influence behavior such as how client host names and FQDN options are handled. They have been moved out of the `dhcp-ddns` section so that they may be specified at the global, shared-network, and/or subnet levels. Furthermore, they are inherited downward from global to shared-network to subnet. In other words, if a parameter is not specified at a given level, the value for that level comes from the level above it. The behavioral parameter as follows:

   - `ddns-send-updates`
   - `ddns-override-no-update`
   - `ddns-override-client-update`
   - `ddns-replace-client-name"`
   - `ddns-generated-prefix`
   - `ddns-qualifying-suffix`
   - `hostname-char-set`
   - `hostname-char-replacement`

Note: For backward compatibility, configuration parsing will still recognize the original behavioral parameters specified in `dhcp-ddns`. It will do so by translating the parameter into its global equivalent. If a parameter is specified both globally and in `dhcp-ddns`, the latter value will be ignored. In either case, a log will be emitted explaining what has occurred. Specifying these values within `dhcp-ddns` is deprecated and support for it will be removed at some future date.

The default configuration and values would appear as follows:

```
"Dhcp6": {
     "dhcp-ddns": {
        // Connectivity parameters
        "enable-updates": false,
         "server-ip": "127.0.0.1",
         "server-port":53001,
         "sender-ip":"",
         "sender-port":0,
         "max-queue-size":1024,
         "ncr-protocol":"UDP",
         "ncr-format":"JSON"
     },
    // Behavioral parameters (global)
    "ddns-send-updates": true,
    "ddns-override-no-update": false,
    "ddns-override-client-update": false,
    "ddns-replace-client-name": "never",
    "ddns-generated-prefix": "myhost",
    "ddns-qualifying-suffix": "",
    "hostname-char-set": "",
    "hostname-char-replacement": ""
    ...
}
```

As of Kea 1.7.1, there are two parameters which determine if kea-dhcp6 can generate DDNS requests to D2. The existing, `dhcp-ddns:enable-updates` parameter which now only controls whether kea-dhcp6 connects to D2. And the new behavioral parameter, `ddns-send-updates`, which determines if DDNS updates are enabled at a given level (i.e global, shared-network, or subnet). The following table shows how the two parameters function together:

Table 9.2: Enabling and Disabling DDNS Updates

dhcp-ddns: enable-updates	Global ddns-send-udpates	Outcome
false (default)	false	no updates at any scope
false	true (default)	no updates at any scope
true	false	updates only at scopes with a local value of true for ddns-enable-updates
true	true	updates at all scopes except those with a local value of false for ddns-enable-updates

DHCP-DDNS Server Connectivity

For NCRs to reach the D2 server, kea-dhcp6 must be able to communicate with it. kea-dhcp6 uses the following configuration parameters to control this communication:

- `enable-updates` - As of Kea 1.7.1, this parameter only enables connectivity to kea-dhcp-ddns such that DDNS updates can be constructed and sent. It must be true for NCRs to be generated and sent to D2. It defaults to false.
- `server-ip` - IP address on which D2 listens for requests. The default is the local loopback interface at address 127.0.0.1. Either an IPv4 or IPv6 address may be specified.
- `server-port` - port on which D2 listens for requests. The default value is 53001.
- `sender-ip` - the IP address which kea-dhcp6 uses to send requests to D2. The default value is blank, which instructs kea-dhcp6 to select a suitable address.
- `sender-port` - the port which kea-dhcp6 uses to send requests to D2. The default value of 0 instructs kea-dhcp6 to select a suitable port.
- `max-queue-size` - the maximum number of requests allowed to queue waiting to be sent to D2. This value guards against requests accumulating uncontrollably if they are being generated faster than they can be delivered. If the number of requests queued for transmission reaches this value, DDNS updating will be turned off until the queue backlog has been sufficiently reduced. The intent is to allow the kea-dhcp6 server to continue lease operations without running the risk that its memory usage grows without limit. The default value is 1024.
- `ncr-protocol` - the socket protocol to use when sending requests to D2. Currently only UDP is supported.

- `ncr-format` - the packet format to use when sending requests to D2. Currently only JSON format is supported.

By default, kea-dhcp-ddns is assumed to be running on the same machine as kea-dhcp6, and all of the default values mentioned above should be sufficient. If, however, D2 has been configured to listen on a different address or port, these values must be altered accordingly. For example, if D2 has been configured to listen on 2001:db8::5 port 900, the following configuration is required:

```
"Dhcp6": {
    "dhcp-ddns": {
        "server-ip": "2001:db8::5",
        "server-port": 900,
        ...
    },
    ...
}
```

When Does the kea-dhcp6 Server Generate a DDNS Request?

kea-dhcp6 follows the behavior prescribed for DHCP servers in RFC 4704. It is important to keep in mind that kea-dhcp6 makes the initial decision of when and what to update and forwards that information to D2 in the form of NCRs. Carrying out the actual DNS updates and dealing with such things as conflict resolution are within the purview of D2 itself (see *The DHCP-DDNS Server*). This section describes when kea-dhcp6 will generate NCRs and the configuration parameters that can be used to influence this decision. It assumes that the `enable-updates` parameter is true.

Note: Currently the interface between kea-dhcp6 and D2 only supports requests which update DNS entries for a single IP address. If a lease grants more than one address, kea-dhcp6 will create the DDNS update request for only the first of these addresses.

In general, kea-dhcp6 will generate DDNS update requests when:

1. A new lease is granted in response to a DHCPREQUEST;
2. An existing lease is renewed but the FQDN associated with it has changed; or
3. An existing lease is released in response to a DHCPRELEASE.

In the second case, lease renewal, two DDNS requests will be issued: one request to remove entries for the previous FQDN, and a second request to add entries for the new FQDN. In the last case, a lease release, a single DDNS request to remove its entries will be made.

As for the first case, the decisions involved when granting a new lease are more complex. When a new lease is granted, kea-dhcp6 will generate a DDNS update request only if the DHCPREQUEST contains the FQDN option (code 39). By default, kea-dhcp6 will respect the FQDN N and S flags specified by the client as shown in the following table:

Table 9.3: Default FQDN Flag Behavior

Client Flags:N-S	Client Intent	Server Response	Server Flags:N-S-O
0-0	Client wants to do forward updates, server should do reverse updates	Server generates reverse-only request	1-0-0
0-1	Server should do both forward and reverse updates	Server generates request to update both directions	0-1-0
1-0	Client wants no updates done	Server does not generate a request	1-0-0

The first row in the table above represents "client delegation." Here the DHCP client states that it intends to do the forward DNS updates and the server should do the reverse updates. By default, kea-dhcp6 will honor the client's wishes and generate a DDNS request to D2 to update only reverse DNS data. The parameter `ddns-override-client-update` can be used to instruct the server to override client delegation requests. When this parameter is "true", kea-dhcp6 will disregard requests for client delegation and generate a DDNS request to update both forward and reverse DNS data. In this case, the N-S-O flags in the server's response to the client will be 0-1-1 respectively.

(Note that the flag combination N=1, S=1 is prohibited according to RFC 4702. If such a combination is received from the client, the packet will be dropped by kea-dhcp6.)

To override client delegation, set the following values in the configuration file:

```
"Dhcp6": {
    ...
    "ddns-override-client-update": true,
    ...
}
```

The third row in the table above describes the case in which the client requests that no DNS updates be done. The parameter, `ddns-override-no-update`, can be used to instruct the server to disregard the client's wishes. When this parameter is true, kea-dhcp6 will generate DDNS update requests to kea-dhcp-ddns even if the client requests that no updates be done. The N-S-O flags in the server's response to the client will be 0-1-1.

To override client delegation, issue the following commands:

```
"Dhcp6": {
    ...
    "ddns-override-no-update": true,
    ...
}
```

kea-dhcp6 Name Generation for DDNS Update Requests

Each Name Change Request must of course include the fully qualified domain name whose DNS entries are to be affected. kea-dhcp6 can be configured to supply a portion or all of that name, based upon what it receives from the client in the DHCPREQUEST.

The default rules for constructing the FQDN that will be used for DNS entries are:

1. If the DHCPREQUEST contains the client FQDN option, take the candidate name from there.
2. If the candidate name is a partial (i.e. unqualified) name, then add a configurable suffix to the name and use the result as the FQDN.
3. If the candidate name provided is empty, generate an FQDN using a configurable prefix and suffix.
4. If the client provides neither option, then take no DNS action.

These rules can be amended by setting the `ddns-replace-client-name` parameter, which provides the following modes of behavior:

- `never` - use the name the client sent. If the client sent no name, do not generate one. This is the default mode.
- `always` - replace the name the client sent. If the client sent no name, generate one for the client.
- `when-present` - replace the name the client sent. If the client sent no name, do not generate one.
- `when-not-present` - use the name the client sent. If the client sent no name, generate one for the client.

Note: Note that in early versions of Kea, this parameter was a boolean and permitted only values of `true` and `false`. Boolean values have been deprecated and are no longer accepted. Administrators currently using booleans must replace them with the desired mode name. A value of `true` maps to `"when-present"`, while `false` maps to `"never"`.

For example, to instruct kea-dhcp6 to always generate the FQDN for a client, set the parameter `ddns-replace-client-name` to `always` as follows:

```
"Dhcp6": {
    ...
    "ddsn-replace-client-name": "always",
    ...
}
```

The prefix used in the generation of an FQDN is specified by the `ddns-generated-prefix` parameter. The default value is "myhost". To alter its value, simply set it to the desired string:

```
"Dhcp6": {
    ...
    "ddns-generated-prefix": "another.host",
    ...
}
```

The suffix used when generating an FQDN, or when qualifying a partial name, is specified by the `ddns-qualifying-suffix` parameter. This parameter has no default value; thus, it is mandatory when DDNS updates are enabled. To set its value simply set it to the desired string:

```
"Dhcp6": {
    ...
    "ddns-qualifying-suffix": "foo.example.org",
    ...
}
```

When qualifying a partial name, kea-dhcp6 will construct the name in the format:

[**candidate-name**].[**ddns-qualifying-suffix**].

where **candidate-name** is the partial name supplied in the DHCPREQUEST. For example, if the FQDN domain name value is "some-computer" and the `ddsn-qualifying-suffix` "example.com", the generated FQDN is:

some-computer.example.com.

When generating the entire name, kea-dhcp6 will construct the name in the format:

[**ddns-generated-prefix**]-[**address-text**].[**ddns-qualifying-suffix**].

where **address-text** is simply the lease IP address converted to a hyphenated string. For example, if the lease address is 3001:1::70E, the qualifying suffix "example.com", and the default value is used for `ddns-generated-prefix`, the generated FQDN is:

myhost-3001-1–70E.example.com.

Sanitizing Client FQDN Names

Some DHCP clients may provide values in the name component of the FQDN option (option code 39) that contain undesirable characters. It is possible to configure kea-dhcp6 to sanitize these values. The most typical use case is ensuring that only characters that are permitted by RFC 1035 be included: A-Z, a-z, 0-9, and '-'. This may be accomplished with the following two parameters:

- `hostname-char-set` - a regular expression describing the invalid character set. This can be any valid, regular expression using POSIX extended expression syntax. Embedded nuls (0x00) will always be considered an invalid character to be replaced (or omitted).
- `hostname-char-replacement` - a string of zero or more characters with which to replace each invalid character in the host name. An empty string and will cause invalid characters to be OMITTED rather than replaced.

Note: Starting with Kea 1.7.5, the default values are as follows:

- "hostname-char-set": "[^A-Za-z0-9.-]",
- "hostname-char-replacement": ""

This enables sanitizing and will omit any character that is not a letter,digit, hyphen, dot or nul.

The following configuration will replace anything other than a letter, digit, hyphen, or dot with the letter 'x':

```
"Dhcp6": {
    ...
    "hostname-char-set": "[^A-Za-z0-9.-]",
    "hostname-char-replacement": "x",
    ...
}
```

Thus, a client-supplied value of "myhost-$[123.org" would become "myhost-xx123.org". Sanitizing is performed only on the portion of the name supplied by the client, and it is performed before applying a qualifying suffix (if one is defined and needed).

Note: The following are some considerations to keep in mind: Name sanitizing is meant to catch the more common cases of invalid characters through a relatively simple character-replacement scheme. It is difficult to devise a scheme that works well in all cases. Administrators who find they have clients with odd corner cases of character combinations that cannot be readily handled with this mechanism should consider writing a hook that can carry out sufficiently complex logic to address their needs.

Do not include dots in the hostname-char-set expression. When scrubbing FQDNs, dots are treated as delimiters and used to separate the option value into individual domain labels that are scrubbed and then re-assembled.

If clients are sending values that differ only by characters considered as invalid by the hostname-char-set, be aware that scrubbing them will yield identical values. In such cases, DDNS conflict rules will permit only one of them to register the name.

Finally, given the latitude clients have in the values they send, it is virtually impossible to guarantee that a combination of these two parameters will always yield a name that is valid for use in DNS. For example, using an empty value for hostname-char-replacement could yield an empty domain label within a name, if that label consists only of invalid characters.

Note: Since the 1.6.0 Kea release it is possible to specify hostname-char-set and/or hostname-char-replacement at the global scope. This allows to sanitize host names without requiring a dhcp-ddns entry. When a hostname-char parameter is defined at the global scope and in a dhcp-ddns entry the second (local) value is used.

9.2.24 DHCPv4-over-DHCPv6: DHCPv6 Side

The support of DHCPv4-over-DHCPv6 transport is described in RFC 7341 and is implemented using cooperating DHCPv4 and DHCPv6 servers. This section is about the configuration of the DHCPv6 side (the DHCPv4 side is described in *DHCPv4-over-DHCPv6: DHCPv4 Side*).

Note: DHCPv4-over-DHCPv6 support is experimental and the details of the inter-process communication may change; both the DHCPv4 and DHCPv6 sides should be running the same version of Kea. For instance, the support of port relay (RFC 8357) introduced an incompatible change.

There is only one specific parameter for the DHCPv6 side: `dhcp4o6-port`, which specifies the first of the two consecutive ports of the UDP sockets used for the communication between the DHCPv6 and DHCPv4 servers. The DHCPv6 server is bound to ::1 on `port` and connected to ::1 on `port` + 1.

Two other configuration entries are generally required: unicast traffic support (see *Unicast Traffic Support*) and DHCP 4o6 server address option (name "dhcp4o6-server-addr", code 88).

The following configuration was used during some tests:

```
{

# DHCPv6 conf
"Dhcp6": {

    "interfaces-config": {
        "interfaces": [ "eno33554984/2001:db8:1:1::1" ]
    },

    "lease-database": {
        "type": "memfile",
        "name": "leases6"
    },

    "preferred-lifetime": 3000,
    "valid-lifetime": 4000,
    "renew-timer": 1000,
    "rebind-timer": 2000,

    "subnet6": [ {
        "subnet": "2001:db8:1:1::/64",
        "interface": "eno33554984",
        "pools": [ { "pool": "2001:db8:1:1::1:0/112" } ]
    } ],

    "dhcp4o6-port": 6767,

    "option-data": [ {
        "name": "dhcp4o6-server-addr",
        "code": 88,
        "space": "dhcp6",
        "csv-format": true,
        "data": "2001:db8:1:1::1"
    } ],


    "loggers": [ {
        "name": "kea-dhcp6",
        "output_options": [ {
            "output": "/tmp/kea-dhcp6.log"
        } ],
        "severity": "DEBUG",
        "debuglevel": 0
    } ]
}

}
```

Note: Relayed DHCPv4-QUERY DHCPv6 messages are not supported.

9.2.25 Sanity Checks in DHCPv6

An important aspect of a well-running DHCP system is an assurance that the data remain consistent. However, in some cases it may be convenient to tolerate certain inconsistent data. For example, a network administrator that temporarily removed a subnet from a configuration would not want all the leases associated with it to disappear from the lease database. Kea has a mechanism to control sanity checks for situations such as this.

Kea supports a configuration scope called `sanity-checks`. It currently allows only a single parameter, called `lease-checks`, which governs the verification carried out when a new lease is loaded from a lease file. This mechanism permits Kea to attempt to correct inconsistent data.

Every subnet has a subnet-id value; this is how Kea internally identifies subnets. Each lease has a subnet-id parameter as well, which identifies which subnet it belongs to. However, if the configuration has changed, it is possible that a lease could exist with a subnet-id, but without any subnet that matches it. Also, it may be possible that the subnet's configuration has changed and the subnet-id now belongs to a subnet that does not match the lease. Kea's corrective algorithm first checks to see if there is a subnet with the subnet-id specified by the lease. If there is, it verifies whether the lease belongs to that subnet. If not, depending on the lease-checks setting, the lease is discarded, a warning is displayed, or a new subnet is selected for the lease that matches it topologically.

Since delegated prefixes do not have to belong to a subnet in which they are offered, there is no way to implement such a mechanism for IPv6 prefixes. As such, the mechanism works for IPv6 addresses only.

There are five levels which are supported:

- `none` - do no special checks; accept the lease as is.
- `warn` - if problems are detected display a warning, but accept the lease data anyway. This is the default value.
- `fix` - if a data inconsistency is discovered, try to correct it. If the correction is not successful, the incorrect data will be inserted anyway.
- `fix-del` - if a data inconsistency is discovered, try to correct it. If the correction is not successful, reject the lease. This setting ensures the data's correctness, but some incorrect data may be lost. Use with care.
- `del` - this is the strictest mode. If any inconsistency is detected, reject the lease. Use with care.

This feature is currently implemented for the memfile backend.

An example configuration that sets this parameter looks as follows:

```
"Dhcp6": {
    "sanity-checks": {
        "lease-checks": "fix-del"
    },
    ...
}
```

9.2.26 Storing Extended Lease Information

In order to support such features as DHCPv6 Reconfigure (RFC 3315) and LeaseQuery (RFC 5007) it is necessary to store additional information with each lease. Because the amount of information stored for each lease has ramifications in terms of performance and system resource consumption, storing this additional information is configurable through the "store-extended-info" parameter. It defaults to false and may be set at the global, shared-network, and subnet levels.

```
"Dhcp6": {
    "store-extended-info": true,
    ...
}
```

When enabled, information relevant to the DHCPv6 query (e.g. REQUEST, RENEW, or REBIND) asking for the lease is added into the lease's user-context as a map element labeled "ISC". Currently the information contained in the map will be a list of relays, one for each relay message layer that encloses the client query. Other values may be added at a future date. The lease's user-context for a two-hop query might look something like this (shown pretty-printed for clarity):

```
{
    "ISC": {
        "relays": [
        {
            "hop": 2,
            "link": "2001:db8::1",
            "peer": "2001:db8::2"
        },
        {
            "hop": 1,
            "link": "2001:db8::3",
            "options": "0x00C800080102030405060708",
            "peer": "2001:db8::4"
        }]
    }
}
```

Note: This feature is intended to be used in conjunction with an upcoming LeaseQuery hook library and at this time there is other use for this information within Kea.

Note: It is possible that other hook libraries are already making use of user-context. Enabling store-extended-info should not interfere with any other user-context content so long as it does not also use an element labled "ISC". In other words, user-context is intended to be a flexible container serving mulitple purposes. As long as no other purpose also writes an "ISC" element to user-context there should not be a conflict.

9.2.27 Multi-threading settings

The Kea server can be configured to process packets in parallel using multiple threads. These settings can be found under `multi-threading` structure and are represented by:

- `enable-multi-threading` - use multiple threads to process packets in parallel (default false).
- `thread-pool-size` - specify the number of threads to process packets in parallel. Supported values are: 0 (auto detect), any positive number sets thread count explicitly (default 0).
- `packet-queue-size` - specify the size of the queue used by the thread pool to process packets. Supported values are: 0 (unlimited), any positive number sets queue size explicitly (default 64).

An example configuration that sets these parameter looks as follows:

```
"Dhcp6": {
    "multi-threading": {
       "enable-multi-threading": true,
       "thread-pool-size": 4,
       "packet-queue-size": 16
    }
    ...
}
```

9.2.28 Multi-threading settings in different backends

Both kea-dhcp4 and kea-dhcp6 are tested internally to determine which settings gave the best performance. Although this section describes our results, those are just recommendations and are very dependent on the particular hardware that was used for testing. We strongly advice to run your own performance tests.

Full report of Kea 1.7 performance results can be found here. This includes hardware description, test scenario descriptions and current results.

After enabling multi-threading, the number of threads is set by `thread-pool-size` parameter, and results from our tests show that best configurations for kea-dhcp6 are:

- `thread-pool-size`: 4 when using `memfile` for storing leases.
- `thread-pool-size`: 12 or more when using `mysql` for storing leases.
- `thread-pool-size`: 6 when using `postgresql`.

Another very important parameter is `packet-queue-size` and in our tests we used it as multiplier of `thread-pool-size`. So actual setting strongly depends on `thread-pool-size`.

Our tests reported best results when:

- `packet-queue-size`: 150 * `thread-pool-size` when using `memfile` for storing leases. In our case it's 150 * 4 = 600. This means that at any given time, up to 600 packets could be queued.
- `packet-queue-size`: 200 * `thread-pool-size` when using `mysql` for storing leases. In our case it's 200 * 12 = 2400. This means that up to 2400 packets could be queued.
- `packet-queue-size`: 11 * `thread-pool-size` when using `postgresql` for storing leases. In our case it's 11 * 6 = 66.

9.3 Host Reservation in DHCPv6

There are many cases where it is useful to provide a configuration on a per-host basis. The most obvious one is to reserve a specific, static IPv6 address or/and prefix for exclusive use by a given client (host); the returning client will receive the same address or/and prefix every time, and other clients will never get that address. Another situation when host reservations are applicable is when a host has specific requirements, e.g. a printer that needs additional DHCP options or a cable modem that needs specific parameters. Yet another possible use case is to define unique names for hosts.

Note that there may be cases when a new reservation has been made for a client for an address or prefix currently in use by another client. We call this situation a "conflict." These conflicts get resolved automatically over time as described in subsequent sections. Once the conflict is resolved, the correct client will receive the reserved configuration when it renews.

Host reservations are defined as parameters for each subnet. Each host must be identified by either DUID or its hardware/MAC address; see *MAC/Hardware Addresses in DHCPv6* for details. There is an optional `reservations` array in the `subnet6` structure; each element in that array is a structure that holds information about a single host. In particular, the structure has an identifier that uniquely identifies a host. In the DHCPv6 context, the identifier is usually a DUID, but it can also be a hardware or MAC address. One or more addresses or prefixes may also be specified, and it is possible to specify a hostname and DHCPv6 options for a given host.

Note: Kea requires that reserved addresses must be within the subnet. Kea 1.7.10 is the last release that does not enforce this. This does not apply to reserved prefixes.

The following example shows how to reserve addresses and prefixes for specific hosts:

```
"subnet6": [
    {
        "subnet": "2001:db8:1::/48",
        "pools": [ { "pool": "2001:db8:1::/80" } ],
        "pd-pools": [
            {
                "prefix": "2001:db8:1:8000::",
                "prefix-len": 48,
                "delegated-len": 64
            }
        ],
        "reservations": [
            {
                "duid": "01:02:03:04:05:0A:0B:0C:0D:0E",
                "ip-addresses": [ "2001:db8:1::100" ]
            },
            {
                "hw-address": "00:01:02:03:04:05",
                "ip-addresses": [ "2001:db8:1::101", "2001:db8:1::102" ]
            },
            {
                "duid": "01:02:03:04:05:06:07:08:09:0A",
                "ip-addresses": [ "2001:db8:1::103" ],
                "prefixes": [ "2001:db8:2:abcd::/64" ],
                "hostname": "foo.example.com"
            }
        ]
    }
]
```

This example includes reservations for three different clients. The first reservation is for the address 2001:db8:1::100 for a client using DUID 01:02:03:04:05:0A:0B:0C:0D:0E. The second reservation is for two addresses, 2001:db8:1::101 and 2001:db8:1::102, for a client using MAC address 00:01:02:03:04:05. Lastly, address 2001:db8:1::103 and prefix 2001:db8:2:abcd::/64 are reserved for a client using DUID 01:02:03:04:05:06:07:08:09:0A. The last reservation also assigns a hostname to this client.

Note that DHCPv6 allows a single client to lease multiple addresses and multiple prefixes at the same time. Therefore `ip-addresses` and `prefixes` are plural and are actually arrays. When the client sends multiple IA options (IA_NA or IA_PD), each reserved address or prefix is assigned to an individual IA of the appropriate type. If the number of IAs of a specific type is lower than the number of reservations of that type, the number of reserved addresses or prefixes assigned to the client is equal to the number of IA_NAs or IA_PDs sent by the client; that is, some reserved addresses or prefixes are not assigned. However, they still remain reserved for this client and the server will not assign them to any other client. If the number of IAs of a specific type sent by the client is greater than the number of reserved addresses or prefixes, the server will try to assign all reserved addresses or prefixes to the individual IAs and dynamically allocate addresses or prefixes to the remaining IAs. If the server cannot assign a reserved address or prefix because it is in use, the server will select the next reserved address or prefix and try to assign it to the client. If the server subsequently finds that there are no more reservations that can be assigned to the client at that moment, the server will try to assign leases dynamically.

Making a reservation for a mobile host that may visit multiple subnets requires a separate host definition in each subnet that host is expected to visit. It is not possible to define multiple host definitions with the same hardware address in a single subnet. Multiple host definitions with the same hardware address are valid if each is in a different subnet. The reservation for a given host should include only one identifier, either DUID or hardware address; defining both for the same host is considered a configuration error.

Adding host reservations incurs a performance penalty. In principle, when a server that does not support host reservation responds to a query, it needs to check whether there is a lease for a given address being considered for allocation or renewal. The server that does support host reservation has to perform additional checks: not only whether the address is currently used (i.e., if there is a lease for it), but also whether the address could be used by someone else (i.e., if there is a reservation for it). That additional check incurs extra overhead.

9.3.1 Address/Prefix Reservation Types

In a typical scenario there is an IPv6 subnet defined, with a certain part of it dedicated for dynamic address allocation by the DHCPv6 server. There may be an additional address space defined for prefix delegation. Those dynamic parts are referred to as dynamic pools, address and prefix pools, or simply pools. In principle, a host reservation can reserve any address or prefix that belongs to the subnet. The reservations that specify addresses that belong to configured pools are called “in-pool reservations.” In contrast, those that do not belong to dynamic pools are called “out-of-pool reservations.” There is no formal difference in the reservation syntax and both reservation types are handled uniformly.

Kea supports global host reservations. These are reservations that are specified at the global level within the configuration and that do not belong to any specific subnet. Kea will still match inbound client packets to a subnet as before, but when the subnet’s reservation mode is set to `"global"`, Kea will look for host reservations only among the global reservations defined. Typically, such reservations would be used to reserve hostnames for clients which may move from one subnet to another.

Note: Global reservations, while useful in certain circumstances, have aspects that must be given due consideration. Please see *Conflicts in DHCPv6 Reservations* for more details.

9.3.2 Conflicts in DHCPv6 Reservations

As reservations and lease information are stored separately, conflicts may arise. Consider the following series of events: the server has configured the dynamic pool of addresses from the range of 2001:db8::10 to 2001:db8::20. Host

A requests an address and gets 2001:db8::10. Now the system administrator decides to reserve address 2001:db8::10 for Host B. In general, reserving an address that is currently assigned to someone else is not recommended, but there are valid use cases where such an operation is warranted.

The server now has a conflict to resolve. If Host B boots up and requests an address, the server is not able to assign the reserved address 2001:db8::10. A naive approach would to be immediately remove the lease for Host A and create a new one for Host B. That would not solve the problem, though, because as soon as Host B gets the address, it will detect that the address is already in use (by Host A) and will send a DHCPDECLINE message. Therefore, in this situation, the server has to temporarily assign a different address from the dynamic pool (not matching what has been reserved) to Host B.

When Host A renews its address, the server will discover that the address being renewed is now reserved for someone else - Host B. The server will remove the lease for 2001:db8::10, select a new address, and create a new lease for it. It will send two addresses in its response: the old address, with lifetime set to 0 to explicitly indicate that it is no longer valid; and the new address, with a non-zero lifetime. When Host B tries to renew its temporarily assigned address, the server will detect that the existing lease does not match the reservation, so it will release the current address Host B has and will create a new lease matching the reservation. As before, the server will send two addresses: the temporarily assigned one with zeroed lifetimes, and the new one that matches the reservation with proper lifetimes set.

This recovery will succeed, even if other hosts attempt to get the reserved address. If Host C requests the address 2001:db8::10 after the reservation is made, the server will propose a different address.

This recovery mechanism allows the server to fully recover from a case where reservations conflict with existing leases; however, this procedure will take roughly take as long as the value set for renew-timer. The best way to avoid such recovery is not to define new reservations that conflict with existing leases. Another recommendation is to use out-of-pool reservations. If the reserved address does not belong to a pool, there is no way that other clients can get it.

Note: The conflict-resolution mechanism does not work for global reservations. Although the global address reservations feature may be useful in certain settings, it is generally recommended not to use global reservations for addresses. Administrators who do choose to use global reservations must manually ensure that the reserved addresses are not in dynamic pools.

9.3.3 Reserving a Hostname

When the reservation for a client includes the `hostname`, the server will assign this hostname to the client and send it back in the Client FQDN, if the client sent the FQDN option to the server. The reserved hostname always takes precedence over the hostname supplied by the client (via the FQDN option) or the autogenerated (from the IPv6 address) hostname.

The server qualifies the reserved hostname with the value of the `ddns-qualifying-suffix` parameter. For example, the following subnet configuration:

```
"subnet6": [
    {
        "subnet": "2001:db8:1::/48",
        "pools": [ { "pool": "2001:db8:1::/80" } ],
        "ddns-qualifying-suffix": "example.isc.org.",
        "reservations": [
            {
                "duid": "01:02:03:04:05:0A:0B:0C:0D:0E",
                "ip-addresses": [ "2001:db8:1::100" ]
                "hostname": "alice-laptop"
            }
        ]
    }
],
"dhcp-ddns": {
    "enable-updates": true
}
```

will result in assigning the "alice-laptop.example.isc.org." hostname to the client using the DUID "01:02:03:04:05:0A:0B:0C:0D:0E". If the `ddns-qualifying-suffix` is not specified, the default (empty) value will be used, and in this case the value specified as a `hostname` will be treated as a fully qualified name. Thus, by leaving the `ddns-qualifying-suffix` empty it is possible to qualify hostnames for different clients with different domain names:

```
"subnet6": [
    {
        "subnet": "2001:db8:1::/48",
        "pools": [ { "pool": "2001:db8:1::/80" } ],
        "reservations": [
            {
                "duid": "01:02:03:04:05:0A:0B:0C:0D:0E",
                "ip-addresses": [ "2001:db8:1::100" ]
                "hostname": "mark-desktop.example.org."
            }
        ]
    }
],
"dhcp-ddns": {
    "enable-updates": true,
}
```

The above example results in the assignment of the "mark-desktop.example.org." hostname to the client using the DUID "01:02:03:04:05:0A:0B:0C:0D:0E".

9.3.4 Including Specific DHCPv6 Options in Reservations

Kea offers the ability to specify options on a per-host basis. These options follow the same rules as any other options. These can be standard options (see *Standard DHCPv6 Options*), custom options (see *Custom DHCPv6 Options*), or vendor-specific options (see *DHCPv6 Vendor-Specific Options*). The following example demonstrates how standard options can be defined.

```
"reservations": [
{
   "duid": "01:02:03:05:06:07:08",
   "ip-addresses": [ "2001:db8:1::2" ],
    "option-data": [
    {
        "option-data": [ {
            "name": "dns-servers",
            "data": "3000:1::234"
        },
        {
            "name": "nis-servers",
            "data": "3000:1::234"
        }
    } ]
} ]
```

Vendor-specific options can be reserved in a similar manner:

```
"reservations": [
{
    "duid": "aa:bb:cc:dd:ee:ff",
    "ip-addresses": [ "2001:db8::1" ],
    "option-data": [
    {
        "name": "vendor-opts",
        "data": 4491
    },
    {
        "name": "tftp-servers",
        "space": "vendor-4491",
        "data": "3000:1::234"
    } ]
} ]
```

Options defined at host level have the highest priority. In other words, if there are options defined with the same type on global, subnet, class, and host levels, the host-specific values will be used.

9.3.5 Reserving Client Classes in DHCPv6

Using Expressions in Classification explains how to configure the server to assign classes to a client, based on the content of the options that this client sends to the server. Host reservations mechanisms also allow for the static assignment of classes to clients. The definitions of these classes are placed in the Kea configuration or a database. The following configuration snippet shows how to specify that a client belongs to classes `reserved-class1` and `reserved-class2`. Those classes are associated with specific options sent to the clients which belong to them.

```
{
    "client-classes": [
    {
       "name": "reserved-class1",
       "option-data": [
       {
           "name": "dns-servers",
           "data": "2001:db8:1::50"
       }
       ]
   },
   {
       "name": "reserved-class2",
       "option-data": [
       {
           "name": "nis-servers",
           "data": "2001:db8:1::100"
       }
       ]
    }
    ],
    "subnet6": [
    {   "pools": [ { "pool": "2001:db8:1::/64" } ],
        "subnet": "2001:db8:1::/48",
        "reservations": [
        {
            "duid": "01:02:03:04:05:06:07:08",

            "client-classes": [ "reserved-class1", "reserved-class2" ]
            } ]
        } ]
}
```

In some cases the host reservations can be used in conjuction with client classes specified within the Kea configuration. In particular, when a host reservation exists for a client within a given subnet, the "KNOWN" built-in class is assigned to the client. Conversely, when there is no static assignment for the client, the "UNKNOWN" class is assigned to the client. Class expressions within the Kea configuration file can refer to "KNOWN" or "UNKNOWN" classes using using the "member" operator. For example:

```
{
    "client-classes": [
        {
            "name": "dependent-class",
            "test": "member('KNOWN')",
            "only-if-required": true
        }
    ]
}
```

Note that the `only-if-required` parameter is needed here to force evaluation of the class after the lease has been allocated and thus the reserved class has been also assigned.

Note: Be aware that the classes specified in non global host reservations are assigned to the processed packet after all classes with the `only-if-required` parameter set to `false` have been evaluated. This has an implication that these classes must not depend on the statically assigned classes from the host reservations. If there is a need to create such dependency, the `only-if-required` must be set to `true` for the dependent classes. Such classes are evaluated after the static classes have been assigned to the packet. This, however, imposes additional configuration overhead, because all classes marked as `only-if-required` must be listed in the `require-client-classes` list for every subnet where they are used.

Note: Client classes specified within the Kea configuration file may depend on the classes specified within the global host reservations. In such case the `only-if-required` parameter is not needed. Refer to the *Pool Selection with Client Class Reservations* and *Subnet Selection with Client Class Reservations* for the specific use cases.

9.3.6 Storing Host Reservations in MySQL, PostgreSQL, or Cassandra

It is possible to store host reservations in MySQL, PostgreSQL, or Cassandra. See *Hosts Storage* for information on how to configure Kea to use reservations stored in MySQL, PostgreSQL, or Cassandra. Kea provides a dedicated hook for managing reservations in a database; section *host_cmds: Host Commands* provides detailed information. The Kea wiki provides some examples of how to conduct common host reservations operations.

Note: In Kea, the maximum length of an option specified per-host is arbitrarily set to 4096 bytes.

9.3.7 Fine-Tuning DHCPv6 Host Reservation

The host reservation capability introduces additional restrictions for the allocation engine (the component of Kea that selects an address for a client) during lease selection and renewal. In particular, three major checks are necessary.

First, when selecting a new lease, it is not sufficient for a candidate lease to simply not be in use by another DHCP client; it also must not be reserved for another client. Second, when renewing a lease, an additional check must be performed to see whether the address being renewed is reserved for another client. Finally, when a host renews an address or a prefix, the server must check whether there is a reservation for this host, so the existing (dynamically allocated) address should be revoked and the reserved one be used instead.

Some of those checks may be unnecessary in certain deployments and not performing them may improve performance. The Kea server provides the `reservation-mode` configuration parameter to select the types of reservations allowed for a particular subnet. Each reservation type has different constraints for the checks to be performed by the server when allocating or renewing a lease for the client. Allowed values are:

- `all` - enables both in-pool and out-of-pool host reservation types. This setting is the default value, and is the safest and most flexible. However, as all checks are conducted, it is also the slowest. It does not check against global reservations.
- `out-of-pool` - allows only out-of-pool host reservations. With this setting in place, the server may assume that all host reservations are for addresses that do not belong to the dynamic pool. Therefore, it can skip the reservation checks when dealing with in-pool addresses, thus improving performance. Do not use this mode if any reservations use in-pool addresses. Caution is advised when using this setting; Kea does not sanity-check the reservations against `reservation-mode` and misconfiguration may cause problems.
- `global` - allows only global host reservations. With this setting in place, the server searches for reservations for a client only among the defined global reservations. If an address is specified, the server skips the reservation checks carried out when dealing in other modes, thus improving performance. Caution is advised when using this setting; Kea does not sanity-check the reservations when `global` and misconfiguration may cause problems.
- `disabled` - host reservation support is disabled. As there are no reservations, the server will skip all checks. Any reservations defined will be completely ignored. As the checks are skipped, the server may operate faster in this mode.

The parameter can be specified at global, subnet, and shared-network levels.

An example configuration that disables reservation looks as follows:

```
"Dhcp6": {
    "subnet6": [
        {
        "subnet": "2001:db8:1::/64",
        "reservation-mode": "disabled",
        ...
        }
    ]
}
```

An example configuration using global reservations is shown below:

```
"Dhcp6": {


    "reservation-mode": "global",
    "reservations": [
       {
        "duid": "00:03:00:01:11:22:33:44:55:66",
        "hostname": "host-one"
       },
       {
        "duid": "00:03:00:01:99:88:77:66:55:44",
        "hostname": "host-two"
        }
    ],

    "subnet6": [
    {
        "subnet": "2001:db8:1::/64",
        ...
    }
    ]
}
```

For more details regarding global reservations, see *Global Reservations in DHCPv6*.

Another aspect of host reservations is the different types of identifiers. Kea currently supports two types of identifiers in DHCPv6: hardware address and DUID. This is beneficial from a usability perspective; however, there is one drawback. For each incoming packet Kea has to extract each identifier type and then query the database to see if there is a reservation by this particular identifier. If nothing is found, the next identifier is extracted and the next query is issued. This process continues until either a reservation is found or all identifier types have been checked. Over time, with an increasing number of supported identifier types, Kea would become slower and slower.

To address this problem, a parameter called `host-reservation-identifiers` is available. It takes a list of identifier types as a parameter. Kea will check only those identifier types enumerated in host-reservation-identifiers. From a performance perspective, the number of identifier types should be kept to a minimum, ideally one. If the deployment uses several reservation types, please enumerate them from most- to least-frequently used, as this increases the chances of Kea finding the reservation using the fewest queries. An example of host reservation identifiers looks as follows:

```
"host-reservation-identifiers": [ "duid", "hw-address" ],
"subnet6": [
    {
        "subnet": "2001:db8:1::/64",
        ...
    }
]
```

If not specified, the default value is:

```
"host-reservation-identifiers": [ "hw-address", "duid" ]
```

9.3.8 Global Reservations in DHCPv6

In some deployments, such as mobile, clients can roam within the network and certain parameters must be specified regardless of the client's current location. To facilitate such a need, a global reservation mechanism has been implemented. The idea behind it is that regular host reservations are tied to specific subnets, by using a specific subnet-id. Kea can specify a global reservation that can be used in every subnet that has global reservations enabled.

This feature can be used to assign certain parameters, such as hostname or other dedicated, host-specific options. It can also be used to assign addresses or prefixes. However, global reservations that assign either of these bypass the whole topology determination provided by DHCP logic implemented in Kea. It is very easy to misuse this feature and get a configuration that is inconsistent. To give a specific example, imagine a global reservation for an address 2001:db8:1111::1 and two subnets 2001:db8:1111::/48 and 2001:db8:ffff::/48. If global reservations are used in both subnets and a device matching global host reservations visits part of the network that is covered by 2001:db8:ffff::/48, it will get an IP address 2001:db8:ffff::1, which will be outside of the prefix announced by its local router using Router Advertisements. Such a configuration is unusable or, at the very least, riddled with issues, such as downlink traffic not reaching the device.

To use global host reservations, a configuration similar to the following can be used:

```
"Dhcp6:" {
    # This specifies global reservations.
    # They will apply to all subnets that
    # have global reservations enabled.

    "reservations": [
    {
       "hw-address": "aa:bb:cc:dd:ee:ff",
       "hostname": "hw-host-dynamic"
    },
    {
       "hw-address": "01:02:03:04:05:06",
       "hostname": "hw-host-fixed",

       # Use of IP address in global reservation is risky.
       # If used outside of matching subnet, such as 3001::/64,
       # it will result in a broken configuration being handed
       # to the client.
       "ip-address": "2001:db8:ff::77"
    },
    {
       "duid": "01:02:03:04:05",
       "hostname": "duid-host"
    }
    ],
    "valid-lifetime": 600,
    "subnet4": [ {
        "subnet": "2001:db8:1::/64",
        "reservation-mode": "global",
        "pools": [ { "pool": "2001:db8:1::-2001:db8:1::100" } ]
    } ]
}
```

When using database backends, the global host reservations are distinguished from regular reservations by using subnet-id value of zero.

9.3.9 Pool Selection with Client Class Reservations

Client classes can be specified both in the Kea configuration file and/or host reservations. The classes specified in the Kea configuration file are evaluated immediately after receiving the DHCP packet and therefore can be used to influence subnet selection using the `client-class` parameter specified in the subnet scope. The classes specified within the host reservations are fetched and assigned to the packet after the server has already selected a subnet for the client. This means that the client class specified within a host reservation cannot be used to influence subnet assignment for this client, unless the subnet belongs to a shared network. If the subnet belongs to a shared network, the server may dynamically change the subnet assignment while trying to allocate a lease. If the subnet does not belong to a shared network, once selected, the subnet is not changed.

If the subnet does not belong to a shared network, it is possible to use host reservation based client classification to select an address pool within the subnet as follows:

```
"Dhcp6": {
    "client-classes": [
        {
            "name": "reserved_class"
        },
        {
            "name": "unreserved_class",
            "test": "not member('reserved_class')"
        }
    ],
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",
            "reservations": [{"
                "hw-address": "aa:bb:cc:dd:ee:fe",
                "client-classes": [ "reserved_class" ]
             }],
            "pools": [
                {
                    "pool": "2001:db8:1::10-2001:db8:1::20",
                    "client-class": "reserved_class"
                },
                {
                    "pool": "2001:db8:1::30-2001:db8:1::40",
                    "client-class": "unreserved_class"
                }
            ]
        }
    ]
}
```

The `reserved_class` is declared without the `test` parameter because it may be only assigned to the client via host reservation mechanism. The second class, `unreserved_class`, is assigned to the clients which do not belong to the `reserved_class`. The first pool within the subnet is only used for the clients having a reservation for the `reserved_class`. The second pool is used for the clients not having such reservation. The configuration snippet includes one host reservation which causes the client having the MAC address of aa:bb:cc:dd:ee:fe to be assigned to the `reserved_class`. Thus, this client will be given an IP address from the first address pool.

9.3.10 Subnet Selection with Client Class Reservations

There is one specific use case when subnet selection may be influenced by client classes specified within host reservations. This is the case when the client belongs to a shared network. In such case it is possible to use classification to select a subnet within this shared network. Consider the following example:

```
"Dhcp6": {
    "client-classes": [
        {
            "name": "reserved_class"
        },
        {
            "name: "unreserved_class",
            "test": "not member('reserved_class")
        }
    ],
    "reservations": [{"
        "hw-address": "aa:bb:cc:dd:ee:fe",
        "client-classes": [ "reserved_class" ]
    }],
    "reservation-mode": "global",
    "shared-networks": [{
        "subnet6": [
            {
                "subnet": "2001:db8:1::/64",
                "pools": [
                    {
                        "pool": "2001:db8:1::10-2001:db8:1::20",
                        "client-class": "reserved_class"
                    }
                ]
            },
            {
                "subnet": "2001:db8:2::/64",
                "pools": [
                    {
                        "pool": "2001:db8:2::10-2001:db8:2::20",
                        "client-class": "unreserved_class"
                    }
                ]
            }
        ]
    }]
}
```

This is similar to the example described in the *Pool Selection with Client Class Reservations*. This time, however, there are two subnets, each of them having a pool associated with a different class. The clients which don't have a reservation for the `reserved_class` will be assigned an address from the subnet 2001:db8:2::/64. Clients having a reservation for the `reserved_class` will be assigned an address from the subnet 2001:db8:1::/64. The subnets must belong to the same shared network. In addition, the reservation for the client class must be specified at the global scope (global reservation) and the `reservation-mode` must be set to `global`.

In the example above the `client-class` could also be specified at the subnet level rather than pool level yielding the same effect.

9.4 Shared Networks in DHCPv6

DHCP servers use subnet information in two ways. First, it is used to determine the point of attachment, or where the client is connected to the network. Second, the subnet information is used to group information pertaining to a specific location in the network. This approach works well in general, but there are scenarios where the boundaries are blurred. Sometimes it is useful to have more than one logical IP subnet being deployed on the same physical link. Understanding that two or more subnets are used on the same link requires additional logic in the DHCP server. This capability is called "shared networks" in the Kea and ISC DHCP projects. (It is sometimes also called "shared subnets"; in Microsoft's nomenclature it is called "multinet.")

There are many use cases where the feature is useful; the most common example in IPv4 is when the server is running out of available addresses in a subnet. This is less common in IPv6, but shared networks are still useful in IPv6. One of the use cases is an exhaustion of IPv6- delegated prefixes within a subnet; another is an experiment with an addressing scheme. With the advent of IPv6 deployment and a vast address space, many organizations split the address space into subnets, deploy it, and then after a while discover that they want to split it differently. In the transition period, they want both old and new addressing to be available; thus the need for more than one subnet on the same physical link.

Finally, the case of cable networks is directly applicable in IPv6. There are two types of devices in cable networks: cable modems and the end-user devices behind them. It is a common practice to use different subnets for cable modems to prevent users from tinkering with them. In this case, the distinction is based on the type of device, rather than on address-space exhaustion.

A client connected to a shared network may be assigned a lease (address or prefix) from any of the pools defined within the subnets belonging to the shared network. Internally, the server selects one of the subnets belonging to a shared network and tries to allocate a lease from this subnet. If the server is unable to allocate a lease from the selected subnet (e.g., due to pools exhaustion), it will use another subnet from the same shared network and will try to allocate a lease from this subnet, etc. Therefore, the server will typically allocate all leases available in a given subnet before it starts allocating leases from other subnets belonging to the same shared network. However, in certain situations the client can be allocated a lease from the other subnets before the pools in the first subnet get exhausted; this sometimes occurs when the client provides a hint that belongs to another subnet, or the client has reservations in a subnet other than the default.

Note: Deployments should not assume that Kea waits until it has allocated all the addresses from the first subnet in a shared network before allocating addresses from other subnets.

In order to define a shared network an additional configuration scope is introduced:

```
"Dhcp6": {
    "shared-networks": [{
        # Name of the shared network. It may be an arbitrary string
        # and it must be unique among all shared networks.
        "name": "ipv6-lab-1",

        # The subnet selector can be specified on the shared network
        # level. Subnets from this shared network will be selected
        # for clients communicating via relay agent having
        # the specified IP address.
        "relay": {
            "ip-addresses": [ "2001:db8:2:34::1" ]
        },

        # This starts a list of subnets in this shared network.
        # There are two subnets in this example.
        "subnet6": [{
            "subnet": "2001:db8::/48",
            "pools": [{ "pool":  "2001:db8::1 - 2001:db8::ffff" }]
        }, {
            "subnet": "3ffe:ffe::/64",
            "pools": [{ "pool":  "3ffe:ffe::/64" }]
        }]
    }], # end of shared-networks

    # It is likely that in the network there will be a mix of regular,
    # "plain" subnets and shared networks. It is perfectly valid
    # to mix them in the same configuration file.
    #
    # This is a regular subnet. It is not part of any shared-network.
    "subnet6": [{
        "subnet": "2001:db9::/48",
        "pools": [{ "pool":  "2001:db9::/64" }],
        "relay": {
            "ip-addresses": [ "2001:db8:1:2::1" ]
        }
    }]
} # end of Dhcp6
```

As demonstrated in the example, it is possible to mix shared and regular ("plain") subnets. Each shared network must have a unique name. This is similar to the ID for subnets, but gives administrators more flexibility. It is used for logging, but also internally for identifying shared networks.

In principle it makes sense to define only shared networks that consist of two or more subnets. However, for testing purposes, an empty subnet or a network with just a single subnet is allowed. This is not a recommended practice in production networks, as the shared network logic requires additional processing and thus lowers the server's performance. To avoid unnecessary performance degradation, the shared subnets should only be defined when required by the deployment.

Shared networks provide an ability to specify many parameters in the shared network scope that apply to all subnets within it. If necessary, it is possible to specify a parameter in the shared network scope and then override its value in the subnet scope. For example:

```
"shared-networks": [
    {
        "name": "lab-network3",
        "relay": {
             "ip-addresses": [ "2001:db8:2:34::1" ]
        },

        # This applies to all subnets in this shared network, unless
        # values are overridden on subnet scope.
        "valid-lifetime": 600,

        # This option is made available to all subnets in this shared
        # network.
        "option-data": [ {
            "name": "dns-servers",
            "data": "2001:db8::8888"
        } ],

        "subnet6": [
            {
                "subnet": "2001:db8:1::/48",
                "pools": [ { "pool":  "2001:db8:1::1 - 2001:db8:1::ffff" } ],

                # This particular subnet uses different values.
                "valid-lifetime": 1200,
                "option-data": [
                {
                    "name": "dns-servers",
                    "data": "2001:db8::1:2"
                },
                {
                    "name": "unicast",
                    "data": "2001:abcd::1"
                } ]
            },
            {
                 "subnet": "2001:db8:2::/48",
                 "pools": [ { "pool":  "2001:db8:2::1 - 2001:db8:2::ffff" } ],

                 # This subnet does not specify its own valid-lifetime value,
                 # so it is inherited from shared network scope.
                 "option-data": [
                 {
                     "name": "dns-servers",
                     "data": "2001:db8:cafe::1"
                 } ]
            }
        ],
    } ]
```

In this example, there is a dns-servers option defined that is available to clients in both subnets in this shared network. Also, the valid lifetime is set to 10 minutes (600s). However, the first subnet overrides some of the values (valid lifetime is 20 minutes, different IP address for dns-servers), but also adds its own option (unicast address). Assuming a client asking for a server unicast and dns-servers options is assigned a lease from this subnet, it will get a lease for 20 minutes and dns-servers, and be allowed to use server unicast at address 2001:abcd::1. If the same client is assigned to the second subnet, it will get a 10-minute lease, a dns-servers value of 2001:db8:cafe::1, and no server unicast.

Some parameters must be the same in all subnets in the same shared network. This restriction applies to the `interface` and `rapid-commit` settings. The most convenient way is to define them on the shared network scope, but they can be specified for each subnet. However, care should be taken for each subnet to have the same value.

Note: There is an inherent ambiguity when using clients that send multiple IA options in a single request and shared-networks whose subnets have different values for options and configuration parameters. The server sequentially processes IA options in the order they are occur in the client's query. If the leases requested in the IA options end up being fulfilled from different subnets then which parameters and options should apply? Currently, the code will use the values from the last subnet of the last IA option fulfilled.

We view this largely as a site configuration issue. A shared-network generally means the same physical link, so services configured by options from subnet A should be as easily reachable from subnet B and vice versa. There are number of ways to avoid this situation:

- Use the same values for options and parameters for subnets within the shared-network.
- Use subnet selectors or client class guards that ensure that for a single client's query, the same subnet will be used for all IA options in that query.
- Avoid using shared-networks with clients that send multiple IA options per query

9.4.1 Local and Relayed Traffic in Shared Networks

It is possible to specify an interface name at the shared network level to tell the server that this specific shared network is reachable directly (not via relays) using the local network interface. As all subnets in a shared network are expected to be used on the same physical link, it is a configuration error to attempt to define a shared network using subnets that are reachable over different interfaces. In other words, all subnets within the shared network must have the same value of the "interface" parameter. The following configuration is wrong.

```
"shared-networks": [
    {
        "name": "office-floor-2",
        "subnet6": [
            {
                "subnet": "2001:db8::/64",
                "pools": [ { "pool":  "2001:db8::1 - 2001:db8::ffff" } ],
                "interface": "eth0"
            },
            {
                 "subnet": "3ffe:abcd::/64",
                 "pools": [ { "pool":  "3ffe:abcd::1 - 3ffe:abcd::ffff" } ],

                 # Specifying the different interface name is a configuration
                 # error. This value should rather be "eth0" or the interface
                 # name in the other subnet should be "eth1".
                # "interface": "eth1"
            }
        ],
    } ]
```

To minimize the chance of the configuration errors, it is often more convenient to simply specify the interface name once, at the shared network level, like shown in the example below.

```
"shared-networks": [
    {
        "name": "office-floor-2",

        # This tells Kea that the whole shared network is reachable over a
        # local interface. This applies to all subnets in this network.
        "interface": "eth0",

        "subnet6": [
            {
                "subnet": "2001:db8::/64",
                "pools": [ { "pool":  "2001:db8::1 - 2001:db8::ffff" } ],
            },
            {
                 "subnet": "3ffe:abcd::/64",
                 "pools": [ { "pool":  "3ffe:abcd::1 - 3ffe:abcd::ffff" } ]
            }
        ],
    } ]
```

In case of the relayed traffic, the subnets are typically selected using the relay agents' addresses. If the subnets are used independently (not grouped within a shared network) it is allowed to specify different relay address for each of these subnets. When multiple subnets belong to a shared network they must be selected via the same relay address and, similarly to the case of the local traffic described above, it is a configuration error to specify different relay addresses for the respective subnets in the shared network. The following configuration is wrong.

```
"shared-networks": [
    {
        "name": "kakapo",
        "subnet6": [
            {
                "subnet": "2001:db8::/64",
                "relay": {
                    "ip-addresses": [ "2001:db8::1234" ]
                },
                "pools": [ { "pool":  "2001:db8::1 - 2001:db8::ffff" } ]
            },
            {
                 "subnet": "3ffe:abcd::/64",
                 "pools": [ { "pool":  "3ffe:abcd::1 - 3ffe:abcd::ffff" } ],
                 "relay": {
                    # Specifying a different relay address for this
                    # subnet is a configuration error. In this case
                    # it should be 2001:db8::1234 or the relay address
                    # in the previous subnet should be 3ffe:abcd::cafe.
                    "ip-addresses": [ "3ffe:abcd::cafe" ]
                 }
            }
        ]
    }
]
```

Again, it is better to specify the relay address at the shared network level and this value will be inherited by all subnets belonging to the shared network.

```
"shared-networks": [
    {
        "name": "kakapo",
        "relay": {
            # This relay address is inherited by both subnets.
            "ip-addresses": [ "2001:db8::1234" ]
        },
        "subnet6": [
            {
                "subnet": "2001:db8::/64",
                "pools": [ { "pool":  "2001:db8::1 - 2001:db8::ffff" } ]
            },
            {
                 "subnet": "3ffe:abcd::/64",
                 "pools": [ { "pool":  "3ffe:abcd::1 - 3ffe:abcd::ffff" } ]
            }
        ]
    }
]
```

Even though it is technically possible to configure two (or more) subnets within the shared network to use different relay addresses, this will almost always lead to a different behavior than what the user would expect. In this case, the Kea server will initially select one of the subnets by matching the relay address in the client's packet with the subnet's conifguration. However, it MAY end up using the other subnet (even though it does not match the relay address) if the client already has a lease in this subnet, has a host reservation in this subnet or simply the initially selected subnet has no more addresses available. Therefore, it is strongly recommended to always specify subnet selectors (interface or a relay address) at shared network level if the subnets belong to a shared network, as it is rarely useful to specify them at the subnet level and it may lead to the configurtion errors described above.

9.4.2 Client Classification in Shared Networks

Sometimes it is desirable to segregate clients into specific subnets based on certain properties. This mechanism is called client classification and is described in *Client Classification*. Client classification can be applied to subnets belonging to shared networks in the same way as it is used for subnets specified outside of shared networks. It is important to understand how the server selects subnets for clients when client classification is in use, to ensure that the desired subnet is selected for a given client type.

If a subnet is associated with a class, only the clients belonging to this class can use this subnet. If there are no classes specified for a subnet, any client connected to a given shared network can use this subnet. A common mistake is to assume that the subnet including a client class is preferred over subnets without client classes. Consider the following example:

```
{
    "client-classes": [
        {
            "name": "b-devices",
            "test": "option[1234].hex == 0x0002"
        }
    ],
    "shared-networks": [
        {
            "name": "galah",
            "relay": {
                "ip-address": [ "2001:db8:2:34::1" ]
            },
            "subnet6": [
                {
                    "subnet": "2001:db8:1::/64",
                    "pools": [ { "pool": "2001:db8:1::20 - 2001:db8:1::ff" } ],
                },
                {
                    "subnet": "2001:db8:3::/64",
                    "pools": [ { "pool": "2001:db8:3::20 - 2001:db8:3::ff" } ],
                    "client-class": "b-devices"
                }
            ]
        }
    ]
}
```

If the client belongs to the “b-devices” class (because it includes option 1234 with a value of 0x0002), that does not guarantee that the subnet 2001:db8:3::/64 will be used (or preferred) for this client. The server can use either of the two subnets, because the subnet 2001:db8:1::/64 is also allowed for this client. The client classification used in this case should be perceived as a way to restrict access to certain subnets, rather than a way to express subnet preference. For example, if the client does not belong to the “b-devices” class it may only use the subnet 2001:db8:1::/64 and will never use the subnet 2001:db8:3::/64.

A typical use case for client classification is in a cable network, where cable modems should use one subnet and other devices should use another subnet within the same shared network. In this case it is necessary to apply classification on all subnets. The following example defines two classes of devices, and the subnet selection is made based on option 1234 values.

```
{
    "client-classes": [
        {

            "name": "a-devices",
            "test": "option[1234].hex == 0x0001"
        },
        {
            "name": "b-devices",
            "test": "option[1234].hex == 0x0002"
        }
    ],
    "shared-networks": [
        {
            "name": "galah",
            "relay": {
                "ip-addresses":  [ "2001:db8:2:34::1" ]
            },
            "subnet6": [
                {
                    "subnet": "2001:db8:1::/64",
                    "pools": [ { "pool": "2001:db8:1::20 - 2001:db8:1::ff" } ],
                    "client-class": "a-devices"
                },
                {
                    "subnet": "2001:db8:3::/64",
                    "pools": [ { "pool": "2001:db8:3::20 - 2001:db8:3::ff" } ],
                    "client-class": "b-devices"
                }
            ]
        }
    ]
}
```

In this example each class has its own restriction. Only clients that belong to class "a-devices" will be able to use subnet 2001:db8:1::/64 and only clients belonging to "b-devices" will be able to use subnet 2001:db8:3::/64. Care should be taken not to define too-restrictive classification rules, as clients that are unable to use any subnets will be refused service. However, this may be a desired outcome if one wishes to provide service only to clients with known properties (e.g. only VoIP phones allowed on a given link).

Note that it is possible to achieve an effect similar to the one presented in this section without the use of shared networks. If the subnets are placed in the global subnets scope, rather than in the shared network, the server will still use classification rules to pick the right subnet for a given class of devices. The major benefit of placing subnets within the shared network is that common parameters for the logically grouped subnets can be specified once, in the shared network scope, e.g. the "interface" or "relay" parameter. All subnets belonging to this shared network will inherit those parameters.

9.4.3 Host Reservations in Shared Networks

Subnets that are part of a shared network allow host reservations, similar to regular subnets:

```
{
    "shared-networks": [
    {
        "name": "frog",
        "relay": {
            "ip-addresses": [ "2001:db8:2:34::1" ]
        },
        "subnet6": [
            {
                "subnet": "2001:db8:1::/64",
                "id": 100,
                "pools": [ { "2001:db8:1::1 - 2001:db8:1::64" } ],
                "reservations": [
                {
                    "duid": "00:03:00:01:11:22:33:44:55:66",
                    "ip-addresses": [ "2001:db8:1::28" ]
                }
                ]
            },
            {
                "subnet": "2001:db8:3::/64",
                "id": 101,
                "pools": [ { "pool": "2001:db8:3::1 - 2001:db8:3::64" } ],
                "reservations": [
                    {
                        "duid": "00:03:00:01:aa:bb:cc:dd:ee:ff",
                        "ip-addresses": [ "2001:db8:2::28" ]
                    }
                ]
            }
        ]
    }
    ]
}
```

It is worth noting that Kea conducts additional checks when processing a packet if shared networks are defined. First, instead of simply checking whether there's a reservation for a given client in its initially selected subnet, Kea looks through all subnets in a shared network for a reservation. This is one of the reasons why defining a shared network may impact performance. If there is a reservation for a client in any subnet, that particular subnet will be picked for the client. Although it is technically not an error, it is considered a bad practice to define reservations for the same host in multiple subnets belonging to the same shared network.

While not strictly mandatory, it is strongly recommended to use explicit "id" values for subnets if database storage will be used for host reservations. If an ID is not specified, the values for it are autogenerated, i.e. it assigns increasing integer values starting from 1. Thus, the autogenerated IDs are not stable across configuration changes.

9.5 Server Identifier in DHCPv6

The DHCPv6 protocol uses a "server identifier" (also known as a DUID) to allow clients to discriminate between several servers present on the same link. RFC 8415 currently defines four DUID types: DUID-LLT, DUID-EN, DUID-LL, and DUID-UUID.

The Kea DHCPv6 server generates a server identifier once, upon the first startup, and stores it in a file. This identifier is not modified across restarts of the server and so is a stable identifier.

Kea follows the recommendation from RFC 8415 to use DUID-LLT as the default server identifier. However, ISC has received reports that some deployments require different DUID types, and there is a need to administratively select both the DUID type and/or its contents.

The server identifier can be configured using parameters within the `server-id` map element in the global scope of the Kea configuration file. The following example demonstrates how to select DUID-EN as a server identifier:

```
"Dhcp6": {
    "server-id": {
        "type": "EN"
    },
    ...
}
```

Currently supported values for the `type` parameter are: "LLT", "EN", and "LL", for DUID-LLT, DUID-EN, and DUID-LL respectively.

When a new DUID type is selected, the server generates its value and replaces any existing DUID in the file. The server then uses the new server identifier in all future interactions with the clients.

Note: If the new server identifier is created after some clients have obtained their leases, the clients using the old identifier are not able to renew the leases; the server will ignore messages containing the old server identifier. Clients will continue sending Renew until they transition to the rebinding state. In this state, they will start sending Rebind messages to the multicast address without a server identifier. The server will respond to the Rebind messages with a new server identifier, and the clients will associate the new server identifier with their leases. Although the clients will be able to keep their leases and will eventually learn the new server identifier, this will be at the cost of an increased number of renewals and multicast traffic due to a need to rebind. Therefore, it is recommended that modification of the server identifier type and value be avoided if the server has already assigned leases and these leases are still valid.

There are cases when an administrator needs to explicitly specify a DUID value rather than allow the server to generate it. The following example demonstrates how to explicitly set all components of a DUID-LLT.

```
"Dhcp6": {
    "server-id": {
        "type": "LLT",
        "htype": 8,
        "identifier": "A65DC7410F05",
        "time": 2518920166
    },
    ...
}
```

where:

- `htype` is a 16-bit unsigned value specifying hardware type,
- `identifier` is a link-layer address, specified as a string of hexadecimal digits, and
- `time` is a 32-bit unsigned time value.

The hexadecimal representation of the DUID generated as a result of the configuration specified above is:

```
 00:01:00:08:96:23:AB:E6:A6:5D:C7:41:0F:05
|type |htype|   time    |   identifier    |
```

A special value of 0 for "htype" and "time" is allowed, which indicates that the server should use ANY value for these components. If the server already uses a DUID-LLT, it will use the values from this DUID; if the server uses a DUID of a different type or doesn't yet use any DUID, it will generate these values. Similarly, if the "identifier" is assigned an empty string, the value of the identifier will be generated. Omitting any of these parameters is equivalent to setting them to those special values.

For example, the following configuration:

```
"Dhcp6": {
    "server-id": {
        "type": "LLT",
        "htype": 0,
        "identifier": "",
        "time": 2518920166
    },
    ...
}
```

indicates that the server should use ANY link-layer address and hardware type. If the server is already using DUID-LLT, it will use the link-layer address and hardware type from the existing DUID. If the server is not yet using any DUID, it will use the link-layer address and hardware type from one of the available network interfaces. The server will use an explicit value of time; if it is different than a time value present in the currently used DUID, that value will be replaced, effectively modifying the current server identifier.

The following example demonstrates an explicit configuration of a DUID-EN:

```
"Dhcp6": {
    "server-id": {
        "type": "EN",
        "enterprise-id": 2495,
        "identifier": "87ABEF7A5BB545"
    },
    ...
}
```

where:

- `enterprise-id` is a 32-bit unsigned value holding an enterprise number, and
- `identifier` is a variable- length identifier within DUID-EN.

The hexadecimal representation of the DUID-EN created according to the configuration above is:

```
 00:02:00:00:09:BF:87:AB:EF:7A:5B:B5:45
|type |  ent-id   |      identifier     |
```

As in the case of the DUID-LLT, special values can be used for the configuration of the DUID-EN. If the `enterprise-id` is 0, the server will use a value from the existing DUID-EN. If the server is not using any DUID or the existing DUID has a different type, the ISC enterprise id will be used. When an empty string is entered for `identifier`, the identifier from the existing DUID-EN will be used. If the server is not using any DUID-EN, a new 6-byte-long identifier will be generated.

DUID-LL is configured in the same way as DUID-LLT except that the `time` parameter has no effect for DUID-LL, because this DUID type only comprises a hardware type and link-layer address. The following example demonstrates how to configure DUID-LL:

```
"Dhcp6": {
    "server-id": {
        "type": "LL",
        "htype": 8,
        "identifier": "A65DC7410F05"
    },
    ...
}
```

which will result in the following server identifier:

```
 00:03:00:08:A6:5D:C7:41:0F:05
|type |htype|   identifier    |
```

The server stores the generated server identifier in the following location: [kea-install-dir]/var/lib/kea/kea-dhcp6-serverid.

In some uncommon deployments where no stable storage is available, the server should be configured not to try to store the server identifier. This choice is controlled by the value of the `persist` boolean parameter:

```
"Dhcp6": {
    "server-id": {
        "type": "EN",
        "enterprise-id": 2495,
        "identifier": "87ABEF7A5BB545",
        "persist": false
    },
    ...
}
```

The default value of the "persist" parameter is `true`, which configures the server to store the server identifier on a disk.

In the example above, the server is configured not to store the generated server identifier on a disk. But if the server identifier is not modified in the configuration, the same value will be used after server restart, because the entire server identifier is explicitly specified in the configuration.

9.6 DHCPv6 data directory

The Kea DHCPv6 server puts the server identifier file and the default memory lease file into its data directory. By default this directory is `prefix/var/lib/kea` but this location can be changed using the `data-directory` global parameter as in:

```
"Dhcp6": {
    "data-directory": "/var/tmp/kea-server6",
    ...
}
```

9.7 Stateless DHCPv6 (Information-Request Message)

Typically DHCPv6 is used to assign both addresses and options. These assignments (leases) have a state that changes over time, hence their description as stateful. DHCPv6 also supports a stateless mode, where clients request configuration options only. This mode is considered lightweight from the server perspective, as it does not require any state tracking, and carries the stateless name.

The Kea server supports stateless mode. Clients can send Information-Request messages and the server sends back answers with the requested options, providing the options are available in the server configuration. The server attempts to use per-subnet options first; if that fails for any reason, it then tries to provide options defined in the global scope.

Stateless and stateful mode can be used together. No special configuration directives are required to handle this; simply use the configuration for stateful clients and the stateless clients will get only the options they requested.

This usage of global options allows for an interesting case. It is possible to run a server that provides only options and no addresses or prefixes. If the options have the same value in each subnet, the configuration can define required options in the global scope and skip subnet definitions altogether. Here's a simple example of such a configuration:

```
"Dhcp6": {
    "interfaces-config": {
        "interfaces": [ "ethX" ]
    },
    "option-data": [ {
        "name": "dns-servers",
        "data": "2001:db8::1, 2001:db8::2"
    } ],
    "lease-database": {
        "type": "memfile"
    }
 }
```

This very simple configuration provides DNS server information to all clients in the network, regardless of their location. Note the specification of the memfile lease database; this is needed as Kea requires a lease database to be specified even if it is not used.

9.8 Support for RFC 7550 (now part of RFC 8415)

RFC 7550 introduced some changes to the previous DHCPv6 specifications, RFC 3315 and RFC 3633, to resolve a few issues with the coexistence of multiple stateful options in the messages sent between clients and servers. Those changes were later included in the most recent DHCPv6 protocol specification, RFC 8415, which obsoleted RFC 7550. Kea supports RFC 8415 along with these protocol changes, which are briefly described below.

When a client, such as a requesting router, requests an allocation of both addresses and prefixes during the 4-way (SARR) exchange with the server, and the server is not configured to allocate any prefixes but it can allocate some addresses, it will respond with the IA_NA(s) containing allocated addresses and the IA_PD(s) containing the NoPrefixAvail status code. According to the updated specifications, if the client can operate without prefixes it should accept allocated addresses and transition to the "bound" state. When the client subsequently sends Renew/Rebind messages to the server, according to the T1 and T2 times, to extend the lifetimes of the allocated addresses, and if the client is still interested in obtaining prefixes from the server, it may also include an IA_PD in the Renew/Rebind to request allocation of the prefixes. If the server still cannot allocate the prefixes, it will respond with the IA_PD(s) containing the NoPrefixAvail status code. However, if the server can allocate the prefixes it will allocate and send them in the IA_PD(s) to the client. A similar situation occurs when the server is unable to allocate addresses for the client but can delegate prefixes. The client may request allocation of the addresses while renewing the delegated prefixes. Allocating leases for other IA types while renewing existing leases is by default supported by the Kea DHCPv6 server, and the server provides no configuration mechanisms to disable this behavior.

The following are the other behaviors first introduced in RFC 7550 (now part of RFC 8415) and supported by the Kea DHCPv6 server:

- Set T1/T2 timers to the same value for all stateful (IA_NA and IA_PD) options to facilitate renewal of all of a client's leases at the same time (in a single message exchange).
- Place NoAddrsAvail and NoPrefixAvail status codes in the IA_NA and IA_PD options in the Advertise message, rather than as the top-level options.

9.9 Using a Specific Relay Agent for a Subnet

The DHCPv6 server follows the same principles as the DHCPv4 server to select a subnet for the client with noticable differences mainly for relays.

Note: Starting with Kea 1.7.9, the order used to find a subnet which matches required conditions to be selected is the ascending subnet identifier order. When the selected subnet is a member of a shared network the whole shared network is selected.

The relay must have an interface connected to the link on which the clients are being configured. Typically the relay has a global IPv6 address configured on that interface, which belongs to the subnet from which the server will assign addresses. Normally, the server is able to use the IPv6 address inserted by the relay (in the link-addr field in RELAY-FORW message) to select the appropriate subnet.

However, that is not always the case. The relay address may not match the subnet in certain deployments. This usually means that there is more than one subnet allocated for a given link. The two most common examples where this is the case are long-lasting network renumbering (where both old and new address space is still being used) and a cable network. In a cable network, both cable modems and the devices behind them are physically connected to the same link, yet they use distinct addressing. In such a case, the DHCPv6 server needs additional information (like the value of the interface-id option or the IPv6 address inserted in the link-addr field in the RELAY-FORW message) to properly select an appropriate subnet.

The following example assumes that there is a subnet 2001:db8:1::/64 that is accessible via a relay that uses 3000::1 as its IPv6 address. The server is able to select this subnet for any incoming packets that come from a relay that has an address in the 2001:db8:1::/64 subnet. It also selects that subnet for a relay with address 3000::1.

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",
            "pools": [
                 {
                     "pool": "2001:db8:1::1-2001:db8:1::ffff"
                 }
             ],
             "relay": {
                 "ip-addresses": [ "3000::1" ]
             }
         }
     ]
}
```

If “relay” is specified, the “ip-addresses” parameter within it is mandatory.

Note: The current version of Kea uses the “ip-addresses” parameter, which supports specifying a list of addresses.

9.10 Segregating IPv6 Clients in a Cable Network

In certain cases, it is useful to mix relay address information (introduced in *Using a Specific Relay Agent for a Subnet*), with client classification, explained in *Client Classification*. One specific example is in a cable network, where modems typically get addresses from a different subnet than all the devices connected behind them.

Let us assume that there is one CMTS (Cable Modem Termination System) with one CM MAC (a physical link that modems are connected to). We want the modems to get addresses from the 3000::/64 subnet, while everything connected behind the modems should get addresses from another subnet (2001:db8:1::/64). The CMTS that acts as a relay uses address 3000::1. The following configuration can serve that configuration:

```
"Dhcp6": {
    "subnet6": [
        {
            "subnet": "3000::/64",
            "pools": [
                { "pool": "3000::2 - 3000::ffff" }
            ],
            "client-class": "VENDOR_CLASS_docsis3.0",
            "relay": {
                "ip-addresses": [ "3000::1" ]
            }
        },

        {
            "subnet": "2001:db8:1::/64",
            "pools": [
                 {
                     "pool": "2001:db8:1::1-2001:db8:1::ffff"
                 }
             ],
             "relay": {
                 "ip-addresses": [ "3000::1" ]
             }
        }
    ]
}
```

9.11 MAC/Hardware Addresses in DHCPv6

MAC/hardware addresses are available in DHCPv4 messages from the clients, and administrators frequently use that information to perform certain tasks like per-host configuration and address reservation for specific MAC addresses. Unfortunately, the DHCPv6 protocol does not provide any completely reliable way to retrieve that information. To mitigate that issue, a number of mechanisms have been implemented in Kea. Each of these mechanisms works in certain cases, but may fail in others. Whether the mechanism works in a particular deployment is somewhat dependent on the network topology and the technologies used.

Kea allows specification of which of the supported methods should be used and in what order. This configuration may be considered a fine tuning of the DHCP deployment. In a typical deployment the default value of `"any"` is sufficient and there is no need to select specific methods. Changing the value of this parameter is most useful in cases when an administrator wants to disable certain methods; for example, if the administrator trusts the network infrastructure more than the information provided by the clients themselves, they may prefer information provided by the relays over that provided by clients.

The configuration is controlled by the `mac-sources` parameter as follows:

```
"Dhcp6": {
    "mac-sources": [ "method1", "method2", "method3", ... ],

    "subnet6": [ ... ],

    ...
}
```

When not specified, a special value of “any” is used, which instructs the server to attempt to try all the methods in sequence and use the value returned by the first one that succeeds. If specified, it must have at least one value.

Supported methods are:

- `any` - not an actual method, just a keyword that instructs Kea to try all other methods and use the first one that succeeds. This is the default operation if no `mac-sources` are defined.
- `raw` - in principle, a DHCPv6 server could use raw sockets to receive incoming traffic and extract MAC/hardware address information. This is currently not implemented for DHCPv6 and this value has no effect.
- `duid` - DHCPv6 uses DUID identifiers instead of MAC addresses. There are currently four DUID types defined, and two of them (DUID-LLT, which is the default, and DUID-LL) convey MAC address information. Although RFC 8415 forbids it, it is possible to parse those DUIDs and extract necessary information from them. This method is not completely reliable, as clients may use other DUID types, namely DUID-EN or DUID-UUID.
- `ipv6-link-local` - another possible acquisition method comes from the source IPv6 address. In typical usage, clients are sending their packets from IPv6 link-local addresses. There is a good chance that those addresses are based on EUI-64, which contains a MAC address. This method is not completely reliable, as clients may use other link-local address types. In particular, privacy extensions, defined in RFC 4941, do not use MAC addresses. Also note that successful extraction requires that the address’s u-bit must be set to 1 and its g-bit set to 0, indicating that it is an interface identifier as per RFC 2373, section 2.5.1.
- `client-link-addr-option` - one extension defined to alleviate missing MAC issues is the client link-layer address option, defined in RFC 6939. This is an option that is inserted by a relay and contains information about a client’s MAC address. This method requires a relay agent that supports the option and is configured to insert it. This method is useless for directly connected clients. This parameter can also be specified as `rfc6939`, which is an alias for `client-link-addr-option`.
- `remote-id` - RFC 4649 defines a remote-id option that is inserted by a relay agent. Depending on the relay agent configuration, the inserted option may convey the client’s MAC address information. This parameter can also be specified as `rfc4649`, which is an alias for `remote-id`.

- `subscriber-id` - Another option that is somewhat similar to the previous one is subscriber-id, defined in RFC 4580. It, too, is inserted by a relay agent that is configured to insert it. This parameter can also be specified as `rfc4580`, which is an alias for `subscriber-id`. This method is currently not implemented.
- `docsis-cmts` - Yet another possible source of MAC address information are the DOCSIS options inserted by a CMTS that acts as a DHCPv6 relay agent in cable networks. This method attempts to extract MAC address information from sub-option 1026 (cm mac) of the vendor-specific option with vendor-id=4491. This vendor option is extracted from the relay-forward message, not the original client's message.
- `docsis-modem` - The final possible source of MAC address information are the DOCSIS options inserted by the cable modem itself. This method attempts to extract MAC address information from sub-option 36 (device id) of the vendor-specific option with vendor-id=4491. This vendor option is extracted from the original client's message, not from any relay options.

Empty mac-sources is not allowed. Administrators who do not want to specify it should either simply omit the mac-sources definition or specify it with the "any" value, which is the default.

9.12 Duplicate Addresses (DECLINE Support)

The DHCPv6 server is configured with a certain pool of addresses that it is expected to hand out to DHCPv6 clients. It is assumed that the server is authoritative and has complete jurisdiction over those addresses. However, for various reasons, such as misconfiguration or a faulty client implementation that retains its address beyond the valid lifetime, there may be devices connected that use those addresses without the server's approval or knowledge.

Such an unwelcome event can be detected by legitimate clients (using Duplicate Address Detection) and reported to the DHCPv6 server using a DHCPDECLINE message. The server will do a sanity check (to see whether the client declining an address really was supposed to use it), and then will conduct a clean-up operation and confirm it by sending back a REPLY message. Any DNS entries related to that address will be removed, the fact will be logged, and hooks will be triggered. After that is complete, the address will be marked as declined (which indicates that it is used by an unknown entity and thus not available for assignment) and a probation time will be set on it. Unless otherwise configured, the probation period lasts 24 hours; after that period, the server will recover the lease (i.e. put it back into the available state) and the address will be available for assignment again. It should be noted that if the underlying issue of a misconfigured device is not resolved, the duplicate-address scenario will repeat. If reconfigured correctly, this mechanism provides an opportunity to recover from such an event automatically, without any system administrator intervention.

To configure the decline probation period to a value other than the default, the following syntax can be used:

```
  "Dhcp6": {
    "decline-probation-period": 3600,
    "subnet6": [ ... ],
    ...
}
```

The parameter is expressed in seconds, so the example above will instruct the server to recycle declined leases after one hour.

There are several statistics and hook points associated with the Decline handling procedure. The lease6_decline hook is triggered after the incoming DHCPDECLINE message has been sanitized and the server is about to decline the lease. The declined-addresses statistic is increased after the hook returns (both global and subnet-specific variants). (See *Statistics in the DHCPv6 Server* and *Hooks Libraries* for more details on DHCPv6 statistics and Kea hook points.)

Once the probation time elapses, the declined lease is recovered using the standard expired-lease reclamation procedure, with several additional steps. In particular, both declined-addresses statistics (global and subnet-specific) are decreased. At the same time, reclaimed-declined-addresses statistics (again in two variants, global and subnet-specific) are increased.

A note about statistics: The server does not decrease the assigned-nas statistics when a DHCPDECLINE message is received and processed successfully. While technically a declined address is no longer assigned, the primary usage of the assigned-nas statistic is to monitor pool utilization. Most people would forget to include declined-addresses in the calculation, and simply use assigned-nas/total-nas. This would cause a bias towards under-representing pool utilization. As this has a potential for major issues, ISC decided not to decrease assigned-nas immediately after receiving DHCPDECLINE, but to do it later when Kea recovers the address back to the available pool.

9.13 Statistics in the DHCPv6 Server

The DHCPv6 server supports the following statistics:

Table 9.4: DHCPv6 Statistics

Statistic	Data Type	Description
pkt6-received	integer	Number of DHCPv6 packets received. This includes all packets: valid, bogus, corrupted, rejected, etc. This statistic is expected to grow rapidly.
pkt6-receive-drop	integer	Number of incoming packets that were dropped. The exact reason for dropping packets is logged, but the most common reasons may be: an unacceptable or not supported packet type is received, direct responses are forbidden, the server-id sent by the client does not match the server's server-id, or the packet is malformed.
pkt6-parse-failed	integer	Number of incoming packets that could not be parsed. A non-zero value of this statistic indicates that the server received a malformed or truncated packet. This may indicate problems in the network, faulty clients, faulty relay agents, or a bug in the server.
pkt6-solicit-received	integer	Number of SOLICIT packets received. This statistic is expected to grow; its increase means that clients that just booted started their configuration process and their initial packets reached the Kea server.
pkt6-advertise-received	integer	Number of ADVERTISE packets received. Advertise packets are sent by the server and the server is never expected to receive them. A non-zero value of this statistic indicates an error occurring in the network. One likely cause would be a misbehaving relay agent that incorrectly forwards ADVERTISE messages towards the server, rather than back to the clients.
pkt6-request-received	integer	Number of DHCPREQUEST packets received. This statistic is expected to grow. Its increase means that clients that just booted received the server's response (DHCPADVERTISE) and accepted it, and are now requesting an address (DHCPREQUEST).
pkt6-reply-received	integer	Number of REPLY packets received. This statistic is expected to remain zero at all times, as REPLY packets are sent by the server and the server is never expected to receive them. A non-zero value indicates an error. One likely cause would be a misbehaving relay agent that incorrectly forwards REPLY messages towards the server, rather than back to the clients.
pkt6-renew-received	integer	Number of RENEW packets received. This statistic is expected to grow; its increase means that clients received their addresses and prefixes and are trying to renew them.
pkt6-rebind-received	integer	Number of REBIND packets received. A non-zero value indicates that clients did not receive responses to their RENEW messages (through the regular lease-renewal mechanism) and are attempting to find any server that is able to take over their leases. It may mean that some servers' REPLY messages never reached the clients.

Continued on next page

Table 9.4 – continued from previous page

Statistic	Data Type	Description
pkt6-release-received	integer	Number of RELEASE packets received. This statistic is expected to grow when a device is being shut down in the network; it indicates that the address or prefix assigned is reported as no longer needed. Note that many devices, especially wireless, do not send RELEASE packets either because of design choice or due to the client moving out of range.
pkt6-decline-received	integer	Number of DECLINE packets received. This statistic is expected to remain close to zero. Its increase means that a client leased an address, but discovered that the address is currently used by an unknown device in the network. If this statistic is growing, it may indicate a misconfigured server or devices that have statically assigned conflicting addresses.
pkt6-infrequest-received	integer	Number of INFORMATION-REQUEST packets received. This statistic is expected to grow if there are devices that are using stateless DHCPv6. INFORMATION-REQUEST messages are used by clients that request stateless configuration, i.e. options and parameters other than addresses or prefixes.
pkt6-dhcpv4-query-received	integer	Number of DHCPv4-QUERY packets received. This statistic is expected to grow if there are devices that are using DHCPv4-over-DHCPv6. DHCPv4-QUERY messages are used by DHCPv4 clients on an IPv6-only line which encapsulates the requests over DHCPv6.
pkt6-dhcpv4-response-received	integer	Number of DHCPv4-RESPONSE packets received. This statistic is expected to remain zero at all times, as DHCPv4-RESPONSE packets are sent by the server and the server is never expected to receive them. A non-zero value indicates an error. One likely cause would be a misbehaving relay agent that incorrectly forwards DHCPv4-RESPONSE message towards the server rather than back to the clients.
pkt6-unknown-received	integer	Number of packets received of an unknown type. A non-zero value of this statistic indicates that the server received a packet that it wasn't able to recognize; either it had an unsupported type or was possibly malformed.
pkt6-sent	integer	Number of DHCPv6 packets sent. This statistic is expected to grow every time the server transmits a packet. In general, it should roughly match pkt6-received, as most incoming packets cause the server to respond. There are exceptions (e.g. server receiving a REQUEST with server-id matching other server), so do not worry if it is less than pkt6-received.
pkt6-advertise-sent	integer	Number of ADVERTISE packets sent. This statistic is expected to grow in most cases after a SOLICIT is processed. There are certain uncommon, but valid, cases where incoming SOLICIT packets are dropped, but in general this statistic is expected to be close to pkt6-solicit-received.
pkt6-reply-sent	integer	Number of REPLY packets sent. This statistic is expected to grow in most cases after a SOLICIT (with rapid-commit), REQUEST, RENEW, REBIND, RELEASE, DECLINE, or INFORMATION-REQUEST is processed. There are certain cases where there is no response.
pkt6-dhcpv4-response-sent	integer	Number of DHCPv4-RESPONSE packets sent. This statistic is expected to grow in most cases after a DHCPv4-QUERY is processed. There are certain cases where there is no response.
subnet[id].total-nas	integer	Total number of NA addresses available for DHCPv6 management for a given subnet; in other words, this is the sum of all addresses in all configured pools. This statistic changes only during configuration changes. Note that it does not take into account any addresses that may be reserved due to host reservation. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.

Continued on next page

Table 9.4 – continued from previous page

Statistic	Data Type	Description
cumulative-assigned-nas	integer	Cumulative number of NA addresses that have been assigned since server startup. It is incremented each time a NA address is assigned and is not reset when the server is reconfigured.
subnet[id].cumulative-assigned-nas	integer	Cumulative number of NA addresses in a given subnet that were assigned. It increases every time a new lease is allocated (as a result of receiving a REQUEST message) and is never decreased. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.
subnet[id].assigned-nas	integer	Number of NA addresses in a given subnet that are assigned. It increases every time a new lease is allocated (as a result of receiving a REQUEST message) and is decreased every time a lease is released (a RELEASE message is received) or expires. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.
subnet[id].total-pds	integer	Total number of PD prefixes available for DHCPv6 management for a given subnet; in other words, this is the sum of all prefixes in all configured pools. This statistic changes only during configuration changes. Note it does not take into account any prefixes that may be reserved due to host reservation. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.
cumulative-assigned-pds	integer	Cumulative number of PD prefixes that have been assigned since server startup. It is incremented each time a PD prefix is assigned and is not reset when the server is reconfigured.
subnet[id].cumulative-assigned-pds	integer	Cumulative number of PD prefixes in a given subnet that were assigned. It increases every time a new lease is allocated (as a result of receiving a REQUEST message) and is never decreased. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.
subnet[id].assigned-pds	integer	Number of PD prefixes in a given subnet that are assigned. It increases every time a new lease is allocated (as a result of receiving a REQUEST message) and is decreased every time a lease is released (a RELEASE message is received) or expires. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately, and is reset during a reconfiguration event.
reclaimed-leases	integer	Number of expired leases that have been reclaimed since server startup. It is incremented each time an expired lease is reclaimed (counting both NA and PD reclamations). This statistic never decreases. It can be used as a long-term indicator of how many actual leases have been reclaimed. This is a global statistic that covers all subnets.
subnet[id].reclaimed-leases	integer	Number of expired leases associated with a given subnet (*"id"* is the subnet-id) that have been reclaimed since server startup. It is incremented each time an expired lease is reclaimed (counting both NA and PD reclamations). The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately.
declined-addresses	integer	Number of IPv6 addresses that are currently declined; a count of the number of leases currently unavailable. Once a lease is recovered, this statistic will be decreased; ideally, this statistic should be zero. If this statistic is non-zero or increasing, a network administrator should investigate whether there is a misbehaving device in the network. This is a global statistic that covers all subnets.

Continued on next page

Table 9.4 – continued from previous page

Statistic	Data Type	Description
subnet[id].declined-addresses	integer	Number of IPv6 addresses that are currently declined in a given subnet; a count of the number of leases currently unavailable. Once a lease is recovered, this statistic will be decreased; ideally, this statistic should be zero. If this statistic is non-zero or increasing, a network administrator should investigate whether there is a misbehaving device in the network. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately.
reclaimed-declined-addresses	integer	Number of IPv6 addresses that were declined, but have now been recovered. Unlike declined-addresses, this statistic never decreases. It can be used as a long-term indicator of how many actual valid Declines were processed and recovered from. This is a global statistic that covers all subnets.
subnet[id].reclaimed-declined-addresses	integer	Number of IPv6 addresses that were declined, but have now been recovered. Unlike declined-addresses, this statistic never decreases. It can be used as a long-term indicator of how many actual valid Declines were processed and recovered from. The *id* is the subnet-id of a given subnet. This statistic is exposed for each subnet separately.

Note: This section describes DHCPv6-specific statistics. For a general overview and usage of statistics, see *Statistics*.

Beginning with Kea 1.7.7 the DHCPv6 server provides two global parameters to control statistics default sample limits:

- `statistic-default-sample-count` - determines the default maximum number of samples which will be kept. The special value of zero means to use a default maximum age.
- `statistic-default-sample-age` - determines the default maximum age in seconds of samples which will be kept.

For instance to reduce the statistic keeping overhead you can set the default maximum sample count to 1 so only one sample will be kept by:

```
  "Dhcp6": {
    "statistic-default-sample-count": 1,
    "subnet6": [ ... ],
    ...
}
```

Statistics can be retrieved periodically to gain more insight into Kea operations. One tool that leverages that capability is ISC Stork. See *Monitoring Kea with Stork* for details.

9.14 Management API for the DHCPv6 Server

The management API allows the issuing of specific management commands, such as statistics retrieval, reconfiguration, or shutdown. For more details, see *Management API*. Currently, the only supported communication channel type is UNIX stream socket. By default there are no sockets open; to instruct Kea to open a socket, the following entry in the configuration file can be used:

```
"Dhcp6": {
    "control-socket": {
        "socket-type": "unix",
        "socket-name": "/path/to/the/unix/socket"
    },

    "subnet6": [
        ...
    ],
    ...
}
```

The length of the path specified by the `socket-name` parameter is restricted by the maximum length for the UNIX socket name on the administrator's operating system, i.e. the size of the `sun_path` field in the `sockaddr_un` structure, decreased by 1. This value varies on different operating systems between 91 and 107 characters. Typical values are 107 on Linux and 103 on FreeBSD.

Communication over the control channel is conducted using JSON structures. See the Control Channel section in the Kea Developer's Guide for more details.

The DHCPv6 server supports the following operational commands:

- build-report
- config-get
- config-reload
- config-set
- config-test
- config-write
- dhcp-disable
- dhcp-enable
- leases-reclaim
- list-commands
- shutdown
- status-get
- version-get

as described in *Commands Supported by Both the DHCPv4 and DHCPv6 Servers*. In addition, it supports the following statistics-related commands:

- statistic-get
- statistic-reset
- statistic-remove
- statistic-get-all
- statistic-reset-all
- statistic-remove-all
- statistic-sample-age-set
- statistic-sample-age-set-all
- statistic-sample-count-set
- statistic-sample-count-set-all

as described in *Commands for Manipulating Statistics*.

9.15 User Contexts in IPv6

Kea allows loading hook libraries that can sometimes benefit from additional parameters. If such a parameter is specific to the whole library, it is typically defined as a parameter for the hook library. However, sometimes there is a need to specify parameters that are different for each pool.

See *Comments and User Context* for additional background regarding the user context idea. See *User Contexts in Hooks* for a discussion from the hooks perspective.

User contexts can be specified at global scope, shared network, subnet, pool, client class, option data, or definition level, and via host reservation. One other useful feature is the ability to store comments or descriptions.

Let's consider a lightweight 4over6 deployment as an example. It is an IPv6 transition technology that allows mapping IPv6 prefixes into full or partial IPv4 addresses. In the DHCP context, these are specific parameters that are supposed to be delivered to clients in the form of additional options. Values of these options are correlated to delegated prefixes, so it is reasonable to keep these parameters together with the PD pool. On the other hand, lightweight 4over6 is not a commonly used feature, so it is not a part of the base Kea code. The solution to this problem is to specify a user context. For each PD pool that is expected to be used for lightweight 4over6, a user context with extra parameters is defined. Those extra parameters will be used by a hook library and loaded only when dynamic calculation of the lightweight 4over6 option is actually needed. An example configuration looks as follows:

```
"Dhcp6": {
    "subnet6": [ {
        "pd-pools": [
        {
            "prefix":  "2001:db8::",
            "prefix-len": 56,
            "delegated-len": 64,

            # This is a pool specific context.
            "user-context": {
                "threshold-percent": 85,
                "v4-network": "192.168.0.0/16",
                "v4-overflow": "10.0.0.0/16",
                "lw4over6-sharing-ratio": 64,
                "lw4over6-v4-pool": "192.0.2.0/24",
                "lw4over6-sysports-exclude": true,
                "lw4over6-bind-prefix-len": 56
            }
        } ],
        "subnet": "2001:db8::/32",

        # This is a subnet-specific context. Any type of
        # information can be entered here as long as it is valid JSON.
        "user-context": {
            "comment": "Those v4-v6 migration technologies are tricky.",
            "experimental": true,
            "billing-department": 42,
            "contact-points": [ "Alice", "Bob" ]
        }
    } ],
    ...
}
```

Kea does not interpret or use the user context information; it simply stores it and makes it available to the hook libraries. It is up to each hook library to extract that information and use it. The parser translates a "comment" entry into a user context with the entry, which allows a comment to be attached inside the configuration itself.

9.16 Supported DHCPv6 Standards

The following standards are currently supported:

- *Dynamic Host Configuration Protocol for IPv6*, RFC 3315: Supported messages are SOLICIT, ADVERTISE, REQUEST, RELEASE, RENEW, REBIND, INFORMATION-REQUEST, CONFIRM, DECLINE and REPLY. The only not supported message is RECONFIGURE.
- *Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers*, RFC 3319: All defined options are supported.
- *IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6*, RFC 3633: Supported options are IA_PD and IA_PREFIX. Also supported is the status code NoPrefixAvail.
- *DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, RFC 3646: All defined options are supported.
- *Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6*, RFC 3736: The server operation in stateless mode is supported. Kea is currently server only, so the client side is not implemented.
- *Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, RFC 4242: The sole defined option (information-refresh-time) is supported.
- *The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Relay Agent Remote-ID Option*, RFC 4649: REMOTE-ID option is supported.
- *Resolution of Fully Qualified Domain Name (FQDN) Conflicts among Dynamic Host Configuration Protocol (DHCP) Clients*, RFC 4703: The DHCPv6 server uses DHCP-DDNS server to resolve conflicts.
- *The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option*, RFC 4704: Supported option is CLIENT_FQDN.
- *Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Option for Dual-Stack Lite*, RFC 6334: the AFTR-Name DHCPv6 Option is supported.
- *Relay-Supplied DHCP Options*, RFC 6422: Full functionality is supported: OPTION_RSOO, ability of the server to echo back the options, checks whether an option is RSOO-enabled, ability to mark additional options as RSOO-enabled.
- *Prefix Exclude Option for DHCPv6-based Prefix Delegation*, RFC 6603: Prefix Exclude option is supported.
- *Client Link-Layer Address Option in DHCPv6*, RFC 6939: Supported option is client link-layer address option.
- *Issues and Recommendations with Multiple Stateful DHCPv6 Options*, RFC 7550: All recommendations related to the DHCPv6 server operation are supported.
- *DHCPv6 Options for Configuration of Softwire Address and Port-Mapped Clients*, RFC 7598: All options indicated in this specification are supported by the DHCPv6 server.
- *Generalized UDP Source Port for DHCP Relay*, RFC 8357: The Kea server is able to handle Relay Source Port option in a received Relay-Forward message, remembers the UDP port and sends back Relay-Reply with a copy of the option to the relay agent using this UDP port.
- *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, RFC 8415: New DHCPv6 protocol specification which obsoletes RFC 3315, RFC 3633, RFC 3736, RFC 4242, RFC 7083, RFC 7283, and RFC 7550. All features, with the exception of Reconfigure mechanism and the now deprecated temporary addresses (IA_TA) mechanism, are supported.

9.17 DHCPv6 Server Limitations

These are the current limitations of the DHCPv6 server software. Most of them are reflections of the current stage of development and should be treated as "not implemented yet", rather than actual limitations.

- The server will allocate, renew, or rebind a maximum of one lease for a particular IA option (IA_NA or IA_PD) sent by a client. RFC 8415 allows for multiple addresses or prefixes to be allocated for a single IA.
- Temporary addresses are not supported. There is no intention to ever implement this feature, as it is deprecated in RFC8415.
- Client reconfiguration (RECONFIGURE) is not yet supported.

9.18 Kea DHCPv6 server examples

A collection of simple-to-use examples for the DHCPv6 component of Kea is available with the source files, located in the doc/examples/kea6 directory.

9.19 Configuration Backend in DHCPv6

In the *Kea Configuration Backend* section we have described the Configuration Backend feature, its applicability, and its limitations. This section focuses on the usage of the CB with the DHCPv6 server. It lists the supported parameters, describes limitations, and gives examples of the DHCPv6 server configuration to take advantage of the CB. Please also refer to the sibling section *Configuration Backend in DHCPv4* for the DHCPv4-specific usage of the CB.

9.19.1 Supported Parameters

The ultimate goal for the CB is to serve as a central configuration repository for one or multiple Kea servers connected to the database. In the future it will be possible to store most of the server's configuration in the database and reduce the configuration file to a bare minimum; the only mandatory parameter will be the `config-control`, which includes the necessary information to connect to the database. In the Kea 1.6.0 release, however, only a subset of the DHCPv4 server parameters can be stored in the database. All other parameters must be specified in the JSON configuration file, if required.

The following table lists DHCPv6-specific parameters supported by the Configuration Backend, with an indication on which level of the hierarchy it is currently supported. "n/a" is used in cases when a particular parameter is not applicable on a particular level of the hierarchy, or in cases when the parameter is not supported by the server at this level of the hierarchy. "no" is used when the parameter is supported by the server on the given level of the hierarchy, but is not configurable via the Configuration Backend.

All supported parameters can be configured via `cb_cmds` hooks library described in the *cb_cmds: Configuration Backend Commands* section. The general rule is that the scalar global parameters are set using the `remote-global-parameter6-set`; the shared network-specific parameters are set using `remote-network6-set`; and the subnet- and pool-level parameters are set using `remote-subnet6-set`. Whenever there is an exception to this general rule, it is highlighted in the table. The non-scalar global parameters have dedicated commands; for example, the global DHCPv6 options (`option-data`) are modified using `remote-option6-global-set`.

Table 9.5: List of DHCPv6 Parameters Supported by the Configuration Backend

Parameter	Global	Shared Network	Subnet	Pool	Prefix Delegation Pool
calculate-tee-times	yes	yes	yes	n/a	n/a
client-class	n/a	yes	yes	yes	yes
ddns-send-update	yes	yes	yes	n/a	n/a
ddns-override-no-update	yes	yes	yes	n/a	n/a
ddns-override-client-update	yes	yes	yes	n/a	n/a
ddns-replace-client-name	yes	yes	yes	n/a	n/a
ddns-generated-prefix	yes	yes	yes	n/a	n/a
ddns-qualifying-suffix	yes	yes	yes	n/a	n/a
decline-probation-period	yes	n/a	n/a	n/a	n/a
delegated-len	n/a	n/a	n/a	n/a	yes
dhcp4o6-port	yes	n/a	n/a	n/a	n/a
excluded-prefix	n/a	n/a	n/a	n/a	yes
excluded-prefix-len	n/a	n/a	n/a	n/a	yes
hostname-char-set	no	no	no	n/a	n/a
hostname-char-replacement	no	no	no	n/a	n/a
interface	n/a	yes	yes	n/a	n/a
interface-id	n/a	yes	yes	n/a	n/a
option-data	yes (via remote-option6-global-set)	yes	yes	yes	yes
option-def	yes (via remote-option-def6-set)	n/a	n/a	n/a	n/a
preferred-lifetime	yes	yes	yes	n/a	n/a
prefix	n/a	n/a	n/a	n/a	yes
prefix-len	n/a	n/a	n/a	n/a	yes
rapid-commit	yes	yes	yes	n/a	n/a
rebind-timer	yes	yes	yes	n/a	n/a
relay	n/a	yes	yes	n/a	n/a
renew-timer	yes	yes	yes	n/a	n/a
require-client-classes	n/a	yes	yes	yes	yes
reservation-mode	yes	yes	yes	n/a	n/a
t1-percent	yes	yes	yes	n/a	n/a
t2-percent	yes	yes	yes	n/a	n/a
valid-lifetime	yes	yes	yes	n/a	n/a

9.19.2 Enabling Configuration Backend

The following configuration snippet demonstrates how to enable the MySQL Configuration Backend for the DHCPv6 server:

```
{
    "Dhcp6": {
    "server-tag": "my DHCPv6 server",
        "config-control": {
            "config-databases": [
                {
                    "type": "mysql",
                    "name": "kea",
                    "user": "kea",
                    "password": "kea",
                    "host": "2001:db8:1::1",
                    "port": 3302
                }
            ],
            "config-fetch-wait-time": 20
        },
        "hooks-libraries": [
            {
                "library": "/usr/local/lib/kea/hooks/libdhcp_mysql_cb.so"
            },
            {
                "library": "/usr/local/lib/kea/hooks/libdhcp_cb_cmds.so"
            }
        ],
        ...
    }
}
```

The configuration structure is almost identical to that of the DHCPv4 server (see *Enabling Configuration Backend* for the detailed description).

CHAPTER

TEN

DATABASE CONNECTIVITY

Kea servers (kea-dhcp4 and kea-dhcp6) can be configured to use a variety of database back ends for leases, hosts and configuration. All of them may be configured to support automatic recovery when connectivity is lost (see `max-reconnect-tries` and `reconnect-wait-time`).

It is important to understand how and when automatic recovery comes into play. Automatic recovery, when configured, only operates after a successful start up or reconfiguration during which connectivity to all back ends has been successfully established.

During server start up the inability to connect to any of the configured back ends is always considered fatal. A fatal error will be logged and the server will exit. This is done based on the idea that configuration should be valid at start up. Exiting to the operating system allows nanny scripts to detect the problem.

During dynamic reconfiguration, all back ends are disconnected and then reconnected using the new configuration. If connectivity to any of the back ends cannot be established, the server will log a fatal error but remain up. It will be able to process commands but will not serve clients. This allows the configuration to be corrected via command, if required.

During normal operations, if connectivity to any of the back ends is lost and automatic recovery for that back end is enabled, the server will disconnect from all back ends, and then attempt to reconnect them. During the recovery process the server will cease serving clients but continue to respond to commands. If connectivity to all back ends is restored the server will return to normal operations. If connectivity cannot be restored after `max-reconnect-tries`, the server will issue a fatal error and exit.

CHAPTER
ELEVEN

LEASE EXPIRATION

The primary role of the DHCP server is to assign addresses and/or delegate prefixes to DHCP clients. These addresses and prefixes are often referred to as “leases.” Leases are typically assigned to clients for a finite amount of time, known as the “valid lifetime.” DHCP clients who wish to continue using their assigned leases will periodically renew them by sending the appropriate message to the DHCP server. The DHCP server records the time when these leases are renewed and calculates new expiration times for them.

If the client does not renew a lease before its valid lifetime elapses, the lease is considered expired. There are many situations when the client may cease lease renewals; a common scenario is when the machine running the client shuts down for an extended period of time.

The process through which the DHCP server makes expired leases available for reassignment is referred to as “lease reclamation” and expired leases returned to availability through this process are referred to as “reclaimed.” The DHCP server attempts to reclaim an expired lease as soon as it detects that it has expired. The server has several possible ways to detect expiration: it may attempt to allocate a lease to a client but find this lease already present in the database and expired; or it can periodically query the lease database for expired leases. Regardless of how an expired lease is detected, it must be reclaimed before it can be assigned to a client.

This chapter explains how to configure the server to periodically query for the expired leases, and how to minimize the impact of the periodic lease reclamation process on the server’s responsiveness. Finally, it explains “lease affinity,” which provides the means to assign the same lease to a returning client after its lease has expired.

Although all configuration examples in this section are provided for the DHCPv4 server, the same parameters may be used for DHCPv6 server configuration.

11.1 Lease Reclamation

Lease reclamation is the process through which an expired lease becomes available for assignment to the same or a different client. This process involves the following steps for each reclaimed lease:

- Invoke callouts for the `lease4_expire` or `lease6_expire` hook points, if hook libraries supporting those callouts are currently loaded.
- Update the DNS, i.e. remove any DNS entries associated with the expired lease.
- Update lease information in the lease database to indicate that the lease is now available for re-assignment.
- Update counters on the server, a process that includes increasing the number of reclaimed leases and decreasing the number of assigned addresses or delegated prefixes.

Please refer to *The DHCP-DDNS Server* to see how to configure DNS updates in Kea, and to *Hooks Libraries* for information about using hooks libraries.

11.2 Lease Reclamation Configuration Parameters

The following list presents all configuration parameters pertaining to processing expired leases with their default values:

- `reclaim-timer-wait-time` - this parameter governs intervals between the completion of the previous reclaimation cycle and the start of the next one. Specified in seconds. The default value is 10 [seconds].
- `flush-reclaimed-timer-wait-time` - this parameter controls how often the server initiates the lease reclaimation procedure. Expressed in seconds. The default value is 25 [seconds].
- `hold-reclaimed-time` - this parameter governs how long the lease should be kept after it is reclaimed. This enables lease affinity when set to a non-zero value. Expressed in seconds. The default value is 3600 [seconds].
- `max-reclaim-leases` - this parameter specifies the maximum number of reclaimed leases that can be processed at one time. Zero means unlimited (i.e. process all reclaimed leases). The default value is 100.
- `max-reclaim-time` - this parameter specifies an upper limit to the length of time a lease reclamation procedure can take. Zero means no time limit. Expressed in milliseconds. The default value is 250 [milliseconds].
- `unwarned-reclaim-cycles` - if lease reclamation limits are specified (`max-reclaim-leases` and/or `max-reclaim-time`), then under certain circumstances the server may not be able to deal with the leases to be reclaimed fast enough. This parameter specifies how many consecutive clean-up cycles must end with remaining leases to be processed before a warning is printed. The default is 5 [cycles].

The parameters are explained in more detail in the rest of this chapter.

The default value for any parameter is used when the parameter is not explicitly specified in the configuration. If the `expired-leases-processing` map is omitted entirely in the configuration, the default values are used for all parameters listed above.

11.3 Configuring Lease Reclamation

Kea can be configured to periodically detect and reclaim expired leases. During this process the lease entries in the database are modified or removed. While this is happening, the server will not process incoming DHCP messages to avoid issues with concurrent access to database information. As a result, the server will be unresponsive while lease reclamation is performed and DHCP queries will accumulate; responses will be sent once the lease-reclamation cycle is complete.

In deployments where response time is critical, administrators may wish to minimize the interruptions in service caused by lease reclamation. To this end, Kea provides configuration parameters to control the frequency of lease reclamation cycles, the maximum number of leases processed in a single reclamation cycle, and the maximum amount of time a single reclamation cycle is allowed to run before being interrupted. The following examples demonstrate how these parameters can be used:

```
"Dhcp4": {
    ...

    "expired-leases-processing": {
        "reclaim-timer-wait-time": 5,
        "max-reclaim-leases": 0,
        "max-reclaim-time": 0,
    },

    ...
}
```

The first parameter is expressed in seconds and specifies an interval between the two consecutive lease reclamation cycles. This is explained by the following diagram:

```
|  c1  |            | c2 |            |c3|            | c4 |
|<---->|<---------->|<-->|<---------->|<>|<---------->|<-->|<--
----------------------------------------------------------------->
|      |     5s     |    |     5s     |  |     5s     |    | time
```

This diagram shows four lease-reclamation cycles (c1 through c4) of variable duration. Note that the duration of the reclamation cycle depends on the number of expired leases detected and processed in the particular cycle. This duration is usually significantly shorter than the interval between the cycles.

According to the `reclaim-timer-wait-time`, the server keeps fixed intervals of five seconds between the end of one cycle and the start of the next cycle. This guarantees the presence of 5-second-long periods during which the server remains responsive to DHCP queries and does not perform lease reclamation. The `max-reclaim-leases` and `max-reclaim-time` are set to 0, which sets no restriction on the maximum number of leases reclaimed in the particular cycle, or on the maximum duration of each cycle.

In deployments with high lease-pool utilization, relatively short valid lifetimes, and frequently disconnecting clients which allow leases to expire, the number of expired leases requiring reclamation at any given time may rise significantly. In this case, it is often desirable to apply restrictions to the maximum duration of a reclamation cycle or the maximum number of leases reclaimed in a cycle. The following configuration demonstrates how this can be done:

```
"Dhcp4": {
    ...

    "expired-leases-processing": {
        "reclaim-timer-wait-time": 3,
        "max-reclaim-leases": 100,
        "max-reclaim-time": 50,
        "unwarned-reclaim-cycles": 10,
    },

    ...
}
```

The `max-reclaim-leases` parameter limits the number of leases reclaimed in a single cycle to 100. The `max-reclaim-time` limits the maximum duration of each cycle to 50ms. The lease-reclamation cycle will be interrupted if either of these limitations is reached. The reclamation of any unreclaimed leases will be attempted in subsequent cycles.

The following diagram illustrates the behavior of the system in the presence of many expired leases, when the limits are applied for the reclamation cycles:

```
| c1 |               | c2 |               | c3 |               | c4 |
|<-->|<------------->|<-->|<------------->|<-->|<------------->|<-->|<--
---------------------------------------------------------------------------->
|50ms|      3s       |50ms|      3s       |50ms|      3s       |50ms|  time
```

This diagram demonstrates the case when each reclamation cycle takes more than 50ms, and thus is interrupted according to the value of the `max-reclaim-time`. This results in equal durations of all reclamation cycles over time. Note that in this example the limitation of the maximum 100 leases is not reached. This may be the case when database transactions or callouts in the hook libraries attached to the server are slow. Regardless, the chosen values for either the maximum number of leases or a maximum cycle time strongly depend on the particular deployment, the lease database backend being used, and any hooks libraries, etc. Administrators may need to experiment to tune the system to suit the dynamics of their deployment.

It is important to realize that with the use of these limits, there is a risk that expired leases will accumulate faster than the server can reclaim them. This should not be a problem if the server is dealing with a temporary burst of expirations, because it should be able to eventually deal with them over time. However, if leases expire at a high rate for a long period of time, the unreclaimed leases will pile up in the database. To notify the administrator that the current configuration does not satisfy the needs for reclamation of expired leases, the server issues a warning message in the log if it is unable to reclaim all leases within several reclamation cycles. The number of cycles after which such a warning is issued is specified with the `unwarned-reclaim-cycles` configuration parameter.

Setting the `reclaim-timer-wait-time` to 0 disables periodic reclamation of the expired leases.

11.4 Configuring Lease Affinity

Suppose that a laptop goes into sleep mode after a period of user inactivity. While the laptop is in sleep mode, its DHCP client will not renew leases obtained from the server and these leases will eventually expire. When the laptop wakes up, it is often desirable for it to continue using its previous assigned IP addresses. To facilitate this, the server needs to correlate returning clients with their expired leases. When the client returns, the server will first check for those leases and re-assign them if they have not been assigned to another client. The ability of the server to re-assign the same lease to a returning client is referred to as “lease affinity.”

When lease affinity is enabled (i.e. when `hold-reclaimed-time` is configured to a value greater than zero), the server will still reclaim leases according to the parameters described in *Configuring Lease Reclamation*, but the reclaimed leases will be held in the database for a specified amount of time rather than removed. When the client returns, the server will first verify whether there are any reclaimed leases associated with this client and will re-assign them if possible. However, it is important to note that any reclaimed lease may be assigned to another client if that client specifically asks for it. Therefore, lease affinity does not guarantee that the reclaimed lease will be available for the client who used it before; it merely increases the chances of the client being assigned the same lease. If the lease pool is small - namely, in DHCPv4, for which address space is small - there is an increased likelihood that the expired lease will be assigned to another client.

Consider the following configuration:

```
"Dhcp4": {
    ...

    "expired-leases-processing": {
        "reclaim-timer-wait-time": 3,
        "hold-reclaimed-time": 1800,
        "flush-reclaimed-timer-wait-time": 5
    },

    ...
}
```

The `hold-reclaim-time` specifies how many seconds after an expiration a reclaimed lease should be held in the database for re-assignment to the same client. In the example given above, reclaimed leases will be held for 30 minutes (1800s) after their expiration. During this time, the server will likely be able to re-assign the same lease to the returning client, unless another client specifically requests this lease and the server assigns it.

The server must periodically remove reclaimed leases for which the time indicated by `hold-reclaim-time` has elapsed. The `flush-reclaimed-timer-wait-time` parameter controls how often the server removes such leases. In the example provided above, the server will initiate removal of such leases five seconds after the previous removal attempt was completed. Setting this value to 0 disables lease affinity, meaning leases will be removed from the lease database when they are reclaimed. If lease affinity is enabled, it is recommended that `hold-reclaim-time` be set to a value significantly higher than the `reclaim-timer-wait-time`, as timely removal of expired-reclaimed leases is less critical than the removal process, which may impact server responsiveness.

There is no guarantee that lease affinity will work every time; if a server is running out of addresses, it will reassign expired addresses to new clients. Also, clients can request specific addresses and the server will try to honor such requests if possible. Administrators who want to ensure a client keeps its address, even after periods of inactivity, should consider using host reservations or leases with very long lifetimes.

11.5 Reclaiming Expired Leases via Command

The `leases-reclaim` command can be used to trigger lease reclamation at any time. Please consult the *The leases-reclaim Command* section for details about using this command.

CHAPTER

TWELVE

CONGESTION HANDLING

12.1 What is Congestion?

Congestion occurs when servers are subjected to client queries faster than those queries can be processed. As a result, the servers begin accumulating a backlog of pending queries. The longer the high rate of traffic continues, the farther behind the servers fall. Depending on the client implementations, those that fail to get leases either give up or simply continue to retry forever. In the former case, the server may eventually recover, but the latter case is a vicious cycle from which the server is unable to escape.

In a well-planned deployment, the number and capacity of servers is matched to the maximum client loads expected. As long as capacity is matched to load, congestion does not occur. If the load is routinely too heavy, then the deployment needs to be re-evaluated. Congestion typically occurs when there is a network event that causes overly large numbers of clients to simultaneously need leases, such as recovery after a network outage.

The goal of congestion handling is to help servers mitigate the peak in traffic by fulfilling as many of the most relevant requests as possible until the congestion subsides.

Prior to Kea 1.5, kea-dhcp4 and kea-dhcp6 read inbound packets directly from the interface sockets in the main application thread, which meant that packets waiting to be processed were held in socket buffers themselves. Once these buffers filled, any new packets were discarded. Under swamped conditions, the servers ended up processing client packets that were no longer relevant, or worse, were redundant. In other words, the packets waiting in the FIFO socket buffers became increasingly stale.

12.2 Configuring Congestion Handling

Kea 1.5 introduced the Congestion Handling feature. Congestion handling offers the ability to configure the server to use a separate thread to read packets from the interface socket buffers. As the thread reads packets from the buffers, they are added to an internal packet queue, and the server's main application thread processes packets from this queue rather than from the socket buffers. By structuring it this way, a configurable layer has been introduced which can make decisions on which packets to process, how to store them, and the order in which they are processed by the server.

The default packet queue implementation for both kea-dhcp4 and kea-dhcp6 is a simple ring buffer. Once it reaches capacity, new packets get added to the back of the queue by discarding packets from the front of the queue. Rather than always discarding the newest packets, Kea now always discards the oldest packets. The capacity of the buffer, i.e the maximum number of packets the buffer can contain, is configurable. A reasonable starting point would be to match the capacity to the number of leases per second a specific installation of Kea can handle. Please note that this figure varies widely depending on the specifics of an individual deployment.

As there is no one algorithm that will best handle the dynamics of all sites, and because over time new approaches will evolve, the packet queue is implemented as a plug-in, which can replaced by a custom queue implementation via a hook library. This should make it straightforward for interested parties to experiment with their own solutions.

(Developers can refer to isc::dhcp::PacketQueue and isc::dhcp::PacketQueueMgr, described in the Kea Developer's Guide.)

Packet queue behavior is configured in both kea-dhcp4 and kea-dhcp6 servers through an optional, top-level, configuration element, `dhcp-queue-control`. Omitting this element disables packet queueing:

```
"dhcp-queue-control": {
    "enable-queue": true|false,
    "queue-type": "queue type",
    "capacity" : n
}
```

where:

- `enable-queue` true|false - enables or disables packet queueing. When true, the server processes packets from the packet queue, which is filled by a separate thread. When false, the server processes packets directly from the socket buffers in the main thread. It is disabled by default.
- `queue-type` - name of the queue implementation to use. This value exists so that custom implementations can be registered (via a hook library) and then selected. There is a default packet queue implementation that is pre-registered during server start up: "kea-ring4" for kea-dhcp4 and "kea-ring6" for kea-dhcp6.
- `capacity` = n [packets] - this is the maximum number of packets the queue can hold before packets are discarded. The optimal value for this is extremely site-dependent. The default value is 64 for both kea-ring4 and kea-ring6.

The following example enables the default packet queue for kea-dhcp4, with a queue capacity of 250 packets:

```
"Dhcp4":
{
    ...
   "dhcp-queue-control": {
       "enable-queue": true,
       "queue-type": "kea-ring4",
       "capacity" : 250
    },
    ...
}
```

The following example enables the default packet queue for kea-dhcp6, with a queue capacity of 300 packets:

```
"Dhcp6":
{
    ...
   "dhcp-queue-control": {
       "enable-queue": true,
       "queue-type": "kea-ring6",
       "capacity" : 300
    },
    ...
}
```

CHAPTER

THIRTEEN

THE DHCP-DDNS SERVER

13.1 Overview

The DHCP-DDNS Server (kea-dhcp-ddns, known informally as D2) conducts the client side of the Dynamic DNS protocol (DDNS, defined in RFC 2136) on behalf of the DHCPv4 and DHCPv6 servers (kea-dhcp4 and kea-dhcp6 respectively). The DHCP servers construct DDNS update requests, known as Name Change Requests (NCRs), based on DHCP lease change events and then post them to D2. D2 attempts to match each request to the appropriate DNS server(s) and carries out the necessary conversation with those servers to update the DNS data.

13.1.1 DNS Server Selection

In order to match a request to the appropriate DNS servers, D2 must have a catalog of servers from which to select. In fact, D2 has two such catalogs, one for forward DNS and one for reverse DNS; these catalogs are referred to as DDNS Domain Lists. Each list consists of one or more named DDNS Domains. Further, each DDNS Domain has a list of one or more DNS servers that publish the DNS data for that domain.

When conducting forward domain matching, D2 compares the fully qualified domain name (FQDN) in the request against the name of each Forward DDNS Domain in its catalog. The domain whose name matches the longest portion of the FQDN is considered the best match. For example, if the FQDN is "myhost.sample.example.com.", and there are two forward domains in the catalog, "sample.example.com." and "example.com.", the former is regarded as the best match. In some cases, it may not be possible to find a suitable match. Given the same two forward domains there would be no match for the FQDN, "bogus.net", so the request would be rejected. Finally, if there are no Forward DDNS Domains defined, D2 simply disregards the forward update portion of requests.

When conducting reverse domain matching, D2 constructs a reverse FQDN from the lease address in the request and compares that against the name of each Reverse DDNS Domain. Again, the domain whose name matches the longest portion of the FQDN is considered the best match. For instance, if the lease address is "172.16.1.40" and there are two reverse domains in the catalog, "1.16.172.in-addr.arpa." and "16.172.in-addr.arpa", the former is the best match. As with forward matching, it may not find a suitable match. Given the same two domains, there would be no match for the lease address, "192.168.1.50", and the request would be rejected. Finally, if there are no Reverse DDNS Domains defined, D2 simply disregards the reverse update portion of requests.

13.1.2 Conflict Resolution

D2 implements the conflict resolution strategy prescribed by RFC 4703. Conflict resolution is intended to prevent different clients from mapping to the same FQDN at the same time. To make this possible, the RFC requires that forward DNS entries for a given FQDN must be accompanied by a DHCID resource record (RR). This record contains a client identifier that uniquely identifies the client to whom the name belongs. Furthermore, any DNS updater that wishes to update or remove existing forward entries for an FQDN may only do so if their client matches that of the DHCID RR.

In other words, the DHCID RR maps an FQDN to the client to whom it belongs, and thereafter changes to that mapping should only be done by or at the behest of that client.

Currently, conflict detection is always performed.

13.1.3 Dual-Stack Environments

RFC 4703, section 5.2, describes issues that may arise with dual-stack clients. These are clients that wish to have have both IPv4 and IPv6 mappings for the same FQDN. For this to work properly, the clients are required to embed their IPv6 DUID within their IPv4 client identifier option, as described in RFC 4703. In this way, DNS updates for both IPv4 and IPv6 can be managed under the same DHCID RR. Kea does not currently support this feature.

13.2 Starting and Stopping the DHCP-DDNS Server

`kea-dhcp-ddns` is the Kea DHCP-DDNS server and, due to the nature of DDNS, it runs alongside either the DHCPv4 or DHCPv6 component (or both). Like other parts of Kea, it is a separate binary that can be run on its own or through `keactrl` (see *Managing Kea with keactrl*). In normal operation, controlling `kea-dhcp-ddns` with `keactrl` is recommended; however, it is also possible to run the DHCP-DDNS server directly. It accepts the following command-line switches:

- `-c file` - specifies the configuration file. This is the only mandatory switch.
- `-d` - specifies whether the server logging should be switched to debug/verbose mode. In verbose mode, the logging severity and debuglevel specified in the configuration file are ignored and "debug" severity and the maximum debuglevel (99) are assumed. The flag is convenient for temporarily switching the server into maximum verbosity, e.g. when debugging.
- `-v` - displays the Kea version and exits.
- `-W` - displays the Kea configuration report and exits. The report is a copy of the `config.report` file produced by `./configure`; it is embedded in the executable binary.
- `-t file` - specifies the configuration file to be tested. Kea-dhcp-ddns will attempt to load it and will conduct sanity checks. Note that certain checks are possible only while running the actual server. The actual status is reported with an exit code (0 = configuration looks ok, 1 = error encountered). Kea prints out log messages to standard output and errors to standard error when testing the configuration.

The `config.report` may also be accessed more directly, via the following command. The binary `path` may be found in the install directory or in the `.libs` subdirectory in the source tree. For example: `kea/src/bin/d2/.libs/kea-dhcp-ddns`.

```
strings path/kea-dhcp-ddns | sed -n 's/;;;; //p'
```

Upon startup, the module will load its configuration and begin listening for NCRs based on that configuration.

During startup, the server will attempt to create a PID file of the form: [**runstatedir**]/[**conf name**].kea-dhcp-ddns.pid where:

- `runstatedir` - is the value as passed into the build configure script; it defaults to "/usr/local/var/run". Note that this value may be overridden at runtime by setting the environment variable KEA_PIDFILE_DIR. This is intended primarily for testing purposes.
- `conf name` - is the configuration file name used to start the server, minus all preceding paths and the file extension. For example, given a pathname of "/usr/local/etc/kea/myconf.txt", the portion used would be "myconf".

If the file already exists and contains the PID of a live process, the server will issue a DHCP_DDNS_ALREADY_RUNNING log message and exit. It is possible, though unlikely, that the file is a remnant of a system crash and the process to which the PID belongs is unrelated to Kea. In such a case it is necessary to manually delete the PID file.

13.3 Configuring the DHCP-DDNS Server

Before starting the `kea-dhcp-ddns` module for the first time, a configuration file must be created. The following default configuration is a template that can be customized to individual requirements.

```
"DhcpDdns": {
    "ip-address": "127.0.0.1",
    "port": 53001,
    "dns-server-timeout": 100,
    "ncr-protocol": "UDP",
    "ncr-format": "JSON",
    "tsig-keys": [ ],
    "forward-ddns": {
        "ddns-domains": [ ]
    },
    "reverse-ddns": {
        "ddns-domains": [ ]
    }
}
```

The configuration can be divided into the following sections, each of which is described below:

- *Global Server Parameters* - define values which control connectivity and global server behavior.
- *Control Socket* - defines the Control Socket type and name.
- *TSIG Key Info* - defines the TSIG keys used for secure traffic with DNS servers.
- *Forward DDNS* - defines the catalog of Forward DDNS Domains.
- *Reverse DDNS* - defines the catalog of Forward DDNS Domains.

13.3.1 Global Server Parameters

- `ip-address` - the IP address on which D2 listens for requests. The default is the local loopback interface at address 127.0.0.1. Either an IPv4 or IPv6 address may be specified.
- `port` - the port on which D2 listens for requests. The default value is 53001.
- `dns-server-timeout` - the maximum amount of time, in milliseconds, that D2 will wait for a response from a DNS server to a single DNS update message.
- `ncr-protocol` - the socket protocol to use when sending requests to D2. Currently only UDP is supported.
- `ncr-format` - the packet format to use when sending requests to D2. Currently only JSON format is supported.

D2 must listen for change requests on a known address and port. By default it listens at 127.0.0.1 on port 53001. The following example illustrates how to change D2's global parameters so it will listen at 192.168.1.10 port 900:

```
"DhcpDdns": {
    "ip-address": "192.168.1.10",
    "port": 900,
    ...
    }
}
```

Warning: It is possible for a malicious attacker to send bogus NameChangeRequests to the DHCP-DDNS server. Addresses other than the IPv4 or IPv6 loopback addresses (127.0.0.1 or ::1) should only be used for testing purposes; note that local users may still communicate with the DHCP-DDNS server.

Note: If the ip-address and port are changed, the corresponding values in the DHCP servers' "dhcp-ddns" configuration section must be changed.

13.3.2 Management API for the D2 Server

The management API allows the issuing of specific management commands, such as configuration retrieval or shutdown. For more details, see *Management API*. Currently, the only supported communication channel type is UNIX stream socket. By default there are no sockets open; to instruct Kea to open a socket, the following entry in the configuration file can be used:

```
"DhcpDdns": {
    "control-socket": {
        "socket-type": "unix",
        "socket-name": "/path/to/the/unix/socket"
    },
    ...
}
```

The length of the path specified by the `socket-name` parameter is restricted by the maximum length for the UNIX socket name on the operating system, i.e. the size of the `sun_path` field in the `sockaddr_un` structure, decreased by 1. This value varies on different operating systems between 91 and 107 characters. Typical values are 107 on Linux and 103 on FreeBSD.

Communication over the control channel is conducted using JSON structures. See the Control Channel section in the Kea Developer's Guide for more details.

The D2 server supports the following operational commands:

- build-report
- config-get
- config-reload
- config-set
- config-test
- config-write
- list-commands
- shutdown
- status-get
- version-get

The `shutdown` command supports extra `type` argument which controls the way the D2 server cleans up on exit. The supported shutdown types are:

- `normal` - Stops the queue manager and finishes all current transactions before exiting. This is the default.
- `drain_first` - Stops the queue manager but continues processing requests from the queue until it is empty.
- `now` - Exits immediately.

An example command may look like this:

```
{
    "command": "shutdown"
    "arguments": {
        "exit-value": 3,
        "type": "drain_first"
    }
}
```

13.3.3 TSIG Key List

A DDNS protocol exchange can be conducted with or without TSIG (defined in RFC 2845). This configuration section allows the administrator to define the set of TSIG keys that may be used in such exchanges.

To use TSIG when updating entries in a DNS domain, a key must be defined in the TSIG Key list and referenced by name in that domain's configuration entry. When D2 matches a change request to a domain, it checks whether the domain has a TSIG key associated with it. If so, D2 uses that key to sign DNS update messages sent to and verify responses received from the domain's DNS server(s). For each TSIG key required by the DNS servers that D2 will be working with, there must be a corresponding TSIG key in the TSIG Key list.

As one might gather from the name, the tsig-key section of the D2 configuration lists the TSIG keys. Each entry describes a TSIG key used by one or more DNS servers to authenticate requests and sign responses. Every entry in the list has three parameters:

- `name` - is a unique text label used to identify this key within the list. This value is used to specify which key (if any) should be used when updating a specific domain. As long as the name is unique its content is arbitrary, although for clarity and ease of maintenance it is recommended that it match the name used on the DNS server(s). This field cannot be blank.
- `algorithm` - specifies which hashing algorithm should be used with this key. This value must specify the same algorithm used for the key on the DNS server(s). The supported algorithms are listed below:
    - HMAC-MD5
    - HMAC-SHA1
    - HMAC-SHA224
    - HMAC-SHA256
    - HMAC-SHA384
    - HMAC-SHA512

  This value is not case-sensitive.
- `digest-bits` - is used to specify the minimum truncated length in bits. The default value 0 means truncation is forbidden; non-zero values must be an integral number of octets, and be greater than both 80 and half of the full length. (Note that in BIND 9 this parameter is appended after a dash to the algorithm name.)
- `secret` - is used to specify the shared secret key code for this key. This value is case-sensitive and must exactly match the value specified on the DNS server(s). It is a base64-encoded text value.

As an example, suppose that a domain D2 will be updating is maintained by a BIND 9 DNS server, which requires dynamic updates to be secured with TSIG. Suppose further that the entry for the TSIG key in BIND 9's named.conf file looks like this:

```
:
key "key.four.example.com." {
    algorithm hmac-sha224;
    secret "bZEG7Ow8OgAUPfLWV3aAUQ==";
};
:
```

By default, the TSIG Key list is empty:

```
"DhcpDdns": {
   "tsig-keys": [ ],
   ...
}
```

We must extend the list with a new key:

```
"DhcpDdns": {
    "tsig-keys": [
        {
            "name": "key.four.example.com.",
            "algorithm": "HMAC-SHA224",
            "secret": "bZEG7Ow8OgAUPfLWV3aAUQ=="
        }
    ],
    ...
}
```

These steps would be repeated for each TSIG key needed. Note that the same TSIG key can be used with more than one domain.

13.3.4 Forward DDNS

The Forward DDNS section is used to configure D2's forward update behavior. Currently it contains a single parameter, the catalog of Forward DDNS Domains, which is a list of structures.

```
"DhcpDdns": {
    "forward-ddns": {
        "ddns-domains": [ ]
    },
    ...
}
```

By default, this list is empty, which will cause the server to ignore the forward update portions of requests.

Adding Forward DDNS Domains

A Forward DDNS Domain maps a forward DNS zone to a set of DNS servers which maintain the forward DNS data (i.e. name-to-address mapping) for that zone. Each zone served needs one Forward DDNS Domain. It may very well be that some or all of the zones are maintained by the same servers, but one DDNS Domain is still needed for each zone. Remember that matching a request to the appropriate server(s) is done by zone and a DDNS Domain only defines a single zone.

This section describes how to add Forward DDNS Domains; repeat these steps for each Forward DDNS Domain desired. Each Forward DDNS Domain has the following parameters:

- `name` - the fully qualified domain name (or zone) that this DDNS Domain can update. This value is compared against the request FQDN during forward matching. It must be unique within the catalog.
- `key-name` - if TSIG is used with this domain's servers, this value should be the name of the key from the TSIG Key list. If the value is blank (the default), TSIG will not be used in DDNS conversations with this domain's servers.
- `dns-servers` - a list of one or more DNS servers which can conduct the server side of the DDNS protocol for this domain. The servers are used in a first-to-last preference; in other words, when D2 begins to process a request for this domain, it will pick the first server in this list and attempt to communicate with it. If that attempt fails, D2 will move to next one in the list and so on until either it is successful or the list is exhausted.

To create a new Forward DDNS Domain, add a new domain element and set its parameters:

```
"DhcpDdns": {
    "forward-ddns": {
        "ddns-domains": [
            {
                "name": "other.example.com.",
                "key-name": "",
                "dns-servers": [
                ]
            }
        ]
    }
}
```

It is possible to add a domain without any servers; however, if that domain matches a request, the request will fail. To make the domain useful, at least one DNS server must be added to it.

Adding Forward DNS Servers

This section describes how to add DNS servers to a Forward DDNS Domain. Repeat these instructions as needed for all the servers in each domain.

Forward DNS Server entries represent actual DNS servers which support the server side of the DDNS protocol. Each Forward DNS Server has the following parameters:

- `hostname` - the resolvable host name of the DNS server; this parameter is not yet implemented.
- `ip-address` - the IP address at which the server listens for DDNS requests. This may be either an IPv4 or an IPv6 address.
- `port` - the port on which the server listens for DDNS requests. It defaults to the standard DNS service port of 53.

To create a new Forward DNS Server, a new server element must be added to the domain and its parameters filled in. If, for example, the service is running at "172.88.99.10", set the Forward DNS Server as follows:

```
"DhcpDdns": {
    "forward-ddns": {
        "ddns-domains": [
            {
                "name": "other.example.com.",
                "key-name": "",
                "dns-servers": [
                    {
                        "hostname": "",
                        "ip-address": "172.88.99.10",
                        "port": 53
                    }
                ]
            }
        ]
    }
}
```

Note: Since "hostname" is not yet supported, the parameter "ip-address" must be set to the address of the DNS server.

13.3.5 Reverse DDNS

The Reverse DDNS section is used to configure D2's reverse update behavior, and the concepts are the same as for the forward DDNS section. Currently it contains a single parameter, the catalog of Reverse DDNS Domains, which is a list of structures.

```
"DhcpDdns": {
    "reverse-ddns": {
        "ddns-domains": [ ]
    }
    ...
}
```

By default, this list is empty, which will cause the server to ignore the reverse update portions of requests.

Adding Reverse DDNS Domains

A Reverse DDNS Domain maps a reverse DNS zone to a set of DNS servers which maintain the reverse DNS data (address-to-name mapping) for that zone. Each zone served needs one Reverse DDNS Domain. It may very well be that some or all of the zones are maintained by the same servers, but one DDNS Domain entry is still needed for each zone. Remember that matching a request to the appropriate server(s) is done by zone and a DDNS Domain only defines a single zone.

This section describes how to add Reverse DDNS Domains; repeat these steps for each Reverse DDNS Domain desired. Each Reverse DDNS Domain has the following parameters:

- `name` - the fully qualified reverse zone that this DDNS domain can update. This is the value used during reverse matching, which will compare it with a reversed version of the request's lease address. The zone name should follow the appropriate standards; for example, to support the IPv4 subnet 172.16.1, the name should be "1.16.172.in-addr.arpa.". Similarly, to support an IPv6 subnet of 2001:db8:1, the name should be "1.0.0.0.8.B.D.0.1.0.0.2.ip6.arpa." Whatever the name, it must be unique within the catalog.
- `key-name` - if TSIG is used with this domain's servers, this value should be the name of the key from the TSIG Key List. If the value is blank (the default), TSIG will not be used in DDNS conversations with this domain's servers. Currently this value is not used as TSIG has not been implemented.
- `dns-servers` - a list of one or more DNS servers which can conduct the server side of the DDNS protocol for this domain. Currently, the servers are used in a first-to-last preference; in other words, when D2 begins to process a request for this domain, it will pick the first server in this list and attempt to communicate with it.

If that attempt fails, D2 will move to the next one in the list and so on until either it is successful or the list is exhausted.

To create a new Reverse DDNS Domain, a new domain element must be added and its parameters set. For example, to support subnet 2001:db8:1::, the following configuration could be used:

```
"DhcpDdns": {
    "reverse-ddns": {
        "ddns-domains": [
            {
                "name": "1.0.0.0.8.B.D.0.1.0.0.2.ip6.arpa.",
                "key-name": "",
                "dns-servers": [
                ]
            }
        ]
    }
}
```

It is possible to add a domain without any servers; however, if that domain matches a request, the request will fail. To make the domain useful, at least one DNS server must be added to it.

Adding Reverse DNS Servers

This section describes how to add DNS servers to a Reverse DDNS Domain. Repeat these instructions as needed for all the servers in each domain.

Reverse DNS Server entries represent actual DNS servers which support the server side of the DDNS protocol. Each Reverse DNS Server has the following parameters:

- `hostname` - the resolvable host name of the DNS server; this value is currently ignored.
- `ip-address` - the IP address at which the server listens for DDNS requests.
- `port` - the port on which the server listens for DDNS requests. It defaults to the standard DNS service port of 53.

To create a new reverse DNS Server, a new server element must be added to the domain and its parameters filled in. If, for example, the service is running at “172.88.99.10”, then set it as follows:

```
"DhcpDdns": {
    "reverse-ddns": {
        "ddns-domains": [
            {
                "name": "1.0.0.0.8.B.D.0.1.0.0.2.ip6.arpa.",
                "key-name": "",
                "dns-servers": [
                    {
                        "hostname": "",
                        "ip-address": "172.88.99.10",
                        "port": 53
                    }
                ]
            }
        ]
    }
}
```

Note: Since "hostname" is not yet supported, the parameter "ip-address" must be set to the address of the DNS server.

13.3.6 User Contexts in DDNS

Note: User contexts were designed for hook libraries, which are not yet supported for DHCP-DDNS server configuration.

See *Comments and User Context* for additional background regarding the user context idea.

User contexts can be specified on global scope, DDNS domain, DNS server, TSIG key, and loggers. One other useful usage is the ability to store comments or descriptions; the parser translates a "comment" entry into a user context with the entry, which allows a comment to be attached inside the configuration itself.

13.3.7 Example DHCP-DDNS Server Configuration

This section provides a sample DHCP-DDNS server configuration, based on a small example network. Let's suppose our example network has three domains, each with their own subnet.

Table 13.1: Our Example Network

Domain	Subnet	Forward DNS Servers	Reverse DNS Servers
four.example.com	192.0.2.0/24	172.16.1.5, 172.16.2.5	172.16.1.5, 172.16.2.5
six.example.com	2001:db8:1::/64	3001:1::50	3001:1::51
example.com	192.0.0.0/16	172.16.2.5	172.16.2.5

We need to construct three Forward DDNS Domains:

Table 13.2: Forward DDNS Domains Needed

#	DDNS Domain Name	DNS Servers
1.	four.example.com.	172.16.1.5, 172.16.2.5
2.	six.example.com.	3001:1::50
3.	example.com.	172.16.2.5

As discussed earlier, FQDN-to-domain matching is based on the longest match. The FQDN, "myhost.four.example.com.", will match the first domain ("four.example.com") while "admin.example.com." will match the third domain ("example.com"). The FQDN, "other.example.net.", will fail to match any domain and is rejected.

The following example configuration specifies the Forward DDNS Domains.

```
"DhcpDdns": {
    "comment": "example configuration: forward part",
    "forward-ddns": {
        "ddns-domains": [
            {
                "name": "four.example.com.",
                "key-name": "",
                "dns-servers": [
                    { "ip-address": "172.16.1.5" },
                    { "ip-address": "172.16.2.5" }
                ]
            },
            {
                "name": "six.example.com.",
                "key-name": "",
                "dns-servers": [
                    { "ip-address": "2001:db8::1" }
                ]
            },
            {
                "name": "example.com.",
                "key-name": "",
                "dns-servers": [
                    { "ip-address": "172.16.2.5" }
                ],
                "user-context": { "backup": false }
            },

        ]
    }
}
```

Similarly, we need to construct the three Reverse DDNS Domains:

Table 13.3: Reverse DDNS Domains Needed

#	DDNS Domain Name	DNS Servers
1.	2.0.192.in-addr.arpa.	172.16.1.5, 172.16.2.5
2.	1.0.0.0.8.d.b.0.1.0.0.2.ip6.arpa.	3001:1::50
3.	0.182.in-addr.arpa.	172.16.2.5

An address of "192.0.2.150" will match the first domain, "2001:db8:1::10" will match the second domain, and "192.0.50.77" the third domain.

These Reverse DDNS Domains are specified as follows:

```
"DhcpDdns": {
    "comment": "example configuration: reverse part",
    "reverse-ddns": {
        "ddns-domains": [
            {
                "name": "2.0.192.in-addr.arpa.",
                "key-name": "",
                "dns-servers": [
                    { "ip-address": "172.16.1.5" },
                    { "ip-address": "172.16.2.5" }
                ]
            }
            {
                "name": "1.0.0.0.8.B.D.0.1.0.0.2.ip6.arpa.",
                "key-name": "",
                "dns-servers": [
                    { "ip-address": "2001:db8::1" }
                ]
            }
            {
                "name": "0.192.in-addr.arpa.",
                "key-name": "",
                "dns-servers": [
                    { "ip-address": "172.16.2.5" }
                ]
            }
        ]
    }
}
```

13.4 DHCP-DDNS Server Limitations

The following are the current limitations of the DHCP-DDNS Server.

- Requests received from the DHCP servers are placed in a queue until they are processed. Currently, all queued requests are lost if the server shuts down.

13.5 Supported Standards

The following RFCs are supported by the DHCP-DDNS server:

- *Secret Key Transaction Authentication for DNS (TSIG)*, RFC 2845: All DNS Update packets sent and received by DHCP-DDNS server can be protected by TSIG signatures.
- *Dynamic Updates in the Domain Name System (DNS UPDATE)*, RFC 2136: The whole DNS Update mechanism is supported.
- *Resolution of Fully Qualified Domain Name (FQDN) Conflicts among Dynamic Host Configuration Protocol (DHCP) Clients*, RFC 4703: The DHCP-DDNS takes care of the conflict resolution. This capability is used by DHCPv4 and DHCPv6 servers.
- *A DNS Resource Record (RR) for Encoding Dynamic Host Configuration Protocol (DHCP) Information (DHCID RR)*, RFC 4701: The DHCP-DDNS server uses the DHCID records.

CHAPTER

FOURTEEN

THE LFC PROCESS

14.1 Overview

`kea-lfc` is a service process that removes redundant information from the files used to provide persistent storage for the Memfile database backend. This service is written to run as a standalone process.

While `kea-lfc` can be started externally, there is usually no need to do s. `kea-lfc` is run on a periodic basis by the Kea DHCP servers.

The process operates on a set of files, using them to receive input and output of the lease entries and to indicate what stage the process is in, in the event of an interruption. Currently the caller must supply names for all of the files.

14.2 Command-Line Options

`kea-lfc` is run as follows:

```
kea-lfc [-4 | -6] -c config-file -p pid-file -x previous-file -i copy-file -o output-
↪file -f finish-file
```

The argument `-4` or `-6` selects the protocol version of the lease files.

The `-c` argument specifies the configuration file. This is required, but is not currently used by the process.

The `-p` argument specifies the PID file. When the `kea-lfc` process starts, it attempts to determine whether another instance of the process is already running by examining the PID file. If one is already running, the new process is terminated; if one is not running, Kea writes its PID into the PID file.

The other filenames specify where the `kea-lfc` process should look for input, write its output, and perform its bookkeeping:

- `previous` — when `kea-lfc` starts, this is the result of any previous run of `kea-lfc`. When `kea-lfc` finishes, it is the result of this run. If `kea-lfc` is interrupted before completing, this file may not exist.
- `input` — before the DHCP server invokes `kea-lfc`, it moves the current lease file here and then calls `kea-lfc` with this file.
- `output` — This is the temporary file where `kea-lfc` writes the leases. Once the file has finished writing, it will be moved to the finish file (see below).
- `finish` — This is another temporary file `kea-lfc` uses for bookkeeping. When `kea-lfc` completes writing the output file, it moves the output to this file name. After `kea-lfc` finishes deleting the other files (previous and input), it moves this file to the previous lease file. By moving the files in this fashion, the `kea-lfc` and the DHCP server processes can determine the correct file to use even if one of the processes were interrupted before completing its task.

There are several additional arguments, mostly for debugging purposes. `-d` sets the logging level to debug. `-v` and `-V` print out version stamps, with `-V` providing a longer form. `-h` prints out the usage string.

CHAPTER

FIFTEEN

CLIENT CLASSIFICATION

15.1 Client Classification Overview

In certain cases it is useful to differentiate among different types of clients and treat them accordingly. Common reasons include:

- The clients represent different pieces of topology, e.g. a cable modem is not the same as the clients behind that modem.
- The clients have different behavior, e.g. a smartphone behaves differently from a laptop.
- The clients require different values for some options, e.g. a docsis3.0 cable modem requires different settings from a docsis2.0 cable modem.

To make management easier, different clients can be grouped into a client class to receive common options.

An incoming packet can be associated with a client class in several ways:

- Implicitly, using a vendor class option or another built-in condition.
- Using an expression which evaluates to true.
- Using static host reservations, a shared network, a subnet, etc.
- Using a hook.

It is envisaged that client classification will be used to change the behavior of almost any part of the DHCP message processing. There are currently five mechanisms that take advantage of client classification: subnet selection, pool selection, definition of DHCPv4 private (codes 224-254) and code 43 options, assignment of different options, and, for DHCPv4 cable modems, the setting of specific options for use with the TFTP server address and the boot file field.

The classification process is conducted in several steps:

1. The ALL class is associated with the incoming packet.
2. Vendor class options are processed.
3. Classes with matching expressions and not marked for later evaluation ("on request" or depending on the KNOWN/UNKNOWN built-in classes) are processed in the order they are defined in the configuration; the boolean expression is evaluated and, if it returns true ("match"), the incoming packet is associated with the class.
4. If a private or code 43 DHCPv4 option is received, it is decoded following its client class or global (or, for option 43, last resort) definition.
5. When the incoming packet belongs the special class, *DROP*, it is dropped and an informational message is logged with the packet information.

6. A subnet is chosen, possibly based on the class information when some subnets are reserved. More precisely: when choosing a subnet, the server iterates over all of the subnets that are feasible given the information found in the packet (client address, relay address, etc.). It uses the first subnet it finds that either doesn't have a class associated with it, or has a class which matches one of the packet's classes.
7. The server looks for host reservations. If an identifier from the incoming packet matches a host reservation in the subnet or shared network, the packet is associated with the KNOWN class and all classes of the host reservation. If a reservation is not found, the packet is assigned to the UNKNOWN class.
8. Classes with matching expressions - directly, or indirectly using the KNOWN/UNKNOWN built-in classes and not marked for later evaluation ("on request") - are processed in the order they are defined in the configuration; the boolean expression is evaluated and, if it returns true ("match"), the incoming packet is associated with the class. After a subnet is selected, the server determines whether there is a reservation for a given client. Therefore, it is not possible to use KNOWN/UNKNOWN classes to select a shared network or a subnet, nor to make DROP class dependent of KNOWN/UNKNOWN classes.
9. If needed, addresses and prefixes from pools are assigned, possibly based on the class information when some pools are reserved for class members.
10. Classes marked as "required" are evaluated in the order in which they are listed: first the shared network, then the subnet, and finally the pools that assigned resources belong to.
11. Options are assigned, again possibly based on the class information in the order that classes were associated with the incoming packet. For DHCPv4 private and code 43 options, this includes class local option definitions.

Note: Client classes in Kea follow the order in which they are specified in the configuration (vs. alphabetical order). Required classes follow the order in which they are required.

When determining which options to include in the response, the server examines the union of options from all of the assigned classes. If two or more classes include the same option, the value from the first class examined is used; classes are examined in the order they were associated, so ALL is always the first class and matching required classes are last.

As an example, imagine that an incoming packet matches two classes. Class "foo" defines values for an NTP server (option 42 in DHCPv4) and an SMTP server (option 69 in DHCPv4), while class "bar" defines values for an NTP server and a POP3 server (option 70 in DHCPv4). The server examines the three options - NTP, SMTP, and POP3 - and returns any that the client requested. As the NTP server was defined twice, the server chooses only one of the values for the reply; the class from which the value is obtained determined as explained in the previous paragraph.

Note: Care should be taken with client classification, as it is easy for clients that do not meet any class criteria to be denied service altogether.

15.2 Built-in Client Classes

Some classes are built-in, so they do not need to be defined. The main example uses Vendor Class information: the server checks whether an incoming DHCPv4 packet includes the vendor class identifier option (60) or an incoming DHCPv6 packet includes the vendor class option (16). If it does, the content of that option is prepended with "VENDOR_CLASS_" and the result is interpreted as a class. For example, modern cable modems send this option with value "docsis3.0", so the packet belongs to class "VENDOR_CLASS_docsis3.0".

The "HA_" prefix is used by the High Availability hooks library to designate certain servers to process DHCP packets as a result of load balancing. The class name is constructed by prepending the "HA_" prefix to the name of the server which should process the DHCP packet. This server uses an appropriate pool or subnet to allocate IP addresses (and/or prefixes), based on the assigned client classes. The details can be found in *ha: High Availability*.

The BOOTP class is used by the BOOTP hook library to classify and respond to inbound BOOTP queries.

Other examples are the ALL class, which all incoming packets belong to, and the KNOWN class, assigned when host reservations exist for a particular client. By convention, built-in classes' names begin with all capital letters.

Currently recognized built-in class names are ALL, KNOWN and UNKNOWN, and the prefixes VENDOR_CLASS_, HA_, AFTER_, and EXTERNAL_. Although the AFTER_ prefix is a provision for an as-yet-unwritten hook, the EXTERNAL_ prefix can be freely used; built-in classes are implicitly defined so they never raise warnings if they do not appear in the configuration.

15.3 Using Expressions in Classification

The expression portion of a classification definition contains operators and values. All values are currently strings; operators take a string or strings and return another string. When all the operations have completed, the result should be a value of "true" or "false". The packet belongs to the class (and the class name is added to the list of classes) if the result is "true". Expressions are written in standard format and can be nested.

Expressions are pre-processed during the parsing of the configuration file and converted to an internal representation. This allows certain types of errors to be caught and logged during parsing. Examples of these errors include an incorrect number or type of argument to an operator. The evaluation code also checks for this class of error and generally throws an exception, though this should not occur in a normally functioning system.

Other issues, such as the starting position of a substring being outside of the substring or an option not existing in the packet, result in the operator returning an empty string.

Dependencies between classes are also checked. For instance, forward dependencies are rejected when the configuration is parsed; an expression can only depend on already-defined classes (including built-in classes) which are evaluated in a previous or the same evaluation phase. This does not apply to the KNOWN or UNKNOWN classes.

Table 15.1: List of Classification Values

Name	Example expression	Example value
String literal	'example'	'example'
Hexadecimal string literal	0x5a7d	'Z}'
IP address literal	10.0.0.1	0x0a000001
Integer literal	123	'123'
Binary content of the option	option[123].hex	'(content of the option)'
Option existence	option[123].exists	'true'
Binary content of the sub-option	option[12].option[34].hex	'(content of the sub-option)'
Sub-Option existence	option[12].option[34].exists	'true'
Client class membership	member('foobar')	'true'
Known client	known	member('KNOWN')
Unknown client	unknown	not member('KNOWN')
DHCPv4 relay agent sub-option	relay4[123].hex	'(content of the RAI sub-option)'
DHCPv6 Relay Options	relay6[nest].option[code].hex	(value of the option)
DHCPv6 Relay Peer Address	relay6[nest].peeraddr	2001:DB8::1
DHCPv6 Relay Link Address	relay6[nest].linkaddr	2001:DB8::1
Interface name of packet	pkt.iface	eth0
Source address of packet	pkt.src	10.1.2.3
Destination address of packet	pkt.dst	10.1.2.3
Length of packet	pkt.len	513

Continued on next page

Table 15.1 – continued from previous page

Name	Example expression	Example value
Hardware address in DHCPv4 packet	pkt4.mac	0x010203040506
Hardware length in DHCPv4 packet	pkt4.hlen	6
Hardware type in DHCPv4 packet	pkt4.htype	6
ciaddr field in DHCPv4 packet	pkt4.ciaddr	192.0.2.1
giaddr field in DHCPv4 packet	pkt4.giaddr	192.0.2.1
yiaddr field in DHCPv4 packet	pkt4.yiaddr	192.0.2.1
siaddr field in DHCPv4 packet	pkt4.siaddr	192.0.2.1
Message type in DHCPv4 packet	pkt4.msgtype	1
Transaction ID (xid) in DHCPv4 packet	pkt4.transid	12345
Message type in DHCPv6 packet	pkt6.msgtype	1
Transaction ID in DHCPv6 packet	pkt6.transid	12345
Vendor option existence (any vendor)	vendor[*].exists	true
Vendor option existence (specific vendor)	vendor[4491].exists	true
Enterprise-id from vendor option	vendor.enterprise	4491
Vendor sub-option existence	vendor[4491].option[1].exists	true
Vendor sub-option content	vendor[4491].option[1].hex	docsis3.0
Vendor class option existence (any vendor)	vendor-class[*].exist s	true
Vendor class option existence (specific vendor)	vendor-class[4491].exists	true
Enterprise-id from vendor class option	vendor-class.enterprise	4491
First data chunk from vendor class option	vendor-class[4491].data	docsis3.0
Specific data chunk from vendor class option	vendor-class[4491].data[3]	docsis3.0

Notes:

- Hexadecimal strings are converted into a string as expected. The starting “0X” or “0x” is removed, and if the string is an odd number of characters a “0” is prepended to it.
- IP addresses are converted into strings of length 4 or 16. IPv4, IPv6, and IPv4-embedded IPv6 (e.g. IPv4-mapped IPv6) addresses are supported.
- Integers in an expression are converted to 32-bit unsigned integers and are represented as four-byte strings; for example, 123 is represented as 0x0000007b. All expressions that return numeric values use 32-bit unsigned integers, even if the field in the packet is smaller. In general, it is easier to use decimal notation to represent integers, but it is also possible to use hexadecimal notation. When writing an integer in hexadecimal, care should be taken to make sure the value is represented as 32 bits, e.g. use 0x00000001 instead of 0x1 or 0x01. Also, make sure the value is specified in network order, e.g. 1 is represented as 0x00000001.
- “option[code].hex” extracts the value of the option with the code “code” from the incoming packet. If the packet doesn’t contain the option, it returns an empty string. The string is presented as a byte string of the option payload, without the type code or length fields.
- “option[code].exists” checks whether an option with the code “code” is present in the incoming packet. It can be used with empty options.
- “member(‘foobar’)” checks whether the packet belongs to the client class “foobar”. To avoid dependency loops, the configuration file parser verifies whether client classes were already defined or are built-in, i.e., beginning by “VENDOR_CLASS_”, “AFTER_” (for the to-come “after” hook) and “EXTERNAL_” or equal to “ALL”, “KNOWN”, “UNKNOWN”, etc.

  “known” and “unknown” are shorthand for “member(‘KNOWN’)” and “not member(‘KNOWN’)”. Note that the evaluation of any expression using directly or indirectly the “KNOWN” class is deferred after the host reservation lookup (i.e. when the “KNOWN” or “UNKNOWN” partition is determined).
- “relay4[code].hex” attempts to extract the value of the sub-option “code” from the option inserted as the DHCPv4 Relay Agent Information (82) option. If the packet doesn’t contain a RAI option, or the RAI op-

tion doesn't contain the requested sub-option, the expression returns an empty string. The string is presented as a byte string of the option payload without the type code or length fields. This expression is allowed in DHCPv4 only.

- "relay4" shares the same representation types as "option"; for instance, "relay4[code].exists" is supported.
- "relay6[nest]" allows access to the encapsulations used by any DHCPv6 relays that forwarded the packet. The "nest" level specifies the relay from which to extract the information, with a value of 0 indicating the relay closest to the DHCPv6 server. Negative values allow specifying relays counted from the DHCPv6 client, -1 indicating the relay closest to the client. In general, negative "nest" level is the same as the number of relays + "nest" level. If the requested encapsulation doesn't exist, an empty string "" is returned. This expression is allowed in DHCPv6 only.
- "relay6[nest].option[code]" shares the same representation types as "option"; for instance, "relay6[nest].option[code].exists" is supported.
- Expressions starting with "pkt4" can be used only in DHCPv4. They allow access to DHCPv4 message fields.
- "pkt6" refers to information from the client request. To access any information from an intermediate relay use "relay6". "pkt6.msgtype" and "pkt6.transid" output a 4-byte binary string for the message type or transaction id. For example the message type SOLICIT will be "0x00000001" or simply 1 as in "pkt6.msgtype == 1".
- Vendor option means the Vendor-Identifying Vendor-Specific Information option in DHCPv4 (code 125; see Section 4 of RFC 3925) and Vendor-Specific Information Option in DHCPv6 (code 17, defined in Section 21.17 of RFC 8415). Vendor class option means Vendor-Identifying Vendor Class Option in DHCPv4 (code 124; see Section 3 of RFC 3925) in DHCPv4 and Class Option in DHCPv6 (code 16; see Section 21.16 of RFC 8415). Vendor options may have sub-options that are referenced by their codes. Vendor class options do not have sub-options, but rather data chunks, which are referenced by index value. Index 0 means the first data chunk, index 1 is for the second data chunk (if present), etc.
- In the vendor and vendor-class constructs an asterisk (*) or 0 can be used to specify a wildcard enterprise-id value, i.e. it will match any enterprise-id value.
- Vendor Class Identifier (option 60 in DHCPv4) can be accessed using the option[60] expression.
- RFC 3925 and RFC 8415 allow for multiple instances of vendor options to appear in a single message. The client classification code currently examines the first instance if more than one appear. For the vendor.enterprise and vendor-class.enterprise expressions, the value from the first instance is returned. Please submit a feature request on the Kea GitLab site to request support for multiple instances.

Table 15.2: List of Classification Expressions

Name	Example	Description
Equal	'foo' == 'bar'	Compare the two values and return "true" or "false"
Not	not ('foo' == 'bar')	Logical negation
And	('foo' == 'bar') and ('bar' == 'foo')	Logical and
Or	('foo' == 'bar') or ('bar' == 'foo')	Logical or
Substring	substring('foobar',0,3)	Return the requested substring
Concat	concat('foo','bar')	Return the concatenation of the strings
Ifelse	ifelse('foo' == 'bar','us','them')	Return the branch value according to the condition
Hexstring	hexstring('foo', '-')	Converts the value to a hexadecimal string, e.g. 0a:1b:2c:3e

15.3.1 Logical operators

The Not, And, and Or logical operators are the common operators. Not has the highest precedence and Or the lowest. And and Or are (left) associative. Parentheses around a logical expression can be used to enforce a specific grouping; for instance, in "A and (B or C)" (without parentheses "A and B or C" means "(A and B) or C").

15.3.2 Substring

The substring operator "substring(value, start, length)" accepts both positive and negative values for the starting position and the length. For "start", a value of 0 is the first byte in the string while -1 is the last byte. If the starting point is outside of the original string an empty string is returned. "length" is the number of bytes to extract. A negative number means to count towards the beginning of the string but does not include the byte pointed to by "start". The special value "all" means to return all bytes from start to the end of the string. If the length is longer than the remaining portion of the string, then the entire remaining portion is returned. Some examples may be helpful:

```
substring('foobar', 0, 6) == 'foobar'
substring('foobar', 3, 3) == 'bar'
substring('foobar', 3, all) == 'bar'
substring('foobar', 1, 4) == 'ooba'
substring('foobar', -5, 4) == 'ooba'
substring('foobar', -1, -3) == 'oba'
substring('foobar', 4, -2) == 'ob'
substring('foobar', 10, 2) == ''
```

15.3.3 Concat

The concat function "concat(string1, string2)" returns the concatenation of its two arguments. For instance:

```
concat('foo', 'bar') == 'foobar'
```

15.3.4 Ifelse

The ifelse function "ifelse(cond, iftrue, ifelse)" returns the "iftrue" or "ifelse" branch value following the boolean condition "cond". For instance:

```
ifelse(option[230].exists, option[230].hex, 'none')
```

15.3.5 Hexstring

The hexstring function "hexstring(binary, separator)" returns the binary value as its hexadecimal string representation: pairs of hexadecimal digits separated by the separator, e.g ':', '-', '' (empty separator).

```
hexstring(pkt4.mac, ':')
```

Note: The expression for each class is executed on each packet received. If the expressions are overly complex, the time taken to execute them may impact the performance of the server. Administrators who need complex or time-consuming expressions should consider writing a *hook* to perform the necessary work.

15.4 Configuring Classes

A class contains five items: a name, a test expression, option data, an option definition, and an only-if-required flag. The name must exist and must be unique among all classes. The test expression, option data and definition, and only-if-required flag are optional.

The test expression is a string containing the logical expression used to determine membership in the class. The entire expression is in double quotes.

The option data is a list which defines any options that should be assigned to members of this class.

The option definition is for DHCPv4 option 43 (*DHCPv4 Vendor-Specific Options*) and DHCPv4 private options (*DHCPv4 Private Options*).

Usually the test expression is evaluated before subnet selection, but in some cases it is useful to evaluate it later when the subnet, shared network, or pools are known but output option processing has not yet been done. The only-if-required flag, false by default, allows the evaluation of the test expression only when it is required, i.e. in a require-client-classes list of the selected subnet, shared network, or pool.

The require-client-classes list which is valid for shared-network, subnet, and pool scope specifies the classes which are evaluated in the second pass before output option processing. The list is built in the reversed precedence order of option data, i.e. an option data item in a subnet takes precedence over one in a shared network, but required class in a subnet is added after one in a shared network. The mechanism is related to the only-if-required flag but it is not mandatory that the flag be set to true.

In the following example, the class named "Client_foo" is defined. It is comprised of all clients whose client ids (option 61) start with the string "foo". Members of this class will be given 192.0.2.1 and 192.0.2.2 as their domain name servers.

```
"Dhcp4": {
    "client-classes": [
        {
            "name": "Client_foo",
            "test": "substring(option[61].hex,0,3) == 'foo'",
            "option-data": [
                {
                    "name": "domain-name-servers",
                    "code": 6,
                    "space": "dhcp4",
                    "csv-format": true,
                    "data": "192.0.2.1, 192.0.2.2"
                }
            ]
        },
        ...
    ],
    ...
}
```

This example shows a client class being defined for use by the DHCPv6 server. In it the class named "Client_enterprise" is defined. It is comprised of all clients whose client identifiers start with the given hex string (which would indicate a DUID based on an enterprise id of 0xAABBCCDD). Members of this class will be given an 2001:db8:0::1 and 2001:db8:2::1 as their domain name servers.

```
"Dhcp6": {
    "client-classes": [
        {
            "name": "Client_enterprise",
            "test": "substring(option[1].hex,0,6) == 0x0002AABBCCDD",
            "option-data": [
                {
                    "name": "dns-servers",
                    "code": 23,
                    "space": "dhcp6",
                    "csv-format": true,
                    "data": "2001:db8:0::1, 2001:db8:2::1"
                }
            ]
        },
        ...
    ],
    ...
}
```

15.5 Using Static Host Reservations In Classification

Classes can be statically assigned to the clients using techniques described in *Reserving Client Classes in DHCPv4* and *Reserving Client Classes in DHCPv6*.

15.6 Configuring Subnets With Class Information

In certain cases it is beneficial to restrict access to certain subnets only to clients that belong to a given class, using the “client-class” keyword when defining the subnet.

Let’s assume that the server is connected to a network segment that uses the 192.0.2.0/24 prefix. The administrator of that network has decided that addresses from the range 192.0.2.10 to 192.0.2.20 are going to be managed by the DHCP4 server. Only clients belonging to client class Client_foo are allowed to use this subnet. Such a configuration can be achieved in the following way:

```
"Dhcp4": {
    "client-classes": [
        {
            "name": "Client_foo",
            "test": "substring(option[61].hex,0,3) == 'foo'",
            "option-data": [
                {
                    "name": "domain-name-servers",
                    "code": 6,
                    "space": "dhcp4",
                    "csv-format": true,
                    "data": "192.0.2.1, 192.0.2.2"
                }
            ]
        },
        ...
    ],
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [ { "pool": "192.0.2.10 - 192.0.2.20" } ],
            "client-class": "Client_foo"
        },
        ...
    ],,
    ...
}
```

The following example shows how to restrict access to a DHCPv6 subnet. This configuration will restrict use of the addresses 2001:db8:1::1 to 2001:db8:1::FFFF to members of the "Client_enterprise" class.

```
"Dhcp6": {
    "client-classes": [
        {
            "name": "Client_enterprise",
            "test": "substring(option[1].hex,0,6) == 0x0002AABBCCDD",
            "option-data": [
                {
                    "name": "dns-servers",
                    "code": 23,
                    "space": "dhcp6",
                    "csv-format": true,
                    "data": "2001:db8:0::1, 2001:db8:2::1"
                }
            ]
        },
        ...
    ],
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",
            "pools": [ { "pool": "2001:db8:1::-2001:db8:1::ffff" } ],
            "client-class": "Client_enterprise"
        }
    ],
    ...
}
```

15.7 Configuring Pools With Class Information

Similar to subnets, in certain cases access to certain address or prefix pools must be restricted to only clients that belong to a given class, using the "client-class" when defining the pool.

Let's assume that the server is connected to a network segment that uses the 192.0.2.0/24 prefix. The administrator of that network has decided that addresses from the range 192.0.2.10 to 192.0.2.20 are going to be managed by the DHCP4 server. Only clients belonging to client class Client_foo are allowed to use this pool. Such a configuration can be achieved in the following way:

```
"Dhcp4": {
    "client-classes": [
        {
            "name": "Client_foo",
            "test": "substring(option[61].hex,0,3) == 'foo'",
            "option-data": [
                {
                    "name": "domain-name-servers",
                    "code": 6,
                    "space": "dhcp4",
                    "csv-format": true,
                    "data": "192.0.2.1, 192.0.2.2"
                }
            ]
        },
        ...
    ],
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [
                {
                    "pool": "192.0.2.10 - 192.0.2.20",
                    "client-class": "Client_foo"
                }
            ]
        },
        ...
    ],,

}
```

The following example shows how to restrict access to an address pool. This configuration will restrict use of the addresses 2001:db8:1::1 to 2001:db8:1::FFFF to members of the "Client_enterprise" class.

```
"Dhcp6": {
    "client-classes": [
        {
            "name": "Client_enterprise_",
            "test": "substring(option[1].hex,0,6) == 0x0002AABBCCDD",
            "option-data": [
                {
                    "name": "dns-servers",
                    "code": 23,
                    "space": "dhcp6",
                    "csv-format": true,
                    "data": "2001:db8:0::1, 2001:db8:2::1"
                }
            ]
        },
        ...
    ],
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",

            "pools": [
                {
                    "pool": "2001:db8:1::-2001:db8:1::ffff",
                    "client-class": "Client_foo"
                }
            ]
        },
        ...
    ],
    ...
}
```

15.8 Using Classes

Currently classes can be used for two functions: they can supply options to members of the class, and they can be used to choose a subnet from which an address will be assigned to a class member.

When supplying options, options defined as part of the class definition are considered "class globals." They will override any global options that may be defined and in turn will be overridden by any options defined for an individual subnet.

15.9 Classes and Hooks

Hooks may be used to classify packets. This may be useful if the expression would be complex or time-consuming to write, and could be better or more easily written as code. Once the hook has added the proper class name to the packet, the rest of the classification system will work as expected in choosing a subnet and selecting options. For a description of hooks, see *Hooks Libraries*; for information on configuring classes, see *Configuring Classes* and *Configuring Subnets With Class Information*.

15.10 Debugging Expressions

While constructing classification expressions, administrators may find it useful to enable logging; see *Logging* for a more complete description of the logging facility.

To enable the debug statements in the classification system, the severity must be set to "DEBUG" and the debug level to at least 55. The specific loggers are "kea-dhcp4.eval" and "kea-dhcp6.eval".

To understand the logging statements, it is essential to understand a bit about how expressions are evaluated; for a more complete description, refer to the design document at https://gitlab.isc.org/isc-projects/kea/wikis/designs/Design-documents. In brief, there are two structures used during the evaluation of an expression: a list of tokens which represent the expressions, and a value stack which represents the values being manipulated.

The list of tokens is created when the configuration file is processed, with most expressions and values being converted to a token. The list is organized in reverse Polish notation. During execution, the list will be traversed in order; as each token is executed it will be able to pop values from the top of the stack and eventually push its result on the top of the stack. Imagine the following expression:

```
"test": "substring(option[61].hex,0,3) == 'foo'",
```

This will result in the following tokens:

```
option, number (0), number (3), substring, text ('foo'), equals
```

In this example the first three tokens will simply push values onto the stack. The substring token will then remove those three values and compute a result that it places on the stack. The text option also places a value on the stack and finally the equals token removes the two tokens on the stack and places its result on the stack.

When debug logging is enabled, each time a token is evaluated it will emit a log message indicating the values of any objects that were popped off of the value stack and any objects that were pushed onto the value stack.

The values will be displayed as either text, if the command is known to use text values, or hexadecimal, if the command either uses binary values or can manipulate either text or binary values. For expressions that pop multiple values off the stack, the values will be displayed in the order they were popped. For most expressions this will not matter, but for the concat expression the values are displayed in reverse order from their written order in the expression.

Let us assume that the following test has been entered into the configuration. This example skips most of the configuration to concentrate on the test.

```
"test": "substring(option[61].hex,0,3) == 'foo'",
```

The logging might then resemble this:

```
2016-05-19 13:35:04.163 DEBUG [kea.eval/44478] EVAL_DEBUG_OPTION Pushing option 61␣
 ↪with value 0x666F6F626172
2016-05-19 13:35:04.164 DEBUG [kea.eval/44478] EVAL_DEBUG_STRING Pushing text string
 ↪'0'
2016-05-19 13:35:04.165 DEBUG [kea.eval/44478] EVAL_DEBUG_STRING Pushing text string
 ↪'3'
2016-05-19 13:35:04.166 DEBUG [kea.eval/44478] EVAL_DEBUG_SUBSTRING Popping length 3,␣
 ↪start 0, string 0x666F6F626172 pushing result 0x666F6F
2016-05-19 13:35:04.167 DEBUG [kea.eval/44478] EVAL_DEBUG_STRING Pushing text string
 ↪'foo'
2016-05-19 13:35:04.168 DEBUG [kea.eval/44478] EVAL_DEBUG_EQUAL Popping 0x666F6F and␣
 ↪0x666F6F pushing result 'true'
```

Note: The debug logging may be quite verbose if there are a number of expressions to evaluate; that is intended as an aid in helping create and debug expressions. Administrators should plan to disable debug logging when the expressions are working correctly. Users may also wish to include only one set of expressions at a time in the configuration file while debugging them, to limit the log statements. For example, when adding a new set of expressions, an administrator might find it more convenient to create a configuration file that only includes the new expressions until they are working correctly, and then add the new set to the main configuration file.

CHAPTER

SIXTEEN

HOOKS LIBRARIES

16.1 Introduction

Kea is both flexible and customizable, via the use of "hooks." This feature lets Kea load one or more dynamically linked libraries (known as "hooks libraries") and, at various points in its processing ("hook points"), call functions in them. Those functions perform whatever custom processing is required.

The hooks concept allows the core Kea code to remain reasonably small by moving features to external libraries that some, but not all, users find useful. Those with no need for specific functions can simply choose not to load the libraries.

Hooks libraries are loaded by individual Kea processes, not by Kea as a whole. This means, among other things, that it is possible to associate one set of libraries with the DHCP4 server and a different set with the DHCP6 server.

Another point to note is that it is possible for a process to load multiple libraries. When processing reaches a hook point, Kea calls the hooks library functions attached to it. If multiple libraries have attached a function to a given hook point, Kea calls all of them, in the order in which the libraries are specified in the configuration file. The order may be important; consult the documentation of the libraries for specifics.

When a Kea process unloads a library, it expects the `dlclose` function removes all library symbols and removes the library code from address space on the last reference. This behavior is not required by the POSIX standard and at least the musl library used by default by Alpine Linux implements the `dlclose` function as a no operation. On such systems a library is loaded forever in a process, for instance it is not possible to replace a library binary by another version using configuration change or reload: the process must be stopped and relaunched.

The next section describes how to configure hooks libraries. Users who are interested in writing their own hooks library can find information in the Hooks Developer's Guide section of the Kea Developer's Guide.

Note that some libraries are available under different licenses.

Please also note that some libraries may require additional dependencies and/or compilation switches to be enabled, e.g. the RADIUS library introduced in Kea 1.4 requires the FreeRadius-client library to be present. If –with-free-radius option is not specified, the RADIUS library will not be built.

16.2 Installing Hook Packages

Note: For more details about installing the Kea Premium Hooks package, please read this Knowledgebase article.

Some hook packages are included in the base Kea sources. There is no need to do anything special to compile or install them, as they are covered by the usual building and installation procedures. Please refer to *Installation* for a general overview of the installation process.

ISC provides several additional premium hooks in the form of packages, which follow a similar installation procedure but with several additional steps. For our users' convenience, the premium hooks installation procedure is described in this section.

1. Download the package; detailed instructions are provided separately on how to get it. The package will be a file with a name similar to kea-premium-1.9.0.tar.gz. (The name may vary depending on the package purchased.)

2. Administrators who still have the sources for the corresponding version of the open-source Kea package still on their system from the initial Kea installation should skip this step. Otherwise, extract the Kea source from the original tarball that was downloaded. For example, with a download of Kea 1.9.0., there should be a tarball called kea-1.9.0.tar.gz on the system. Unpack this tarball:

```
$ tar zxvf kea-1.9.0.tar.gz
```

This will unpack the tarball into the kea-1.9.0 subdirectory of the current working directory.

3. Unpack the Kea premium tarball into the directory into which Kea was unpacked. Once Kea 1.9.0 has been unpacked into a kea-1.9.0 subdirectory and the Kea premium tarball is in the current directory, the following steps will unpack the premium tarball into the correct location:

```
$ cd kea-1.9.0
$ tar xvf ../kea-premium-1.9.0.tar.gz
```

Note that unpacking the Kea premium package will put the files into a directory named "premium". Regardless of the name of the package, the directory will always be called "premium", although its contents will vary depending on the premium package.

4. Run `autoreconf` tools. This step is necessary to update Kea's build script to include the additional directory. If this tool is not already available on the system, install the `automake` and `autoconf` tools. To generate the configure script, please use:

```
$ autoreconf -i
```

5. Rerun configure, using the same configure options that were used when originally building Kea. It is possible to verify that configure has detected the premium package by inspecting the summary printed when it exits. The first section of the output should look something like this:

```
Package:
  Name:              kea
  Version:           1.9.0
  Extended version: 1.9.0 (tarball)
  OS Family:         Linux
  Using GNU sed:     yes
  Premium package:  yes
  Included Hooks:    forensic_log flex_id host_cmds
```

The last line indicates which specific hooks were detected. Note that some hooks may require their own dedicated switches, e.g. the RADIUS hook requires extra switches for FreeRADIUS. Please consult later sections of this chapter for details.

6. Rebuild Kea.

```
$ make
```

If the machine has multiple CPU cores, an interesting option to consider here is using the argument -j X, where X is the number of available cores.

7. Install Kea sources along with the hooks:

```
$ sudo make install
```

Note that as part of the installation procedure, the install script will eventually venture into the premium/ directory and will install additional hook libraries and associated files.

The installation location of the hooks libraries depends on whether the –prefix parameter was specified in the configure script. If not, the default location will be /usr/local/lib/kea/hooks. The proper installation of the libraries can be verified with this command:

```
$ ls -l /usr/local/lib/kea/hooks/*.so
/usr/local/lib/kea/hooks/libdhcp_class_cmds.so
/usr/local/lib/kea/hooks/libdhcp_flex_id.so
/usr/local/lib/kea/hooks/libdhcp_flex_option.so
/usr/local/lib/kea/hooks/libdhcp_host_cmds.so
/usr/local/lib/kea/hooks/libdhcp_lease_cmds.so
/usr/local/lib/kea/hooks/libdhcp_legal_log.so
/usr/local/lib/kea/hooks/libdhcp_subnet_cmds.so
```

The exact list returned will depend on the packages installed. If the directory was specified via –prefix, the hooks libraries will be located in {prefix directory}/lib/kea/hooks.

16.3 Configuring Hooks Libraries

The hooks libraries for a given process are configured using the `hooks-libraries` keyword in the configuration for that process. (Note that the word "hooks" is plural.) The value of the keyword is an array of map structures, with each structure corresponding to a hooks library. For example, to set up two hooks libraries for the DHCPv4 server, the configuration would be:

```
"Dhcp4": {
    :
    "hooks-libraries": [
        {
            "library": "/opt/charging.so"
        },
        {
            "library": "/opt/local/notification.so",
            "parameters": {
                "mail": "spam@example.com",
                "floor": 13,
                "debug": false,
                "users": [ "alice", "bob", "charlie" ],
                "languages": {
                    "french": "bonjour",
                    "klingon": "yl'el"
                }
            }
        }
    ]
    :
}
```

Note: This syntax is effective as of Kea 1.1.0, to facilitate the specification of library-specific parameters. Libraries should allow a parameter entry for comments, as is the case with many configuration scopes.

Note: In all versions of Kea since 1.1.0, libraries are reloaded even if their lists have not changed, because the parameters specified for the library (or the files those parameters point to) may have changed.

Libraries may have additional parameters that are not mandatory, in the sense that there may be libraries that do not require them. However, for a specific library there is often a specific requirement to specify a certain set of parameters. Please consult the documentation for each individual library for details. In the example above, the first library has no parameters. The second library has five parameters: specifying mail (string parameter), floor (integer parameter), debug (boolean parameter), lists (list of strings), and maps (containing strings). Nested parameters can be used if the library supports it. This topic is explained in detail in the Hooks Developer's Guide section of the Kea Developer's Guide.

Notes:

- The full path to each library should be given.
- As noted above, the order in which the hooks are called may be important; consult the documentation for each library for specifics.
- An empty list has the same effect as omitting the `hooks-libraries` configuration element altogether.

  Note: There is one case where this is not true: if Kea is running with a configuration that contains a `hooks-libraries` item, and that item is removed and the configuration reloaded, the removal will be ignored and the libraries remain loaded. As a workaround, instead of removing the `hooks-libraries` item, change it to an empty list. This will be fixed in a future version of Kea.

At the present time, only the kea-dhcp4 and kea-dhcp6 processes support hooks libraries.

16.4 Available Hooks Libraries

As described above, the hooks functionality provides a way to customize a Kea server without modifying the core code. ISC has chosen to take advantage of this feature to provide functions that may only be useful to a subset of Kea users. To this end, ISC has created some hooks libraries, discussed in the following sections.

Note: Some of these libraries are available with the base code, while others will be shared with organizations supporting development of Kea. Users who would like to get access to those premium libraries should consider purchasing a support contract from ISC. This includes professional support, advance security notifications, input into ISC's roadmap planning, and many other benefits, while helping make Kea sustainable in the long term.

The following table provides a list of libraries currently available from ISC. It is important to pay attention to which libraries may be loaded by which Kea processes. It is a common mistake to configure the `kea-ctrl-agent` process to load libraries that should, in fact, be loaded by the `kea-dhcp4` or `kea-dhcp6` processes. If a library from ISC does not work as expected, please make sure that it has been loaded by the correct process per the table below.

Warning: While the Kea Control Agent includes the "hooks" functionality, (i.e. hooks libraries can be loaded by this process), none of ISC's current hooks libraries should be loaded by the Control Agent.

Table 16.1: List of Available Hooks Libraries

Name	Availability	Description
User Check	Kea sources (since 0.8)	Reads known users list from a file. Unknown users will be assigned a lease from the last subnet defined in the configuration file, e.g. to redirect them a captive portal. This demonstrates how an external source of information can be used to influence the Kea allocation engine. This hook is part of the Kea source code and is available in the src/hooks/dhcp/user_chk directory.
Forensic Logging	Support customers (since 1.1)	This library provides hooks that record a detailed log of lease assignments and renewals into a set of log files. In many legal jurisdictions companies, especially ISPs, must record information about the addresses they have leased to DHCP clients. This library is designed to help with that requirement. If the information that it records is sufficient it may be used directly. If your jurisdiction requires that you save a different set of information, you may use it as a template or example and create your own custom logging hooks.
Flexible Identifier	Support customers (since 1.2)	Kea software provides a way to handle host reservations that include addresses, prefixes, options, client classes and other features. The reservation can be based on hardware address, DUID, circuit-id or client-id in DHCPv4 and using hardware address or DUID in DHCPv6. However, there are sometimes scenarios where the reservation is more complex, e.g. uses other options that mentioned above, uses part of specific options or perhaps even a combination of several options and fields to uniquely identify a client. Those scenarios are addressed by the Flexible Identifiers hook application. It allows defining an expression, similar to the one used in client classification, e.g. substring(relay6[0].option[37],0,6). Each incoming packet is evaluated against that expression and its value is then searched in the reservations database.
Flexible Option	Kea sources (since 1.7.1)	This library provides hooks that compute option values instead of static configured values. An expression is evaluated on the query packet. Defined add, supersede and remove actions are applied on the response packet before it is sent using the evaluation result.
Host Commands	Support customers (since 1.2)	Kea provides a way to store host reservations in a database. In many larger deployments it is useful to be able to manage that information while the server is running. This library provides management commands for adding, querying and deleting host reservations in a safe way without restarting the server. In particular, it validates the parameters, so an attempt to insert incorrect data, e.g. add a host with conflicting identifier in the same subnet will be rejected. Those commands are exposed via command channel (JSON over unix sockets) and Control Agent (JSON over RESTful interface). Additional commands and capabilities related to host reservations will be added in the future.
Subnet Commands	Support customers (since 1.3)	In deployments in which subnet configuration needs to be frequently updated, it is a hard requirement that such updates be performed without the need for a full DHCP server reconfiguration or restart. This hooks library allows for incremental changes to the subnet configuration such as: adding a subnet, removing a subnet. It also allows for listing all available subnets and fetching detailed information about a selected subnet. The commands exposed by this library do not affect other subnets or configuration parameters currently used by the server.
Lease Commands	Kea sources (since 1.3)	The lease commands hook library offers a number of new commands used to manage leases. Kea provides a way to store lease information in various backends: memfile, MySQL, PostgreSQL and Cassandra. This library provides a unified interface that can manipulate leases in an unified, safe way. In particular, it allows: manipulate leases in memfile while Kea is running, sanity check changes, check lease existence and remove all leases belonging to specific subnet. It can also catch more obscure errors, like adding a lease with subnet-id that does not exist in the configuration or configuring a lease to use an address that is outside of the subnet to which it is supposed to belong. It provides a way to manage user contexts associated with leases.

Continued on next page

Table 16.1 – continued from previous page

Name	Availability	Description
High Availability	Kea sources (since 1.4)	Minimizing a risk of DHCP service unavailability is achieved by setting up a pair of the DHCP servers in a network. Two modes of operation are supported. The first one is called load balancing and is sometimes referred to as active-active. Each server can handle selected group of clients in this network or all clients, if it detects that its partner has became unavailable. It is also possible to designate one server to serve all DHCP clients, and leave another server as "standby". This mode is called hot standby and is sometimes referenced to as active-passive. This server will activate its DHCP function when it detects that its partner is not available. Such cooperation between the DHCP servers requires that these servers constantly communicate with each other to send updates about allocated leases and to periodically test whether their partners are still operational. The hook library also provides an ability to send lease updates to external backup server, making it much easier to have a replacement that is almost up to date. The "libdhcp_ha" library provides such functionality for Kea DHCP servers.
Statistics Commands	Kea sources (since 1.4)	The Statistics Commands library provides additional commmands for retrieving accurate DHCP lease statistics for Kea DHCP servers that share the same lease database. This setup is common in deployments where DHCP service redundancy is required and a shared lease database is used to avoid lease data replication between the DHCP servers. A feature was introduced in Kea 1.4.0 that allows tracking lease allocations within the lease database, thus making the statistics accessible to all connected DHCP servers. The Statistics Commands hooks library utilizes this feature and returns lease statistics for all subnets respectively.
RADIUS	Support customers (since 1.4)	The RADIUS Hook library allows Kea to interact with the RADIUS servers using access and accounting mechanisms. The access mechanism may be used for access control, assigning specific IPv4 or IPv6 addresses reserved by RADIUS, dynamically assigning addresses from designated pools chosen by RADIUS or rejecting the client's messages altogether. The accounting mechanism allows RADIUS server to keep track of device activity over time.
Host Cache	Support customers (since 1.4)	Some of the database backends, such as RADIUS, are considered slow and may take a long time to respond. Since Kea in general is synchronous, the backend performance directly affects the DHCP performance. To minimize the impact and improve performance, the Host Cache library provides a way to cache responses from other hosts. This includes negative caching, i.e. the ability to remember that there is no client information in the database.
Class Commands	Support customers (since 1.5)	This Class Cmds hooks library allows for adding, updating deleting and fetching configured DHCP client classes without the need to restart the DHCP server.
MySQL Configuration Backend	Kea sources (since 1.6)	The MySQL CB hooks library is an implementation of the Kea Configuration Backend for MySQL. It uses MySQL database as a repository for the Kea configuration information. The Kea servers use this library to fetch their configurations.
Configuration Backend Commands	Support customers (since 1.6)	The Configuration Backend Commands (CB Commands) hooks library implements a collection of commands to manage the configuration information of the Kea servers in the database. This library may only be used in conjuction with one of the supported configuration backend implementations.
BOOTP	Kea sources (since 1.7.3)	The BOOTP hooks library adds BOOTP support, as defined in RFC 1497. It recognizes received BOOTP requests: they are translated into DHCPREQUEST packets, put into the BOOTP client class and get infinite lifetime leases.
Leasequery	Support customers (DHCPv4 since 1.7.8, DHCPv6 since 1.7.9)	The Leasequery hooks library adds support for DHCPv4 Leasequery as described in RFC 4388; and for DHCPv6 Leasequery as described in RFC 5007.

ISC hopes to see more hooks libraries become available as time progresses, developed both internally and externally. Since this list may evolve dynamically, it is maintained on a wiki page, available at this link: https://gitlab.isc.org/isc-projects/kea/wikis/Hooks-available. Developers or others who are aware of any hooks libraries not listed there are asked to please send a note to the kea-users or kea-dev mailing lists for updating.

The libraries developed by ISC are described in detail in the following sections.

16.5 user_chk: Checking User Access

The user_chk library is the first hooks library published by ISC. It serves several purposes:

- To assign "new" or "unregistered" users to a restricted subnet, while "known" or "registered" users are assigned to unrestricted subnets.
- To allow DHCP response options or vendor option values to be customized based on user identity.
- To provide a real-time record of user registration activity, which can be sampled by an external consumer.
- To serve as a demonstration of various capabilities possible using the hooks interface.

Once loaded, the library allows the separation of incoming requests into known and unknown clients. For known clients, packets are processed as usual, although it is possible to override the sending of certain options on a per-host basis. Clients that are not on the known hosts list will be treated as unknown and will be assigned to the last subnet defined in the configuration file.

As an example of a use case, this behavior may be implemented to put unknown users into a separate subnet that leads to a "walled garden," where they can only access a registration portal. Once they fill in necessary data, their details are added to the known clients file and they get a proper address after their device is restarted.

Note: This library was developed several years before the host reservation mechanism became available. Host reservation is much more powerful and flexible, but the user_chk capability to consult an external source of information about clients and alter Kea's behavior remains useful and of educational value.

The library reads the /tmp/user_chk_registry.txt file while being loaded and each time an incoming packet is processed. Each line of the file is expected to contain a self-contained JSON snippet which must have the following two entries:

- `type` - whose value is "HW_ADDR" for IPv4 users or "DUID" for IPv6 users.
- `id` - whose value is either the hardware address or the DUID from the request formatted as a string of hex digits, with or without ":" delimiters.

and may have zero or more of the following entries:

- `bootfile` - whose value is the pathname of the desired file.
- `tftp_server` - whose value is the hostname or IP address of the desired server.

A sample user registry file is shown below:

```
{ "type" : "HW_ADDR", "id" : "0c:0e:0a:01:ff:04", "bootfile" : "/tmp/v4bootfile" }
{ "type" : "HW_ADDR", "id" : "0c:0e:0a:01:ff:06", "tftp_server" : "tftp.v4.example.com
↪" }
{ "type" : "DUID", "id" : "00:01:00:01:19:ef:e6:3b:00:0c:01:02:03:04", "bootfile" : "/
↪tmp/v6bootfile" }
{ "type" : "DUID", "id" : "00:01:00:01:19:ef:e6:3b:00:0c:01:02:03:06", "tftp_server" :
↪ "tftp.v6.example.com" }
```

As with any other hooks libraries provided by ISC, internals of the user_chk code are well-documented. Users may refer to the user_chk library section of the Kea Developer's Guide for information on how the code works internally. That, together with the Hooks Framework section of the Kea Developer's Guide should give users some pointers on how to extend this library and perhaps even write one from scratch.

16.6 legal_log: Forensic Logging Hooks

This section describes the forensic log hooks library. This library provides hooks that record a detailed log of lease assignments and renewals into a set of log files.

Currently this library is only available to ISC customers with a paid support contract.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

In many legal jurisdictions, companies, especially ISPs, must record information about the addresses they have leased to DHCP clients. This library is designed to help with that requirement. If the information that it records is sufficient, it may be used directly. If a jurisdiction requires that a different set of information be saved, users may use this library as a template or example to create their own custom logging hooks.

This logging is done as a set of hooks to allow it to be customized to any particular need. Modifying a hooks library is easier and safer than updating the core code. In addition by using the hooks features, those users who do not need to log this information can leave it out and avoid any performance penalties.

16.6.1 Log File Naming

The names for the log files have the following form:

```
path/base-name.CCYYMMDD.txt
```

The "path" and "base-name" are supplied in the configuration as described below; see *Configuring the Forensic Log Hooks*. The next part of the name is the date the log file was started, with four digits for year, two digits for month, and two digits for day. The file is rotated on a daily basis.

Note: When running Kea servers for both DHCPv4 and DHCPv6, the log names must be distinct. See the examples in *Configuring the Forensic Log Hooks*.

16.6.2 DHCPv4 Log Entries

For DHCPv4, the library creates entries based on DHCPREQUEST messages and corresponding DHCPv4 leases intercepted by the lease4_select (for new leases) and the lease4_renew (for renewed leases) hooks.

An entry is a single string with no embedded end-of-line markers and a prepended timestamp, and has the following sections:

```
timestamp address duration device-id {client-info} {relay-info} {user-context}
```

Where:

- timestamp - the current date and time the log entry was written in "%Y-%m-%d %H:%M:%S %Z" strftime format ("%Z" is the time zone name).

- address - the leased IPv4 address given out and whether it was assigned or renewed.
- duration - the lease lifetime expressed in days (if present), hours, minutes, and seconds. A lease lifetime of 0xFFFFFFFF will be denoted with the text "infinite duration".
- device-id - the client's hardware address shown as numerical type and hex digit string.
- client-info - the DHCP client id option (61) if present, shown as a hex string. When its content is printable it is displayed.
- relay-info - for relayed packets the giaddr and the RAI circuit-id, remote-id, and subscriber-id options (option 82 sub options: 1, 2 and 6) if present. The circuit id and remote id are presented as hex strings. When their content is printable it is displayed.
- user-context - the optional user context associated with the lease.

For instance (line breaks added for readability; they will not be present in the log file):

```
2018-01-06 01:02:03 CET Address: 192.2.1.100 has been renewed for 1 hrs 52 min 15␣
↪secs to a device with hardware address:
hwtype=1 08:00:2b:02:3f:4e, client-id: 17:34:e2:ff:09:92:54 connected via relay at␣
↪address: 192.2.16.33,
identified by circuit-id: 68:6f:77:64:79 (howdy) and remote-id: 87:f6:79:77:ef
```

In addition to logging lease activity driven by DHCPv4 client traffic, the hooks library also logs entries for the following lease management control channel commands: lease4-add, lease4-update, and lease4-del. Each entry is a single string with no embedded end-of-line markers, and it will typically have the following form:

`lease4-add:`

```
*timestamp* Administrator added a lease of address: *address* to a device with␣
↪hardware address: *device-id*
```

Depending on the arguments of the add command, it may also include the client-id and duration.

Example:

```
2018-01-06 01:02:03 CET Administrator added a lease of address: 192.0.2.202 to a␣
↪device with hardware address:
1a:1b:1c:1d:1e:1f for 1 days 0 hrs 0 mins 0 secs
```

`lease4-update:`

```
*timestamp* Administrator updated information on the lease of address: *address* to a␣
↪device with hardware address: *device-id*
```

Depending on the arguments of the update command, it may also include the client-id and lease duration.

Example:

```
2018-01-06 01:02:03 CET Administrator updated information on the lease of address:␣
↪192.0.2.202 to a device
with hardware address: 1a:1b:1c:1d:1e:1f, client-id: 1234567890
```

`lease4-del:` deletes have two forms, one by address and one by identifier and identifier type:

```
*timestamp* Administrator deleted the lease for address: *address*
```

or

```
*timestamp* Administrator deleted a lease for a device identified by: *identifier-
↪type* of *identifier*
```

Currently only a type of @b hw-address (hardware address) is supported.

Examples:

```
2018-01-06 01:02:03 CET Administrator deleted the lease for address: 192.0.2.202

2018-01-06 01:02:12 CET Administrator deleted a lease for a device identified by: hw-
↪address of 1a:1b:1c:1d:1e:1f
```

16.6.3 DHCPv6 Log Entries

For DHCPv6 the library creates entries based on lease management actions intercepted by lease6_select (for new leases), lease6_renew (for renewed leases), and lease6_rebind (for rebound leases).

An entry is a single string with no embedded end-of-line markers and a prepended timestamp, and has the following sections:

```
timestamp address duration device-id {relay-info}* {user-context}
```

Where:

- timestamp - the current date and time the log entry was written in "%Y-%m-%d %H:%M:%S %Z" strftime format ("%Z" is the time zone name).
- address - the leased IPv6 address or prefix given out and whether it was assigned or renewed.
- duration - the lease lifetime expressed in days (if present), hours, minutes, and seconds. A lease lifetime of 0xFFFFFFFF will be denoted with the text "infinite duration".
- device-id - the client's DUID and hardware address (if present).
- relay-info - for relayed packets the content of relay agent messages, remote-id (code 37), subscriber-id (code 38), and interface-id (code 18) options, if present. Note that interface-id option, if present, identifies the whole interface the relay agent received the message on. This typically translates to a single link in the network, but it depends on the specific network topology. Nevertheless, this is useful information to better scope down the location of the device, so it is recorded, if present.
- user-context - the optional user context associated with the lease.

For instance (line breaks added for readability; they will not be present in the log file):

```
2018-01-06 01:02:03 PST Address:2001:db8:1:: has been assigned for 0 hrs 11 mins 53␣
↪secs
to a device with DUID: 17:34:e2:ff:09:92:54 and hardware address: hwtype=1 08:00:2b:
↪02:3f:4e
(from Raw Socket) connected via relay at address: fe80::abcd for client on link␣
↪address: 3001::1,
hop count: 1, identified by remote-id: 01:02:03:04:0a:0b:0c:0d:0e:0f and subscriber-
↪id: 1a:2b:3c:4d:5e:6f
```

In addition to logging lease activity driven by DHCPv6 client traffic, the hooks library also logs entries for the following lease management control channel commands: lease6-add, lease6-update, and lease6-del. Each entry is a single string with no embedded end-of-line markers, and it will typically have the following form:

`lease6-add:`

```
*timestamp* Administrator added a lease of address: *address* to a device with DUID:␣
↪*DUID*
```

Depending on the arguments of the add command, it may also include the hardware address and duration.

Example:

```
2018-01-06 01:02:03 PST Administrator added a lease of address: 2001:db8::3 to a␣
↪device with DUID:
1a:1b:1c:1d:1e:1f:20:21:22:23:24 for 1 days 0 hrs 0 mins 0 secs
```

`lease6-update:`

```
*timestamp* Administrator updated information on the lease of address: *address* to a␣
↪device with DUID: *DUID*
```

Depending on the arguments of the update command, it may also include the hardware address and lease duration.

Example:

```
2018-01-06 01:02:03 PST Administrator updated information on the lease of address:␣
↪2001:db8::3 to a device with
DUID: 1a:1b:1c:1d:1e:1f:20:21:22:23:24, hardware address: 1a:1b:1c:1d:1e:1f
```

`lease6-del:` deletes have two forms, one by address and one by identifier and identifier type:

```
*timestamp* Administrator deleted the lease for address: *address*
```

or

```
*timestamp* Administrator deleted a lease for a device identified by: *identifier-
↪type* of *identifier*
```

Currently only a type of DUID is supported.

Examples:

```
2018-01-06 01:02:03 PST Administrator deleted the lease for address: 2001:db8::3

2018-01-06 01:02:11 PST Administrator deleted a lease for a device identified by:␣
↪duid of 1a:1b:1c:1d:1e:1f:20:21:22:23:24
```

16.6.4 Configuring the Forensic Log Hooks

To use this functionality, the hook library must be included in the configuration of the desired DHCP server modules. The legal_log library is able to save logs to a text file or a database (created using `kea-admin` see *First-Time Creation of the MySQL Database*, *First-Time Creation of the PostgreSQL Database*). Library is installed alongside the Kea libraries in `[kea-install-dir]/var/lib/kea` where `kea-install-dir` is determined by the “–prefix” option of the configure script. It defaults to `/usr/local`. Assuming the default value, configuring kea-dhcp4 to load the legal_log library could be done with the following Kea4 configuration:

```
"Dhcp4": {
    "hooks-libraries": [
        {
            "library": "/usr/local/lib/kea/hooks/libdhcp_legal_log.so",
            "parameters": {
                "path": "/var/lib/kea/log",
                "base-name": "kea-forensic4"
            }
        },
        ...
    ]
}
```

To configure it for kea-dhcp6, the commands are:

```
"Dhcp6": {
    "hooks-libraries": [
        {
            "library": "/usr/local/lib/kea/hooks/libdhcp_legal_log.so",
            "parameters": {
                "path": "/var/lib/kea/log",
                "base-name": "kea-forensic6"
            }
        },
        ...
    ]
}
```

Two hooks library parameters for text file are supported:

- path - the directory in which the forensic file(s) will be written. The default value is `[prefix]/var/lib/kea`. The directory must exist.
- base-name - an arbitrary value which is used in conjunction with the current system date to form the current forensic file name. It defaults to `kea-legal`.

Additional parameters for the database connection can be specified, e.g:

```
 "Dhcp6": {
    "hooks-libraries": [
        {
            "library": "/usr/local/lib/kea/hooks/libdhcp_legal_log.so",
            "parameters": {
                "name":"database-name",
                "password":"passwd",
                "type":"mysql",
                "user":"user-name"
            }
        },
        ...
    ]
}
```

For more specific information about database related parameters please refer to *Lease Database Configuration* and *Lease Database Configuration*.

If it is desired to restrict forensic logging to certain subnets, the “legal-logging” boolean parameter can be specified within a user context of these subnets. For example:

```
"Dhcpv4" {
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [
                {
                        "pool": "192.0.2.1 - 192.0.2.200"
                   }
               ],
               "user-context": {
                   "legal-logging": false
               }
           }
       ]
}
```

This configuration disables legal logging for the subnet "192.0.2.0/24". If the "legal-logging" parameter is not specified, it defaults to 'true', which enables legal logging for the subnet.

The following example demonstrates how to selectively disable legal logging for an IPv6 subnet:

```
"Dhcpv6": {
    "subnet6": [
        {
            "subnet": "2001:db8:1::/64",
            "pools": [
                 {
                     "pool": "2001:db8:1::1-2001:db8:1::ffff"
                 }
            ],
            "user-context": {
                "legal-logging": false
            }
        }
    ]
}
```

See *User Contexts in IPv4* and *User Contexts in IPv6* to learn more about user contexts in Kea configuration.

16.6.5 Database Backend

Log entries can be inserted into a database when Kea is configured with database backend support. A table named "logs" is used that includes a timestamp (timeuuid for Cassandra) generated by the database software, and a text log with the same format as files without the timestamp.

Please refer to *MySQL* for information on using a MySQL database; to *PostgreSQL* for PostgreSQL database information; or to *Cassandra* for information on using a Cassandra (CQL) database. The logs table is part of the Kea database schemas.

Configuration parameters are extended by standard lease database parameters as defined in *Lease Database Configuration*. The "type" parameter should be "mysql", "postgresql", "cql", or "logfile". When it is absent or set to "logfile", files are used.

This database feature is experimental and will be likely improved, for instance to add an address/prefix index (currently the only index is the timestamp). No specific tools are provided to operate the database, but standard tools may be used, for example, to dump the logs table from a CQL database:

```
$ echo 'SELECT dateOf(timeuuid), log FROM logs;' | cqlsh -k database-name

 system.dateof(timeuuid)         | log
---------------------------------+---------------------------------------
 2018-01-06 01:02:03.227000+0000 | Address: 192.2.1.100 has been renewed ...
 ...
```

```
(12 rows)
$
```

16.7 flex_id: Flexible Identifiers for Host Reservations

This section describes a hook application dedicated to generate flexible identifiers for host reservations. The Kea software provides a way to handle host reservations that include addresses, prefixes, options, client classes, and other features. The reservation can be based on hardware address, DUID, circuit-id, or client-id in DHCPv4 and on hardware address or DUID in DHCPv6. However, there are sometimes scenarios where the reservation is more complex; it may use options other than those mentioned above, use parts of specific options, or perhaps even use a combination of several options and fields to uniquely identify a client. Those scenarios are addressed by the Flexible Identifiers hook application.

Currently this library is only available to ISC customers with a paid support contract.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

The library allows the definition of an expression, using notation initially used only for client classification. (See *Using Expressions in Classification* for a detailed description of the syntax available.) One notable difference is that for client classification, the expression currently has to evaluate to either true or false, while the flexible identifier expression is expected to evaluate to a string that will be used as an identifier. It is a valid case for the expression to evaluate to an empty string (e.g. in cases where a client does not send specific options). This expression is then evaluated for each incoming packet, and this evaluation generates an identifier that is used to identify the client. In particular, there may be host reservations that are tied to specific values of the flexible identifier.

The library can be loaded in a similar way as other hook libraries. It takes a mandatory parameter `identifier-expression` and optional boolean parameter `replace-client-id`:

```
"Dhcp6": {
    "hooks-libraries": [
        {
            "library": "/path/libdhcp_flex_id.so",
            "parameters": {
                "identifier-expression": "expression",
                "replace-client-id": false
            }
        },
        ...
    ]
}
```

The flexible identifier library supports both DHCPv4 and DHCPv6.

Let's consider a case of an IPv6 network that has an independent interface for each of its connected customers. Customers are able to plug in whatever device they want, so any type of identifier (e.g. a client-id) is unreliable. Therefore, the operator may decide to use an option inserted by a relay agent to differentiate between clients. In this particular deployment, the operator has verified that the interface-id is unique for each customer-facing interface, so it is suitable for usage as a reservation. However, only the first six bytes of the interface-id are interesting, because remaining bytes are either randomly changed or not unique between devices. Therefore, the customer decided to use the first six bytes of the interface-id option inserted by the relay agent. After adding `flex-id`, the `host-reservation-identifiers` goal can be achieved by using the following configuration:

```
"Dhcp6": {
    "subnet6": [{ ..., # subnet definition starts here
    "reservations": [
        "flex-id": "'port1234'", # value of the first 8 bytes of the interface-id
        "ip-addresses": [ "2001:db8::1" ]
    ],
    }], # end of subnet definitions
    "host-reservation-identifiers": ["duid", "flex-id"], # add "flex-id" to␣
↪reservation identifiers
    "hooks-libraries": [
        {
            "library": "/path/libdhcp_flex_id.so",
            "parameters": {
                "identifier-expression": "substring(relay6[0].option[18].hex,0,8)"
            }
        },
        ...
    ]
}
```

Note: Care should be taken when adjusting the expression. If the expression changes, then all the `flex-id` values may change, possibly rendering all reservations based on `flex-id` unusable until they are manually updated. It is strongly recommended that administrators start with the expression and a handful of reservations, and then adjust the expression as needed. Once the expression is confirmed to do what is desired of it, host reservations can be deployed on a broader scale.

`flex-id` values in host reservations can be specified in two ways. First, they can be expressed as a hex string, e.g. bar string can be represented as 626174. Alternatively, it can be expressed as a quoted value (using double and single quotes), e.g. “‘bar’”. The former is more convenient for printable characters, while hex string values are more convenient for non-printable characters and do not require the use of the `hexstring` operator.

```
"Dhcp6": {
    "subnet6": [{ ..., # subnet definition starts here
    "reservations": [
        "flex-id": "01:02:03:04:05:06", # value of the first 8 bytes of the interface-
↪id
        "ip-addresses": [ "2001:db8::1" ]
    ],
    }], # end of subnet definitions
    "host-reservation-identifiers": ["duid", "flex-id"], # add "flex-id" to␣
↪reservation identifiers
    "hooks-libraries": [
        {
            "library": "/path/libdhcp_flex_id.so",
            "parameters": {
                "identifier-expression": "vendor[4491].option[1026].hex"
            }
        },
        ...
    ]
}
```

When `replace-client-id` is set to “false” (which is the default setting), the flex-id hook library uses the evaluated flexible identifier solely for identifying host reservations, i.e. searching for reservations within a database. This is the functional equivalent of other identifiers, similar to hardware address or circuit-id. However, this mode of operation implies that if a client device is replaced, it may cause a conflict between an existing lease (allocated to the old device) and the new lease being allocated to the new device. The conflict arises because the same flexible identifier is computed for the replaced device, so the server will try to allocate the same lease. The mismatch between client identifiers sent by the new device and the old device causes the server to refuse this new allocation until the old lease expires. A manifestation of this problem is dependent on the specific expression used as the flexible identifier and is likely to appear if only options and other parameters are used that identify where the device is connected (e.g. circuit-id), rather than the device identification itself (e.g. MAC address).

The flex-id library offers a way to overcome the problem with lease conflicts by dynamically replacing the client identifier (or DUID in DHCPv6) with a value derived from the flexible identifier. The server processes the client's query as if the flexible identifier were sent in the client identifier (or DUID) option. This guarantees that a returning client (for which the same flexible identifier is evaluated) will be assigned the same lease despite the client identifier and/or MAC address change.

The following is a stub configuration that enables this behavior:

```
"Dhcp4": {
    "hooks-libraries": [
        {
            "library": "/path/libdhcp_flex_id.so",
            "parameters": {
                "identifier-expression": "expression",
                "replace-client-id": true
            }
        },
        ...
    ]
}
```

In the DHCPv4 case, the value derived from the flexible identifier is formed by prepending one byte with a value of zero to the flexible identifier. In the DHCPv6 case, it is formed by prepending two zero bytes before the flexible identifier.

Note that for this mechanism to take effect, the DHCPv4 server must be configured to respect the client identifier option value during lease allocation, i.e. `match-client-id` must be set to "true". See *Using Client Identifier and Hardware Address* for details. No additional settings are required for DHCPv6.

If the `replace-client-id` option is set to "true", the value of the `echo-client-id` parameter (which governs whether to send back a client-id option) is ignored.

The *lease_cmds: Lease Commands* section describes commands used to retrieve, update, and delete leases using various identifiers, such as "hw-address" and "client-id". The lease_cmds library does not natively support querying for leases by flexible identifier. However, when `replace-client-id` is set to "true", it makes it possible to query for leases using a value derived from the flexible identifier. In the DHCPv4 case, the query will look similar to this:

```
{
    "command": "lease4-get",
    "arguments": {
        "identifier-type": "client-id",
        "identifier": "00:54:64:45:66",
        "subnet-id": 44
    }
}
```

where the hexadecimal value of "54:64:45:66" is a flexible identifier computed for the client.

In the DHCPv6 case, the corresponding query will look similar to this:

```
{
    "command": "lease6-get",
    "arguments": {
        "identifier-type": "duid",
        "identifier": "00:00:54:64:45:66",
        "subnet-id": 10
    }
}
```

16.8 flex_option Flexible Option for Option value settings

This library allows you to define an action to take, for a given option, based upon on the result of an expression. These actions are carried out during the final stages of constructing a query response packet, just before it is sent to the client. The three actions currently supported are `add`, `supersede`, and `remove`.

The syntax used for the action expressions is the same syntax used for client classification and the Flex Identifier hook library (See either *Using Expressions in Classification* or *flex_id: Flexible Identifiers for Host Reservations* for detailed description of the syntax).

The `add` and `supersede` actions use an expression returning a string, doing nothing when it evaluates to the empty string. The `remove` application uses an expression returning true or false, doing nothing on false. When it is necessary to set an option to the empty value this mechanism does not work but a client class can be used instead.

The `add` action adds an option only when the option does not already exist and the expression does not evaluate to the empty string. The `supersede` action does the same but it overwrites the option value if it already exists. The `remove` action removes the option from the response packet if it already exists and the expression evaluates to true.

The option to which an action applies may be specified by either its numeric code or its name.. At least the code or the name must be specified. The option space is the DHCPv4 or DHCPv6 spaces depending of the server where the hook library is loaded. Other spaces as vendor spaces could be supported in a further version.

The library is available since Kea 1.7.1 and can be loaded in a similar way as other hook libraries by the `kea-dhcp4` or *kea-dhcp6‘* process.. It takes a mandatory `options` parameter holding a list of per option parameter maps with code, name, add, supersede and remove actions. Action entries take a string value representing an expression.

```
  "Dhcp4": {
      "hook_libraries": [
          {   "library": "/usr/local/lib/libdhcp_flex_option.so",
              "parameters": {
                  "options": [
                          {
                          "code": 67,
                          "add":
"ifelse(option[host-name].exists,concat(option[host-name].text,'.boot'),'')"
                      }
                  ]
              }
          },
          ...
      ]
  }
```

If (and only if) the query includes a host-name option (code 12), a boot-file-name option (code 67) is added to the response with the host name followed by .boot for content.

The flexible option library supports both DHCPv4 and DHCPv6.

Since Kea 1.9.0, the add and supersede actions take an optional cvs-format boolean parameter. If not specified or configured to false, the option data is set using the raw value of the evaluated expression. When it is configured to true, this value is parsed using the option definition from the option data specified in the configuration file. This eases option setting for options using complex record formats or fully qualified domain names.

For instance if the expression evaluation returns "example.com" and the option is defined with the fqdn type the domain name will be encoded into DNS binary format.

16.9 host_cmds: Host Commands

This section describes a hook application that offers a number of new commands used to query and manipulate host reservations. Kea provides a way to store host reservations in a database. In many larger deployments it is useful to be able to manage that information while the server is running. This library provides management commands for adding, querying, and deleting host reservations in a safe way without restarting the server. In particular, it validates the parameters, so an attempt to insert incorrect data - such as adding a host with a conflicting identifier in the same subnet - will be rejected. Those commands are exposed via the command channel (JSON over UNIX sockets) and the Control Agent (JSON over a RESTful interface). Additional commands and capabilities related to host reservations will be added in the future.

Currently this library is only available to ISC customers with a paid support contract.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

Currently, six commands are supported: reservation-add (which adds a new host reservation), reservation-get (which returns an existing reservation if specified criteria are matched), reservation-get-all (which returns all reservations in a specified subnet), reservation-get-page (a variant of reservation-get-all which returns all reservations in a specified subnet by pages and since Kea version 1.9.0 all reservations), reservation-get-by-hostname (which returns all reservations with a specified hostname and optionally in a subnet) since Kea version 1.7.1, reservation-get-by-id (which returns all reservations with a specified identifier) since Kea version 1.9.0, and reservation-del (which attempts to delete a reservation matching specified criteria). To use commands that change the reservation information (currently these are reservation-add and reservation-del, but this rule applies to other commands that may be implemented in the future), the hosts database must be specified and it must not operate in read-only mode (see the hosts-databases descriptions in *DHCPv4 Hosts Database Configuration* and *DHCPv6 Hosts Database Configuration*). If the hosts-databases are not specified or are running in read-only mode, the host_cmds library will load, but any attempts to use reservation-add or reservation-del will fail.

Additional host reservation commands are planned in future releases of Kea. For a description of envisaged commands, see the Control API Requirements document.

All commands use JSON syntax. They can be issued either using the control channel (see *Management API*) or via the Control Agent (see *The Kea Control Agent*).

The library can be loaded similarly to other hook libraries. It does not take any parameters, and it supports both DHCPv4 and DHCPv6 servers.

```
"Dhcp6": {
    "hooks-libraries": [
        {
            "library": "/path/libdhcp_host_cmds.so"
        }
        ...
    ]
}
```

16.9.1 The subnet-id Parameter

Prior to diving into the individual commands, it is worth discussing the parameter, `subnet-id`. Currently this parameter is mandatory for all of the commands supplied by this library with the exception of reservation-get-by-hostname where it is optional, and since Kea 1.9.0 reservation-get-page where it is optional and reservation-get-by-id where it is forbidden. In previous versions of Kea, reservations had to belong to a specific subnet; as of Kea 1.5.0, reservations may be specified globally. In other words, they are not specific to any subnet. When reservations are supplied via the configuration file, the ID of the containing subnet (or lack thereof) is implicit in the configuration structure. However, when managing reservations using host commands, it is necessary to explicitly identify the scope to which the reservation belongs. This is done via the `subnet-id` parameter. For global reservations, use a value of zero (0). For reservations scoped to a specific subnet, use that subnet's ID.

At the opposite when the subnet id is not specified in the command parameters it is added to each host in responses. If the subnet id has the unused special value this means the host entry belongs only to the other IP version (i.e. IPv6 in DHCPv4 server or IPv4 in DHCPv6 server) and this entry is ignored.

16.9.2 The reservation-add Command

`reservation-add` allows for the insertion of a new host. It takes a set of arguments that vary depending on the nature of the host reservation. Any parameters allowed in the configuration file that pertain to host reservation are permitted here. For details regarding IPv4 reservations, see *Host Reservation in DHCPv4*; for IPv6 reservations, see *Host Reservation in DHCPv6*. The `subnet-id` is mandatory. Use a value of zero (0) to add a global reservation, or the id of the subnet to which the reservation should be added. An example command can be as simple as:

```
{
    "command": "reservation-add",
    "arguments": {
        "reservation": {
            "subnet-id": 1,
            "hw-address": "1a:1b:1c:1d:1e:1f",
            "ip-address": "192.0.2.202"
        }
    }
}
```

but it can also take many more parameters, for example:

```
{
    "command": "reservation-add",
    "arguments": {
        "reservation":
            {
                "subnet-id":1,
                "client-id": "01:0a:0b:0c:0d:0e:0f",
                "ip-address": "192.0.2.205",
                "next-server": "192.0.2.1",
                "server-hostname": "hal9000",
                "boot-file-name": "/dev/null",
                "option-data": [
                    {
                        "name": "domain-name-servers",
                        "data": "10.1.1.202,10.1.1.203"
                    }
                ],
                "client-classes": [ "special_snowflake", "office" ]
            }
    }
}
```

Here is an example of a complex IPv6 reservation:

```
{
    "command": "reservation-add",
    "arguments": {
        "reservation":
            {
                "subnet-id":1,
                "duid": "01:02:03:04:05:06:07:08:09:0A",
                "ip-addresses": [ "2001:db8:1:cafe::1" ],
                "prefixes": [ "2001:db8:2:abcd::/64" ],
                "hostname": "foo.example.com",
                "option-data": [
                    {
                        "name": "vendor-opts",
                        "data": "4491"
                    },
                    {
                        "name": "tftp-servers",
                        "space": "vendor-4491",
                        "data": "3000:1::234"
                    }
                ]
            }
    }
}
```

The command returns a status that indicates either a success (result 0) or a failure (result 1). A failed command always includes a text parameter that explains the cause of the failure. Example results:

```
{ "result": 0, "text": "Host added." }
```

Example failure:

```
{ "result": 1, "text": "Mandatory 'subnet-id' parameter missing." }
```

As `reservation-add` is expected to store the host, the hosts-databases parameter must be specified in the configuration and databases must not run in read-only mode. In future versions of Kea, it will be possible to modify the reservations read from a configuration file. Interested parties are encouraged to contact ISC for more information on developing this functionality.

16.9.3 The reservation-get Command

`reservation-get` can be used to query the host database and retrieve existing reservations. There are two types of parameters this command supports: (subnet-id, address) or (subnet-id, identifier-type, identifier). The first type of query is used when the address (either IPv4 or IPv6) is known, but the details of the reservation are not. One common use case of this type of query is to find out whether a given address is reserved. The second query uses identifiers. For maximum flexibility, Kea stores the host identifying information as a pair of values: the type and the actual identifier. Currently supported identifiers are “hw-address”, “duid”, “circuit-id”, “client-id”, and “flex-id”, but additional types may be added in the future. If any new identifier types are defined in the future, the reservation-get command will support them automatically. The `subnet-id` is mandatory. Use a value of zero (0) to fetch a global reservation, or the id of the subnet to which the reservation belongs.

An example command for getting a host reservation by a (subnet-id, address) pair looks as follows:

```
{
    "command": "reservation-get",
    "arguments": {
        "subnet-id": 1,
        "ip-address": "192.0.2.202"
    }
}
```

An example query by (subnet-id, identifier-type, identifier) looks as follows:

```
{
    "command": "reservation-get",
    "arguments": {
        "subnet-id": 4,
        "identifier-type": "hw-address",
        "identifier": "01:02:03:04:05:06"
    }
}
```

`reservation-get` typically returns the result 0 when the query was conducted properly. In particular, 0 is returned when the host was not found. If the query was successful, a number of host parameters will be returned. An example of a query that did not find the host looks as follows:

```
{ "result": 0, "text": "Host not found." }
```

An example result returned when the host was found looks like this:

```
{
  "arguments": {
    "boot-file-name": "bootfile.efi",
    "client-classes": [

    ],
    "hostname": "somehost.example.org",
    "hw-address": "01:02:03:04:05:06",
    "ip-address": "192.0.2.100",
    "next-server": "192.0.0.2",
    "option-data": [

    ],
    "server-hostname": "server-hostname.example.org"
  },
  "result": 0,
  "text": "Host found."
}
```

An example result returned when the query was malformed might look like this:

```
{ "result": 1, "text": "No 'ip-address' provided and 'identifier-type'
                        is either missing or not a string." }
```

16.9.4 The reservation-get-all Command

`reservation-get-all` can be used to query the host database and retrieve all reservations in a specified subnet. This command uses parameters providing the mandatory subnet-id. Global host reservations can be retrieved by using a subnet-id value of zero (0).

For instance, retrieving host reservations for the subnet 1:

```
{
    "command": "reservation-get-all",
    "arguments": {
        "subnet-id": 1
     }
}
```

returns some IPv4 hosts:

```
{
    "arguments": {
        "hosts": [
            {
                "boot-file-name": "bootfile.efi",
                "client-classes": [ ],
                "hostname": "somehost.example.org",
                "hw-address": "01:02:03:04:05:06",
                "ip-address": "192.0.2.100",
                "next-server": "192.0.0.2",
                "option-data": [ ],
                "server-hostname": "server-hostname.example.org"
            },
            ...
            {
                "boot-file-name": "bootfile.efi",
                "client-classes": [ ],
                "hostname": "otherhost.example.org",
                "hw-address": "01:02:03:04:05:ff",
                "ip-address": "192.0.2.200",
                "next-server": "192.0.0.2",
                "option-data": [ ],
                "server-hostname": "server-hostname.example.org"
            }
        ]
    },
    "result": 0,
    "text": "72 IPv4 host(s) found."
}
```

The response returned by `reservation-get-all` can be very long. The DHCP server does not handle DHCP traffic when preparing a response to reservation-get-all, so if there are many reservations in a subnet, this may be disruptive. Use with caution. For larger deployments, please consider using `reservation-get-page` instead (see *The reservation-get-page command*).

For a reference, see *The reservation-get-all Command*.

16.9.5 The reservation-get-page command

`reservation-get-page` can be used to query the host database and retrieve all reservations in a specified subnet by pages. This command uses parameters providing the mandatory subnet-id. Use a value of zero (0) to fetch global reservations. The second mandatory parameter is the page size limit. Optional source-index and from host id, both defaulting to 0, are used to chain page queries. Since Kea version 1.9.0 the subnet id parameter is optional.

The usage of from and source-index parameters requires additional explanation. For the first call, those parameters should not be specified (or specified as zeros). For any follow-up calls, they should be set to the values returned in previous calls in a next map holding from and source-index values. Subsequent calls should be issued until all reservations are returned. The end is reached once the returned list is empty, the count is 0, no next map is present, and result status 3 (empty) is returned.

Note: The from and source-index parameters are reflecting the internal state of the search. There is no need to understand what they represent; it is simply a value that is supposed to be copied from one response to the next query. However, for those who are curious, the from field represents a 64-bit representation of the host identifier used by a host backend. The source-index is an internal representation of multiple host backends: 0 is used to represent hosts defined in a configuration file, and 1 represents the first database backend. In some uncommon cases there may be more than one database backend configured, so potentially there may be a 2. In any case, Kea will iterate over all backends configured.

For instance, retrieving host reservations for the subnet 1 and requesting the first page can be done by:

```
{
    "command": "reservation-get-page",
    "arguments": {
        "subnet-id": 1,
        "limit": 10
     }
}
```

Since this is the first call, source-index and from should not be specified. They will default to their zero default values.

Some hosts are returned with information to get the next page:

```
{
    "arguments": {
        "count": 72,
        "hosts": [
            {
                "boot-file-name": "bootfile.efi",
                "client-classes": [ ],
                "hostname": "somehost.example.org",
                "hw-address": "01:02:03:04:05:06",
                "ip-address": "192.0.2.100",
                "next-server": "192.0.0.2",
                "option-data": [ ],
                "server-hostname": "server-hostname.example.org"
            },
            ...
            {
                "boot-file-name": "bootfile.efi",
                "client-classes": [ ],
                "hostname": "otherhost.example.org",
                "hw-address": "01:02:03:04:05:ff",
                "ip-address": "192.0.2.200",
                "next-server": "192.0.0.2",
                "option-data": [ ],
                "server-hostname": "server-hostname.example.org"
            }
        ],
        "next": {
            "from": 1234567,
            "source-index": 1
        }
    },
    "result": 0,
    "text": "72 IPv4 host(s) found."
}
```

Note that the "from" and "source-index" fields were specified in the response in the next map. Those two must be copied to the next command, so Kea continues from the place where the last command finished. To get the next page the following command can be sent:

```
{
    "command": "reservation-get-page",
    "arguments": {
        "subnet-id": 1,
        "source-index": 1,
        "from": 1234567,
        "limit": 10
     }
}
```

The response will contain a list of hosts with updated source-index and from fields. Continue calling the command until the last page is received. Its response will look like this:

```
{
    "arguments": {
        "count": 0,
        "hosts": [ ],
    },
    "result": 3,
    "0 IPv4 host(s) found."
}
```

This command is more complex than `reservation-get-all`, but lets users retrieve larger host reservations lists in smaller chunks. For small deployments with few reservations, it is easier to use `reservation-get-all` (see *The reservation-get-all Command*).

Note: Currently `reservation-get-page` is not supported by the Cassandra host backend.

16.9.6 The reservation-get-by-hostname Command

`reservation-get-by-hostname` can be used to query the host database and retrieve all reservations with a specified hostname and optionally in a specified subnet. This command uses parameters providing the mandatory hostname and the optional subnet-id. Global host reservations can be retrieved by using a subnet-id value of zero (0). Hostname matching is case-insensitive. This command is available since Kea version 1.7.1.

For instance, retrieving host reservations for "foobar" in the subnet 1:

```
{
    "command": "reservation-get-by-hostname",
    "arguments": {
        "hostname": "foobar.example.org",
        "subnet-id": 1
     }
}
```

returns some IPv4 hosts:

```
{
    "arguments": {
        "hosts": [
            {
                "boot-file-name": "bootfile.efi",
                "client-classes": [ ],
                "hostname": "foobar.example.org",
                "hw-address": "01:02:03:04:05:06",
                "ip-address": "192.0.2.100",
                "next-server": "192.0.0.2",
                "option-data": [ ],
                "server-hostname": "server-hostname.example.org"
            },
            ...
            {
                "boot-file-name": "bootfile.efi",
                "client-classes": [ ],
                "hostname": "foobar.example.org",
                "hw-address": "01:02:03:04:05:ff",
                "ip-address": "192.0.2.200",
                "next-server": "192.0.0.2",
                "option-data": [ ],
                "server-hostname": "server-hostname.example.org"
            }
        ]
    },
    "result": 0,
    "text": "5 IPv4 host(s) found."
}
```

The response returned by `reservation-get-by-hostname` can be long in particular when responses are not limited to a subnet.

For a reference, see *The reservation-get-by-hostname Command*.

Note: When the host backend is MySQL this commands relies on the fact the hostname column in the hosts table uses a case-insensitive collation as explained in the *MySQL* section of *Kea Database Administration*.

16.9.7 The reservation-get-by-id Command

`reservation-get-by-id` can be used to query the host database and retrieve all reservations with a specified identifier (identifier-type and identifier parameters) independently of subnets. The syntax for parameters is the same as for ref:*command-reservation-get*. The subnet-id parameter is forbidden to avoid confusion. This command is available since Kea version 1.9.0.

For instance, retrieving host reservations for the 01:02:03:04:05:06 MAC address:

```
{
    "command": "reservation-get-by-id",
    "arguments": {
        "identifier-type": "hw-address",
        "identifier": "01:02:03:04:05:06"
     }
 }
```

returns some IPv4 hosts:

```
{
    "arguments": {
        "hosts": [
            {
                "boot-file-name": "bootfile.efi",
                "client-classes": [ ],
                "hostname": "foo.example.org",
                "hw-address": "01:02:03:04:05:06",
                "ip-address": "192.0.2.100",
                "next-server": "192.0.0.2",
                "option-data": [ ],
                "server-hostname": "server-hostname.example.org",
                "subnet-id": 123
            },
            ...
            {
                "boot-file-name": "bootfile.efi",
                "client-classes": [ ],
                "hostname": "bar.example.org",
                "hw-address": "01:02:03:04:05:06",
                "ip-address": "192.0.2.200",
                "next-server": "192.0.0.2",
                "option-data": [ ],
                "server-hostname": "server-hostname.example.org",
                "subnet-id": 345
            }
        ]
    },
    "result": 0,
    "text": "5 IPv4 host(s) found."
}
```

The response returned by `reservation-get-by-id` can be long in particular when responses are not limited to a subnet.

For a reference, see *The reservation-get-by-id Command*.

16.9.8 The reservation-del Command

`reservation-del` can be used to delete a reservation from the host database. There are two types of parameters this command supports: (subnet-id, address) or (subnet-id, identifier-type, identifier). The first type of query is used when the address (either IPv4 or IPv6) is known, but the details of the reservation are not. One common use case of this type of query is to remove a reservation (e.g. a specific address should no longer be reserved). The second query uses identifiers. For maximum flexibility, Kea stores the host identifying information as a pair of values: the type and the actual identifier. Currently supported identifiers are “hw-address”, “duid”, “circuit-id”, “client-id”, and “flex-id”, but additional types may be added in the future. If any new identifier types are defined in the future, the reservation-get command will support them automatically. The `subnet-id` is mandatory. Use a value of zero (0) to delete a global reservation, or the id of the subnet from which the reservation should be deleted.

An example command for deleting a host reservation by (subnet-id, address) pair looks as follows:

```
{
    "command": "reservation-del",
    "arguments": {
        "subnet-id": 1,
        "ip-address": "192.0.2.202"
    }
}
```

An example deletion by (subnet-id, identifier-type, identifier) looks as follows:

```
{
    "command": "reservation-del",
    "arguments":
        "subnet-id": 4,
        "identifier-type": "hw-address",
        "identifier": "01:02:03:04:05:06"
    }
}
```

`reservation-del` returns a result 0 when the host deletion was successful or 1 if it was not. Descriptive text is provided in the event of an error. Example results look as follows:

```
{
    "result": 1,
    "text": "Host not deleted (not found)."
}
```

```
{
    "result": 0,
    "text": "Host deleted."
}
```

```
{
    "result": 1,
    "text": "Unable to delete a host because there is no hosts-database
             configured."
}
```

16.10 lease_cmds: Lease Commands

This section describes the hook library with commands used to manage leases. Kea provides a way to store lease information in several backends (memfile, MySQL, PostgreSQL, and Cassandra), and this library provides an interface that can manipulate leases in a unified, safe way. In particular, it allows things previously impossible: lease manipulation in memfile while Kea is running, sanity check changes, lease existence checks, and removal of all leases belonging to a specific subnet. The hook library can also catch more obscure errors, like an attempt to add a lease with a subnet-id that does not exist in the configuration, or configuring a lease to use an address that is outside of the subnet to which it is supposed to belong. The library also provides a non-programmatic way to manage user contexts associated with leases.

Note: This library may only be loaded by the `kea-dhcp4` or the `kea-dhcp6` process.

There are many use cases where an administrative command may be useful; for example, during migration between servers or different vendors, when a certain network is being retired, or when a device has been disconnected and the system administrator knows that it will not be coming back. The "get" queries may be useful for automating certain management and monitoring tasks. They can also act as preparatory steps for lease updates and removals.

This library provides the following commands:

- `lease4-add` - adds a new IPv4 lease.
- `lease6-add` - adds a new IPv6 lease.
- `lease6-bulk-apply` - creates, updates and/or deletes multiple IPv6 leases in a single transaction.
- `lease4-get` - checks whether an IPv4 lease with the specified parameters exists and returns it if it does.
- `lease6-get` - checks whether an IPv6 lease with the specified parameters exists and returns it if it does.
- `lease4-get-all` - returns all IPv4 leases or all IPv4 leases for the specified subnets.
- `lease6-get-all` - returns all IPv6 leases or all IPv6 leases for the specified subnets.
- `lease4-get-page` - returns a set ("page") of leases from the list of all IPv4 leases in the database. By iterating through the pages it is possible to retrieve all the leases.
- `lease6-get-page` - returns a set ("page") of leases from the list of all IPv6 leases in the database. By iterating through the pages it is possible to retrieve all the leases.
- `lease4-get-by-hw-address` - return all IPv4 leases with the specified hardware address.
- `lease4-get-by-client-id` - return all IPv4 leases with the specified client id.
- `lease6-get-by-duid` - returns all IPv6 leases with the specified DUID.
- `lease4-get-by-hostname` - return all IPv4 leases with the specified hostname.
- `lease6-get-by-hostname` - return all IPv6 leases with the specified hostname.
- `lease4-del` - deletes an IPv4 lease with the specified parameters.
- `lease6-del` - deletes an IPv6 lease with the specified parameters.
- `lease4-update` - updates an IPv4 lease.
- `lease6-update` - updates an IPv6 lease.
- `lease4-wipe` - removes all leases from a specific IPv4 subnet or from all subnets.
- `lease6-wipe` - removes all leases from a specific IPv6 subnet or from all subnets.
- `lease4-resend-ddns` - resend a request to update DNS entries for an existing lease.
- `lease6-resend-ddns` - resend a request to update DNS entries for an existing lease.

The lease commands library is part of the open source code and is available to every Kea user.

All commands use JSON syntax and can be issued either using the control channel (see *Management API*) or Control Agent (see *The Kea Control Agent*).

The library can be loaded in the same way as other hook libraries, and it does not take any parameters. It supports both DHCPv4 and DHCPv6 servers.

```
"Dhcp6": {
    "hooks-libraries": [
        {
            "library": "/path/libdhcp_lease_cmds.so"
        }
        ...
    ]
}
```

16.10.1 The lease4-add, lease6-add Commands

The `lease4-add` and `lease6-add` commands allow for the creation of a new lease. Typically Kea creates a lease when it first sees a new device; however, sometimes it may be convenient to create the lease manually. The `lease4-add` command requires at least two parameters: an IPv4 address and an identifier, i.e. hardware (MAC) address. A third parameter, subnet-id, is optional. If the subnet-id is not specified or the specified value is 0, Kea will try to determine the value by running a subnet-selection procedure. If specified, however, its value must match the existing subnet. The simplest successful call might look as follows:

```
{
    "command": "lease4-add",
    "arguments": {
        "ip-address": "192.0.2.202",
        "hw-address": "1a:1b:1c:1d:1e:1f"
    }
}
```

The `lease6-add` command requires three parameters: an IPv6 address, an IAID value (identity association identifier, a value sent by clients), and a DUID. As with lease4-add, the subnet-id parameter is optional. If the subnet-id is not specified or the provided value is 0, Kea will try to determine the value by running a subnet-selection procedure. If specified, however, its value must match the existing subnet. For example:

```
{
    "command": "lease6-add",
    "arguments": {
        "subnet-id": 66,
        "ip-address": "2001:db8::3",
        "duid": "1a:1b:1c:1d:1e:1f:20:21:22:23:24",
        "iaid": 1234
    }
}
```

`lease6-add` can also be used to add leases for IPv6 prefixes. In this case there are three additional parameters that must be specified: subnet-id, type (set to value of “IA_PD”), and prefix length. The actual prefix is set using the ip-address field. Note that Kea cannot guess subnet-id values for prefixes; they must be specified explicitly. For example, to configure a lease for prefix 2001:db8:abcd::/48, the following command can be used:

```
{
    "command": "lease6-add",
    "arguments": {
        "subnet-id": 66,
        "type": "IA_PD",
        "ip-address": "2001:db8:abcd::",
        "prefix-len": 48,
        "duid": "1a:1b:1c:1d:1e:1f:20:21:22:23:24",
        "iaid": 1234
    }
}
```

The commands can take several additional optional parameters:

- `valid-lft` - specifies the lifetime of the lease, expressed in seconds. If not specified, the value configured in the subnet related to the specified subnet-id is used.
- `expire` - creates a timestamp of the lease expiration time, expressed in UNIX format (seconds since 1 Jan 1970). If not specified, the default value is now + the lease lifetime (the value of valid-lft).
- `fqdn-fwd` - specifies whether the lease should be marked as if a forward DNS update were conducted. Note

this only affects the data stored in the lease database, and no DNS update will be performed. If configured, a DNS update to remove the A or AAAA records will be conducted when the lease is removed due to expiration or being released by a client. If not specified, the default value is false. The hostname parameter must be specified if fqdn-fwd is set to true.

- `fqdn-rev` - specifies whether the lease should be marked as if reverse DNS update were conducted. Note this only affects the the data stored in the lease database, and no DNS update will be performed.. If configured, a DNS update to remove the PTR record will be conducted when the lease is removed due to expiration or being released by a client. If not specified, the default value is false. The hostname parameter must be specified if fqdn-fwd is set to true.
- `hostname` - specifies the hostname to be associated with this lease. Its value must be non-empty if either fqdn-fwd or fwdn-rev are set to true. If not specified, the default value is an empty string.
- `hw-address` - optionally specifies a hardware (MAC) address for an IPv6 lease. It is a mandatory parameter for an IPv4 lease.
- `client-id` - optionally specifies a client identifier for an IPv4 lease.
- `preferred-lft` - optionally specifies a preferred lifetime for IPv6 leases. If not specified, the value configured for the subnet corresponding to the specified subnet-id is used. This parameter is not used when adding an IPv4 lease.
- `state` - specify the state of added lease, can be 0 for `default`, 1 for `declined` and 2 for `expired-reclaimed` state. Any other value will cause error. Note that using 1 for a “IA_PD” lease type is illegal and will be rejected.
- `user-context` - specifies the user context to be associated with this lease. It must be a JSON map.

Here is an example of a more complex lease addition:

```
{
    "command": "lease6-add",
    "arguments": {
        "subnet-id": 66,
        "ip-address": "2001:db8::3",
        "duid": "01:02:03:04:05:06:07:08",
        "iaid": 1234,
        "hw-address": "1a:1b:1c:1d:1e:1f",
        "preferred-lft": 500,
        "valid-lft": 1000,
        "expire": 12345678,
        "fqdn-fwd": true,
        "fqdn-rev": true,
        "state": 0,
        "hostname": "urania.example.org",
        "user-context": { "version": 1 }
    }
}
```

The command returns a status that indicates either success (result 0) or failure (result 1). A failed command always includes a text parameter that explains the cause of failure. For example:

```
{ "result": 0, "text": "Lease added." }
```

Example failure:

```
{ "result": 1, "text": "missing parameter 'ip-address' (<string>:3:19)" }
```

16.10.2 The lease6-bulk-apply Command

The `lease6-bulk-apply` was implemented to address the performance penalty in the High Availability when a single DHCPv6 transaction resulted in multiple lease updates sent to the partner if multiple address and/or prefix leases were allocated. Consider the case when a DHCPv6 client requests the assignment of two IPv6 addresses and two IPv6 prefixes. That may result in allocation of 4 leases. In addition, the DHCPv6 may assign different address than requested by the client during the renew or rebind and delete the leases previously used by this client. The are 6 of lease changes sent between the HA partners is in this case. Sending these updates in individual commands, e.g. `lease6-update` is highly inefficient and produces unnecessary delays in communication between the HA partners and in sending the response to the DHCPv6 client.

The `lease6-bulk-apply` command deals with this problem by aggregating all lease changes in a single command. Both deleted leases and new/updated leases are conveyed in a single command. The receiving server iterates over the deleted leases and deletes them from its lease database. Next, it iterates over the new/updated leases and adds them to the database or updates them if they already exist.

Even though the High Avialability is the major application for this command, it can be freely used in all cases when it is desired to send multiple lease changes in a single command.

In the following example, we ask to delete two leases and to add or update two other leases in the database:

```
 {
   "command": "lease6-bulk-apply",
   "arguments": {
       "deleted-leases": [
           {
               "ip-address": "2001:db8:abcd::",
               "type": "IA_PD",
               ...
           },
           {
               "ip-address": "2001:db8:abcd::234",
               "type": "IA_NA",
               ...
           }
       ],
       "leases": [
           {
               "subnet-id": 66,
               "ip-address": "2001:db8:cafe::",
               "type": "IA_PD",
                ...
           },
           {
               "subnet-id": 66,
               "ip-address": "2001:db8:abcd::333",
               "type": "IA_NA",
               ...
           }
       ]
    }
}
```

If any of the leases is malformed, no leases changes are applied to the lease database. If the leases are well formed but there is a failure to apply any of the lease changes to the database, the command will continue to be processed for other leases. All the leases for which the command was unable to apply the changes in the database will be listed in the response. For example:

```
{
    "result": 0,
    "text": "Bulk apply of 2 IPv6 leases completed".
    "arguments": {
        "failed-deleted-leases": [
            {
                "ip-address": "2001:db8:abcd::",
                "type": "IA_PD",
                "result": 3,
                "error-message": "no lease found"
            }
        ],
        "failed-leases": [
            {
                "ip-address": "2001:db8:cafe::",
                "type": "IA_PD",
                "result": 1,
                "error-message": "unable to communicate with the lease database"
            }
        ]
    }
}
```

The response above indicates that the hooks library was unable to delete the lease for prefix "2001:db8:abcd::" and add or update the lease for prefix "2001:db8:cafe::". However, there are two other lease changes which have been applied as indicated by the text message. The `result` is the status constant that indicates the type of the error experienced for the particular lease. The meaning of the returned codes are the same as the results returned for the commands. In particular, the result of 1 indicates an error while processing the lease, e.g. a communication error with the database. The result of 3 indicates that an attempt to delete the lease was unsuccessful because such lease doesn't exist (empty result).

16.10.3 The lease4-get, lease6-get Commands

`lease4-get` or `lease6-get` can be used to query the lease database and retrieve existing leases. There are two types of parameters the `lease4-get` command supports: (address) or (subnet-id, identifier-type, identifier). There are also two types for `lease6-get`: (address, type) or (subnet-id, identifier-type, identifier, IAID, type). The first type of query is used when the address (either IPv4 or IPv6) is known, but the details of the lease are not; one common use case of this type of query is to find out whether a given address is being used. The second query uses identifiers; currently supported identifiers for leases are: "hw-address" (IPv4 only), "client-id" (IPv4 only), and "duid" (IPv6 only).

An example `lease4-get` command for getting a lease using an IPv4 address is:

```
{
    "command": "lease4-get",
    "arguments": {
        "ip-address": "192.0.2.1"
    }
}
```

An example of the `lease6-get` query is:

```
{
  "command": "lease6-get",
  "arguments": {
    "ip-address": "2001:db8:1234:ab::",
    "type": "IA_PD"
  }
}
```

An example query by "hw-address" for an IPv4 lease looks as follows:

```
{
    "command": "lease4-get",
    "arguments": {
        "identifier-type": "hw-address",
        "identifier": "08:08:08:08:08:08",
        "subnet-id": 44
    }
}
```

An example query by "client-id" for an IPv4 lease looks as follows:

```
{
    "command": "lease4-get",
    "arguments": {
        "identifier-type": "client-id",
        "identifier": "01:01:02:03:04:05:06",
        "subnet-id": 44
    }
}
```

An example query by (subnet-id, identifier-type, identifier, iaid, type) for an IPv6 lease is:

```
{
    "command": "lease4-get",
    "arguments": {
        "identifier-type": "duid",
        "identifier": "08:08:08:08:08:08",
        "iaid": 1234567,
        "type": "IA_NA",
        "subnet-id": 44
    }
}
```

The type is an optional parameter. Supported values are: IA_NA (non-temporary address) and IA_PD (IPv6 prefix). If not specified, IA_NA is assumed.

`leaseX-get` returns a result that indicates a result of the operation and lease details, if found. It has one of the following values: 0 (success), 1 (error), or 3 (empty). An empty result means that a query has been completed properly, but the object (a lease in this case) has not been found. The lease parameters, if found, are returned as arguments.

An example result returned when the host was found:

```
{
  "arguments": {
    "client-id": "42:42:42:42:42:42:42:42",
    "cltt": 12345678,
    "fqdn-fwd": false,
    "fqdn-rev": true,
    "hostname": "myhost.example.com.",
    "hw-address": "08:08:08:08:08:08",
    "ip-address": "192.0.2.1",
    "state": 0,
    "subnet-id": 44,
    "valid-lft": 3600
  },
  "result": 0,
  "text": "IPv4 lease found."
}
```

16.10.4 The lease4-get-all, lease6-get-all Commands

`lease4-get-all` and `lease6-get-all` are used to retrieve all IPv4 or IPv6 leases, or all leases for the specified set of subnets. All leases are returned when there are no arguments specified with the command, as in the following example:

```
{
    "command": "lease4-get-all"
}
```

If arguments are provided, it is expected that they contain the “subnets” parameter, which is a list of subnet identifiers for which the leases should be returned. For example, in order to retrieve all IPv6 leases belonging to the subnets with identifiers 1, 2, 3, and 4:

```
{
    "command": "lease6-get-all",
    "arguments": {
        "subnets": [ 1, 2, 3, 4 ]
    }
}
```

The returned response contains a detailed list of leases in the following format:

```
{
    "arguments": {
        "leases": [
            {
                "cltt": 12345678,
                "duid": "42:42:42:42:42:42:42:42",
                "fqdn-fwd": false,
                "fqdn-rev": true,
                "hostname": "myhost.example.com.",
                "hw-address": "08:08:08:08:08:08",
                "iaid": 1,
                "ip-address": "2001:db8:2::1",
                "preferred-lft": 500,
                "state": 0,
                "subnet-id": 44,
                "type": "IA_NA",
                "valid-lft": 3600
            },
            {
                "cltt": 12345678,
                "duid": "21:21:21:21:21:21:21:21",
                "fqdn-fwd": false,
                "fqdn-rev": true,
                "hostname": "",
                "iaid": 1,
                "ip-address": "2001:db8:0:0:2::",
                "preferred-lft": 500,
                "prefix-len": 80,
                "state": 0,
                "subnet-id": 44,
                "type": "IA_PD",
                "valid-lft": 3600
            }
        ]
    },
    "result": 0,
    "text": "2 IPv6 lease(s) found."
}
```

> **Warning:** The `lease4-get-all` and `lease6-get-all` commands may result in very large responses. This may have a negative impact on the DHCP server's responsiveness while the response is generated and transmitted over the control channel, as the server imposes no restriction on the number of leases returned as a result of this command.

16.10.5 The lease4-get-page, lease6-get-page Commands

The `lease4-get-all` and `lease6-get-all` commands may result in very large responses; generating such a response may consume CPU bandwidth as well as memory. It may even cause the server to become unresponsive. In case of large lease databases it is usually better to retrieve leases in chunks, using the paging mechanism. `lease4-get-page` and `lease6-get-page` implement a paging mechanism for DHCPv4 and DHCPv6 servers respectively. The following command retrieves the first 1024 IPv4 leases:

```
{
    "command": "lease4-get-page",
    "arguments": {
        "from": "start",
        "limit": 1024
    }
}
```

The keyword `start` denotes that the first page of leases should be retrieved. Alternatively, an IPv4 zero address can be specified to retrieve the first page:

```
{
    "command": "lease4-get-page",
    "arguments": {
        "from": "0.0.0.0",
        "limit": 1024
    }
}
```

Similarly, the IPv6 zero address can be specified in the `lease6-get-page` command:

```
{
    "command": "lease6-get-page",
    "arguments": {
        "from": "::",
        "limit": 6
    }
}
```

The response has the following structure:

```
{
    "arguments": {
        "leases": [
            {
                "ip-address": "2001:db8:2::1",
                ...
            },
            {
                "ip-address": "2001:db8:2::9",
                ...
            },
            {
                "ip-address": "2001:db8:3::1",
                ...
            },
            {
                "ip-address": "2001:db8:5::3",
                ...
            }
            {
                "ip-address": "2001:db8:4::1",
                ...
            },
            {
                "ip-address": "2001:db8:2::7",
                ...
            }

        ],
        "count": 6
    },
    "result": 0,
    "text": "6 IPv6 lease(s) found."
}
```

Note that the leases' details were excluded from the response above for brevity.

Generally, the returned list is not sorted in any particular order. Some lease database backends may sort leases in ascending order of addresses, but the controlling client must not rely on this behavior. In cases of highly distributed databases, such as Cassandra, ordering may be inefficient or even impossible.

The `count` parameter contains the number of returned leases on the page.

To fetch the next page, the client must use the last address of the current page as an input to the next `lease4-get-page` or `lease6-get-page` command call. In this example it is:

```
{
    "command": "lease6-get-page",
    "arguments": {
        "from": "2001:db8:2::7",
        "count": 6
    }
}
```

because 2001:db8:2::7 is the last address on the current page.

The client may assume that it has reached the last page when the `count` value is lower than that specified in the command; this includes the case when the `count` is equal to 0, meaning that no leases were found.

16.10.6 The lease4-get-by-*, lease6-get-by-* Commands

`leaseX-get-by-Y` can be used to query the lease database and retrieve all existing leases with a specified hardware address (IPv4 only), client id IPv4 only), duid (IPv6 only) identifiers or hostname.

An example `lease4-get-by-hw-address` command for getting IPv4 leases with a given hardware address is:

```
{
    "command": "lease4-get-by-hw-address",
    "arguments": {
        "hw-address": "08:08:08:08:08:08"
    }
}
```

An example of the `lease6-get-by-hostname` is:

```
{
    "command": "lease6-get-by-hostname",
    "arguments": {
        "hostname": "myhost.example.org"
    }
}
```

The by key is the only parameter. The returned response contains a detailed list of leases in the same format than `leaseX-get-all`. This list can be empty and usually is never large.

16.10.7 The lease4-del, lease6-del Commands

`leaseX-del` can be used to delete a lease from the lease database. There are two types of parameters this command supports, similar to the leaseX-get commands: (address) for both v4 and v6, (subnet-id, identifier-type, identifier) for v4, and (subnet-id, identifier-type, identifier, type, IAID) for v6. The first type of query is used when the address (either IPv4 or IPv6) is known, but the details of the lease are not. One common use case is where an administrator wants a specified address to no longer be used. The second form of the command uses identifiers. For maximum flexibility, this interface uses identifiers as a pair of values: the type and the actual identifier. The currently supported identifiers are “hw-address” (IPv4 only), “client-id” (IPv4 only), and “duid” (IPv6 only).

An example command for deleting a lease by address is

```
{
    "command": "lease4-del",
    "arguments": {
        "ip-address": "192.0.2.202"
    }
}
```

An example IPv4 lease deletion by “hw-address” is:

```
{
  "command": "lease4-del",
  "arguments": {
    "identifier": "08:08:08:08:08:08",
    "identifier-type": "hw-address",
    "subnet-id": 44
  }
}
```

As of Kea 1.7.10, a new parameter, `update-ddns`, is supported (IPv4 and IPv6). When true it instructs the server to queue a request to kea-dhcp-ddns to remove DNS entries after the lease is successfully deleted if:

- DDNS updating is enabled. (i.e. “dhcp-ddns”:{ “enable-updates”: true })
- The lease’s hostname is not be empty.
- At least one of the lease’s DNS direction flags (fdqn_fwd or fdqn_rev) is true.

This parameter defaults to false. An example of its use is shown below:

```
{
    "command": "lease4-del",
    "arguments": {
        "ip-address": "192.0.2.202",
        "update-ddns": true
    }
}
```

`leaseX-del` returns a result that indicates the outcome of the operation. It has one of the following values: 0 (success), 1 (error), or 3 (empty). The empty result means that a query has been completed properly, but the object (a lease in this case) has not been found.

16.10.8 The lease4-update, lease6-update Commands

The `lease4-update` and `lease6-update` commands can be used to update existing leases. Since all lease database backends are indexed by IP addresses, it is not possible to update an address, but all other fields may be altered. If an address needs to be changed, please use `leaseX-del` followed by `leaseX-add`.

The subnet-id parameter is optional. If not specified, or if the specified value is 0, Kea will try to determine its value by running a subnet-selection procedure. If specified, however, its value must match the existing subnet.

The optional boolean parameter “force-create” specifies whether the lease should be created if it does not exist in the database. It defaults to false, which indicates that the lease is not created if it does not exist. In such a case, an error is returned as a result of trying to update a non-existing lease. If the “force-create” parameter is set to true and the updated lease does not exist, the new lease is created as a result of receiving the `leaseX-update`.

An example of a command to update an IPv4 lease is:

```
{
  "command": "lease4-update",
  "arguments": {
    "ip-address": "192.0.2.1",
    "hostname": "newhostname.example.org",
    "hw-address": "1a:1b:1c:1d:1e:1f",
    "subnet-id": 44,
    "force-create": true
  }
}
```

An example of a command to update an IPv6 lease is:

```
{
  "command": "lease6-update",
  "arguments": {
    "ip-address": "2001:db8::1",
    "duid": "88:88:88:88:88:88:88:88",
    "iaid": 7654321,
    "hostname": "newhostname.example.org",
    "subnet-id": 66,
    "force-create": false
  }
}
```

16.10.9 The lease4-wipe, lease6-wipe Commands

`lease4-wipe` and `lease6-wipe` are designed to remove all leases associated with a given subnet. This administrative task is expected to be used when an existing subnet is being retired. Note that the leases are not properly expired; no DNS updates are carried out, no log messages are created, and hooks are not called for the leases being removed.

An example of `lease4-wipe` is:

```
{
  "command": "lease4-wipe",
  "arguments": {
    "subnet-id": 44
  }
}
```

An example of `lease6-wipe` is:

```
{
  "command": "lease6-wipe",
  "arguments": {
    "subnet-id": 66
  }
}
```

The commands return a text description of the number of leases removed, plus the status code 0 (success) if any leases were removed or 3 (empty) if there were no leases. Status code 1 (error) may be returned if the parameters are incorrect or some other exception is encountered.

Subnet-id 0 has a special meaning; it tells Kea to delete leases from all configured subnets. Also, the subnet-id parameter may be omitted. If not specified, leases from all subnets are wiped.

Note: not all backends support this command.

16.10.10 The lease4-resend-ddns, lease6-resend-ddns Commands

`lease4-resend-ddns` or `lease6-resend-ddns` can be used to generate a request to kea-dhcp-ddns to update the DNS entries for an existing lease. The desired lease is selected by a single parameter, "ip-address". In order for an update request to be generated, DDNS updating must be enabled and DNS entries must have already been made (or attempted) for the lease. In other words all of the following must be true:

- DDNS updating must be enabled. (i.e. "dhcp-ddns":{ "enable-updates": true"})
- The lease's hostname must not be empty.

- At least one of the lease's DNS direction flags (fdqn_fwd or fdqn_rev) must be true.

An example `lease4-resend-ddns` command for getting a lease using an IPv4 address is:

```
{
    "command": "lease4-resend-ddns",
    "arguments": {
        "ip-address": "192.0.2.1"
    }
}
```

An example of the `lease6-resend-ddns` query is:

```
{
  "command": "lease6-resend-ddns",
  "arguments": {
    "ip-address": "2001:db8:1::1"
  }
}
```

`leaseX-resend-ddns` returns a result that indicates a result of the operation. It has one of the following values: 0 (success), 1 (error), or 3 (empty). An empty result means that a query has been completed properly, but the object (a lease in this case) has not been found.

A successful result does not mean that DNS has been successfully updated. It indicates that a request to update DNS has been successfully created and queued for transmission to kea-dhcp-ddns.

An example result returned when the lease was found:

```
{
  "result": 0,
  "text": "NCR generated for: 2001:db8:1::1, hostname: example.com."
}
```

16.11 subnet_cmds: Subnet Commands

This section describes a hook application that offers some new commands used to query and manipulate subnet and shared network configurations in Kea. This application is very useful in deployments with a large number of subnets being managed by the DHCP servers, when those subnets are frequently updated. The commands offer a lightweight approach for manipulating subnets without a need to fully reconfigure the server and without affecting existing servers' configurations. An ability to manage shared networks (listing, retrieving details, adding new ones, removing existing ones, and adding subnets to and removing them from shared networks) is also provided.

Currently this library is only available to ISC customers with a paid support contract.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

The following commands are currently supported:

- `subnet4-list/subnet6-list` - lists all configured subnets.
- `subnet4-get/subnet6-get` - retrieves detailed information about a specified subnet.
- `subnet4-add/subnet6-add` - adds a new subnet into the server's configuration.
- `subnet4-update/subnet6-update` - updates a subnet in the server's configuration.

- `subnet4-del/subnet6-del` - removes a subnet from the server's configuration.
- `network4-list/network6-list` - lists all configured shared networks.
- `network4-get/network6-get` - retrieves detailed information about a specified shared network.
- `network4-add/network6-add` - adds a new shared network to the server's configuration.
- `network4-del/network6-del` - removes a shared network from the server's configuration.
- `network4-subnet-add/network6-subnet-add` - adds an existing subnet to an existing shared network.
- `network4-subnet-del/network6-subnet-del` - removes a subnet from an existing shared network and demotes it to a plain subnet.

16.11.1 The subnet4-list Command

This command is used to list all currently configured subnets. Each subnet is returned with a subnet identifier and subnet prefix. To retrieve detailed information about the subnet, use the `subnet4-get` command.

This command has the simple structure:

```
{
    "command": "subnet4-list"
}
```

The list of subnets is returned in the following format:

```
{
    "result": 0,
    "text": "2 IPv4 subnets found",
    "arguments": {
    "subnets": [
        {
            "id": 10,
            "subnet": "10.0.0.0/8"
        },
        {
            "id": 100,
            "subnet": "192.0.2.0/24"
        }
    ]
}
```

If no IPv4 subnets are found, an error code is returned along with the error description.

16.11.2 The subnet6-list Command

This command is used to list all currently configured subnets. Each subnet is returned with a subnet identifier and subnet prefix. To retrieve detailed information about the subnet, use the `subnet6-get` command.

This command has the simple structure:

```
{
    "command": "subnet6-list"
}
```

The list of subnets is returned in the following format:

```
{
    "result": 0,
    "text": "2 IPv6 subnets found",
    "arguments": {
    "subnets": [
        {
            "id": 11,
            "subnet": "2001:db8:1::/64"
        },
        {
            "id": 233,
            "subnet": "3000::/16"
        }
    ]
}
```

If no IPv6 subnets are found, an error code is returned along with the error description.

16.11.3 The subnet4-get Command

This command is used to retrieve detailed information about the specified subnet. This command usually follows `subnet4-list`, which is used to discover available subnets with their respective subnet identifiers and prefixes. Any of those parameters can be then used in `subnet4-get` to fetch subnet information:

```
{
    "command": "subnet4-get",
    "arguments": {
        "id": 10
    }
}
```

or

```
{
    "command": "subnet4-get",
    "arguments": {
        "subnet": "10.0.0.0/8"
    }
}
```

If the subnet exists the response will be similar to this:

```
{
    "result": 0,
    "text": "Info about IPv4 subnet 10.0.0.0/8 (id 10) returned",
    "arguments": {
        "subnets": [
            {
                "subnet": "10.0.0.0/8",
                "id": 1,
                "option-data": [
                    ....
                ]
                ...
            }
        ]
    }
}
```

16.11.4 The subnet6-get Command

This command is used to retrieve detailed information about the specified subnet. This command usually follows `subnet6-list`, which is used to discover available subnets with their respective subnet identifiers and prefixes. Any of those parameters can be then used in `subnet6-get` to fetch subnet information:

```
{
    "command": "subnet6-get",
    "arguments": {
        "id": 11
    }
}
```

or

```
{
    "command": "subnet6-get",
    "arguments": {
        "subnet": "2001:db8:1::/64"
    }
}
```

If the subnet exists the response will be similar to this:

```
{
    "result": 0,
    "text": "Info about IPv6 subnet 2001:db8:1::/64 (id 11) returned",
    "arguments": {
        "subnets": [
            {
                "subnet": "2001:db8:1::/64",
                "id": 1,
                "option-data": [
                    ...
                ]
                ....
            }
        ]
    }
}
```

16.11.5 The subnet4-add Command

This command is used to create and add a new subnet to the existing server configuration. This operation has no impact on other subnets. The subnet identifier must be specified and must be unique among all subnets. If the identifier or a subnet prefix is not unique, an error is reported and the subnet is not added.

The subnet information within this command has the same structure as the subnet information in the server configuration file, with the exception that static host reservations must not be specified within `subnet4-add`. The commands described in *host_cmds: Host Commands* should be used to add, remove, and modify static reservations.

```
{
    "command": "subnet4-add",
    "arguments": {
        "subnet4": [ {
            "id": 123,
            "subnet": "10.20.30.0/24",
            ...
        } ]
    }
}
```

The response to this command has the following structure:

```
{
    "result": 0,
    "text": "IPv4 subnet added",
    "arguments": {
        "subnet4": [
            {
                "id": 123,
                "subnet": "10.20.30.0/24"
            }
        ]
    }
}
```

16.11.6 The subnet6-add Command

This command is used to create and add a new subnet to the existing server configuration. This operation has no impact on other subnets. The subnet identifier must be specified and must be unique among all subnets. If the identifier or a subnet prefix is not unique, an error is reported and the subnet is not added.

The subnet information within this command has the same structure as the subnet information in the server configuration file, with the exception that static host reservations must not be specified within `subnet6-add`. The commands described in *host_cmds: Host Commands* should be used to add, remove, and modify static reservations.

```
{
    "command": "subnet6-add",
    "arguments": {
        "subnet6": [ {
            "id": 234,
            "subnet": "2001:db8:1::/64",
            ...
        } ]
    }
}
```

The response to this command has the following structure:

```
{
    "result": 0,
    "text": "IPv6 subnet added",
    "arguments": {
        "subnet6": [
            {
                "id": 234,
                "subnet": "2001:db8:1::/64"
            }
        ]
    }
}
```

It is recommended, but not mandatory, to specify the subnet ID. If not specified, Kea will try to assign the next subnet-id value. This automatic ID value generator is simple; it returns a previously automatically assigned value, increased by 1. This works well, unless a subnet is manually created with a value bigger than one previously used. For example, if subnet4-add is called five times, each without an ID, Kea will assign IDs 1, 2, 3, 4, and 5 and it will work just fine. However, if subnet4-add is called five times, with the first subnet having the subnet-id of value 3 and the remaining ones having no subnet-id, the operation will fail. The first command (with the explicit value) will use subnet-id 3; the second command will create a subnet with id of 1; the third will use a value of 2; and finally the fourth will have the subnet-id value auto-generated as 3. However, since there is already a subnet with that ID, the process will fail.

The general recommendation is either never use explicit values, so the auto-generated values will always work; or always use explicit values, so the auto-generation is never used. The two approaches can be mixed only if the administrator understands how internal automatic subnet-id generation works in Kea.

Note: Subnet IDs must be greater than zero and less than 4294967295.

16.11.7 The subnet4-update Command

This command is used to update a subnet in the existing server configuration. This operation has no impact on other subnets. The subnet identifier is used to identify the subnet to replace; it must be specified and must be unique among all subnets. The subnet prefix should not be updated.

The subnet information within this command has the same structure as the subnet information in the server configuration file, with the exception that static host reservations must not be specified within `subnet4-update`. The commands described in *host_cmds: Host Commands* should be used to update, remove, and modify static reservations.

```
{
    "command": "subnet4-update",
    "arguments": {
        "subnet4": [ {
            "id": 123,
            "subnet": "10.20.30.0/24",
            ...
        } ]
    }
}
```

The response to this command has the following structure:

```
{
    "result": 0,
    "text": "IPv4 subnet updated",
    "arguments": {
        "subnet4": [
            {
                "id": 123,
                "subnet": "10.20.30.0/24"
            }
        ]
    }
}
```

16.11.8 The subnet6-update Command

This command is used to update a subnet in the existing server configuration. This operation has no impact on other subnets. The subnet identifier is used to identify the subnet to replace; it must be specified and must be unique among all subnets. The subnet prefix should not be updated.

The subnet information within this command has the same structure as the subnet information in the server configuration file, with the exception that static host reservations must not be specified within `subnet6-update`. The commands described in *host_cmds: Host Commands* should be used to update, remove, and modify static reservations.

```
{
    "command": "subnet6-update",
    "arguments": {
        "subnet6": [ {
            "id": 234,
            "subnet": "2001:db8:1::/64",
            ...
        } ]
    }
}
```

The response to this command has the following structure:

```
{
    "result": 0,
    "text": "IPv6 subnet updated",
    "arguments": {
        "subnet6": [
            {
                "id": 234,
                "subnet": "2001:db8:1::/64"
            }
        ]
    }
}
```

16.11.9 The subnet4-del Command

This command is used to remove a subnet from the server's configuration. This command has no effect on other configured subnets, but removing a subnet has certain implications which the server's administrator should be aware of.

In most cases the server has assigned some leases to the clients belonging to the subnet. The server may also be configured with static host reservations which are associated with this subnet. The current implementation of the `subnet4-del` command removes neither the leases nor the host reservations associated with a subnet. This is the safest approach because the server does not lose track of leases assigned to the clients from this subnet. However, removal of the subnet may still cause configuration errors and conflicts. For example: after removal of the subnet, the server administrator may update a new subnet with the ID used previously for the removed subnet. This means that the existing leases and static reservations will be in conflict with this new subnet. Thus, we recommend that this command be used with extreme caution.

This command can also be used to completely delete an IPv4 subnet that is part of a shared network. To simply remove the subnet from a shared network and keep the subnet configuration, use the `network4-subnet-del` command instead.

The command has the following structure:

```
{
    "command": "subnet4-del",
    "arguments": {
        "id": 123
    }
}
```

The example successful response may look like this:

```
{
    "result": 0,
    "text": "IPv4 subnet 192.0.2.0/24 (id 123) deleted",
    "arguments": {
        "subnets": [
            {
                "id": 123,
                "subnet": "192.0.2.0/24"
            }
        ]
    }
}
```

16.11.10 The subnet6-del Command

This command is used to remove a subnet from the server's configuration. This command has no effect on other configured subnets, but removing a subnet has certain implications which the server's administrator should be aware of.

In most cases the server has assigned some leases to the clients belonging to the subnet. The server may also be configured with static host reservations which are associated with this subnet. The current implementation of the `subnet6-del` command removes neither the leases nor the host reservations associated with a subnet. This is the safest approach because the server does not lose track of leases assigned to the clients from this subnet. However, removal of the subnet may still cause configuration errors and conflicts. For example: after removal of the subnet, the server administrator may add a new subnet with the ID used previously for the removed subnet. This means that the existing leases and static reservations will be in conflict with this new subnet. Thus, we recommend that this command be used with extreme caution.

This command can also be used to completely delete an IPv6 subnet that is part of a shared network. To simply remove the subnet from a shared network and keep the subnet configuration, use the `network6-subnet-del` command instead.

The command has the following structure:

```
{
    "command": "subnet6-del",
    "arguments": {
        "id": 234
    }
}
```

The example successful response may look like this:

```
{
    "result": 0,
    "text": "IPv6 subnet 2001:db8:1::/64 (id 234) deleted",
    "subnets": [
        {
            "id": 234,
            "subnet": "2001:db8:1::/64"
        }
    ]
}
```

16.11.11 The network4-list, network6-list Commands

These commands are used to retrieve the full list of currently configured shared networks. The list contains only very basic information about each shared network. If more details are needed, please use `network4-get` or `network6-get` to retrieve all information available. This command does not require any parameters and its invocation is very simple:

```
{
    "command": "network4-list"
}
```

An example response for `network4-list` looks as follows:

```
{
    "arguments": {
        "shared-networks": [
            { "name": "floor1" },
            { "name": "office" }
        ]
    },
    "result": 0,
    "text": "2 IPv4 network(s) found"
}
```

`network6-list` follows exactly the same syntax for both the query and the response.

16.11.12 The network4-get, network6-get Commands

These commands are used to retrieve detailed information about shared networks, including subnets that are currently part of a given network. Both commands take one mandatory parameter, `name`, which specifies the name of the shared network. An example command to retrieve details about an IPv4 shared network with the name “floor13” looks as follows:

```
{
    "command": "network4-get",
    "arguments": {
        "name": "floor13"
    }
}
```

An example response could look as follows:

```
{
    "result": 0,
    "text": "Info about IPv4 shared network 'floor13' returned",
    "arguments": {
        "shared-networks": [
        {
            "match-client-id": true,
            "name": "floor13",
            "option-data": [ ],
            "rebind-timer": 90,
            "relay": {
                "ip-address": "0.0.0.0"
            },
            "renew-timer": 60,
            "reservation-mode": "all",
            "subnet4": [
                {
                    "subnet": "192.0.2.0/24",
                    "id": 5,
                    # many other subnet-specific details here
                },
                {
                    "id": 6,
                    "subnet": "192.0.3.0/31",
                    # many other subnet-specific details here
                }
            ],
            "valid-lifetime": 120
        }
        ]
    }
}
```

Note that the actual response contains many additional fields that are omitted here for clarity. The response format is exactly the same as used in `config-get`, just limited to returning the shared network's information.

16.11.13 The network4-add, network6-add Commands

These commands are used to add a new shared network, which must have a unique name. This command requires one parameter, `shared-networks`, which is a list and should contain exactly one entry that defines the network. The only mandatory element for a network is its name. Although it does not make operational sense, it is possible to add an empty shared network that does not have any subnets in it. That is allowed for testing purposes, but having empty networks (or with only one subnet) is discouraged in production environments. For details regarding syntax, see *Shared Networks in DHCPv4* and *Shared Networks in DHCPv6*.

Note: As opposed to parameter inheritance during the processing of a full new configuration, this command does not fully handle parameter inheritance. Any missing parameters will be filled with default values, rather than inherited from the global scope.

An example that showcases how to add a new IPv4 shared network looks as follows:

```
{
    "command": "network4-add",
    "arguments": {
        "shared-networks": [ {
            "name": "floor13",
            "subnet4": [
            {
                "id": 100,
                "pools": [ { "pool": "192.0.2.2-192.0.2.99" } ],
                "subnet": "192.0.2.0/24",
                "option-data": [
                    {
                        "name": "routers",
                        "data": "192.0.2.1"
                    }
                ]
            },
            {
                "id": 101,
                "pools": [ { "pool": "192.0.3.2-192.0.3.99" } ],
                "subnet": "192.0.3.0/24",
                "option-data": [
                    {
                        "name": "routers",
                        "data": "192.0.3.1"
                    }
                ]
            } ]
        } ]
    }
}
```

Assuming there was no shared network with a name "floor13" and no subnets with IDs 100 and 101 previously configured, the command will be successful and will return the following response:

```
{
    "arguments": {
        "shared-networks": [ { "name": "floor13" } ]
    },
    "result": 0,
    "text": "A new IPv4 shared network 'floor13' added"
}
```

The `network6-add` command uses the same syntax for both the query and the response. However, there are some parameters that are IPv4-only (e.g. match-client-id) and some that are IPv6-only (e.g. interface-id). The same applies to subnets within the network.

16.11.14 The network4-del, network6-del Commands

These commands are used to delete existing shared networks. Both commands take exactly one parameter, `name`, that specifies the name of the network to be removed. An example invocation of the `network4-del` command looks as follows:

```
{
    "command": "network4-del",
    "arguments": {
        "name": "floor13"
    }
}
```

Assuming there was such a network configured, the response will look similar to the following:

```
{
    "arguments": {
        "shared-networks": [
            {
                "name": "floor13"
            }
        ]
    },
    "result": 0,
    "text": "IPv4 shared network 'floor13' deleted"
}
```

The `network6-del` command uses exactly the same syntax for both the command and the response.

If there are any subnets belonging to the shared network being deleted, they will be demoted to a plain subnet. There is an optional parameter called `subnets-action` that, if specified, takes one of two possible values: `keep` (which is the default) and `delete`. It controls whether the subnets are demoted to plain subnets or removed. An example usage in the `network6-del` command that deletes the shared network and all subnets in it could look as follows:

```
{
    "command": "network4-del",
    "arguments": {
        "name": "floor13",
        "subnets-action": "delete"
    }
}
```

Alternatively, to completely remove the subnets, it is possible to use the `subnet4-del` or `subnet6-del` commands.

16.11.15 The network4-subnet-add, network6-subnet-add Commands

These commands are used to add existing subnets to existing shared networks. There are several ways to add a new shared network. The system administrator can add the whole shared network at once, either by editing a configuration file or by calling the `network4-add` or `network6-add` command with the desired subnets in it. This approach works better for completely new shared subnets. However, there may be cases when an existing subnet is running out of addresses and needs to be extended with additional address space; in other words, another subnet needs to be added on top of it. For this scenario, a system administrator can use `network4-add` or `network6-add`, and then add an existing subnet to this newly created shared network using `network4-subnet-add` or `network6-subnet-add`.

The `network4-subnet-add` and `network6-subnet-add` commands take two parameters: `id`, which is an integer and specifies the subnet-id of an existing subnet to be added to a shared network; and `name`, which specifies the name of the shared network to which the subnet will be added. The subnet must not belong to any existing network; to reassign a subnet from one shared network to another, please use the `network4-subnet-del` or `network6-subnet-del` commands first.

An example invocation of the `network4-subnet-add` command looks as follows:

```
{
    "command": "network4-subnet-add",
    "arguments": {
        "name": "floor13",
        "id": 5
    }
}
```

Assuming there is a network named "floor13", and there is a subnet with subnet-id 5 that is not a part of existing network, the command will return a response similar to the following:

```
{
    "result": 0,
    "text": "IPv4 subnet 10.0.0.0/8 (id 5) is now part of shared network 'floor13'"
}
```

The `network6-subnet-add` command uses exactly the same syntax for both the command and the response.

Note: As opposed to parameter inheritance during the processing of a full new configuration or when adding a new shared network with new subnets, this command does not fully handle parameter inheritance. Any missing parameters will be filled with default values, rather than inherited from the global scope or from the shared network.

16.11.16 The network4-subnet-del, network6-subnet-del Commands

These commands are used to remove a subnet that is part of an existing shared network and demote it to a plain, stand-alone subnet. To remove a subnet completely, use the `subnet4-del` or `subnet6-del` commands instead. The `network4-subnet-del` and `network6-subnet-del` commands take two parameters: `id`, which is an integer and specifies the subnet-id of an existing subnet to be removed from a shared network; and `name`, which specifies the name of the shared network from which the subnet will be removed.

An example invocation of the `network4-subnet-del` command looks as follows:

```
{
   "command": "network4-subnet-del",
   "arguments": {
       "name": "floor13",
       "id": 5
   }
}
```

Assuming there was a subnet with subnet-id equal to 5, that was part of a shared network named "floor13", the response would look similar to the following:

```
{
    "result": 0,
    "text": "IPv4 subnet 10.0.0.0/8 (id 5) is now removed from shared network 'floor13
↪'"
}
```

The `network6-subnet-del` command uses exactly the same syntax for both the command and the response.

16.12 BOOTP support

Note: This is library is still in experimental phase. Use with care!

This hooks library adds support for BOOTP with vendor information extensions (RFC 1497). Received BOOTP requests are recognized, translated into DHCPREQUEST packets by adding a dhcp-message-type option and put into the "BOOTP" client class. Members of this class get infinite lifetime leases but the class can be used too for instance to guard a pool of addresses.

The DHCP specific options, such as dhcp-message-type, are removed from the server's responses and responses shorter than the BOOTP minimum size (300 octets) are padded to this size.

The library is available since Kea 1.7.2 and can be loaded in a similar way as other hook libraries by the `kea-dhcp4` process. It takes no parameter.

```
"Dhcp4": {
    "hook_libraries": [
        {   "library": "/usr/local/lib/libdhcp_bootp.so" },
        ...
    ]
}
```

Note: This library is only meant to be loaded by the `kea-dhcp4` process as there is no BOOTP protocol for IPv6.

Note: A host reservation for a BOOTP client should use the hardware address as the identifier (the client-id option is a DHCP specific option).

Incoming BOOTP packets are added to the BOOTP class. This can be used to segregate BOOTP clients to separate pool. For example you can do the following:

```
"Dhcp4": {
    "client-classes": [
        {
            // The DHCP class is the complement of the BOOTP class
            "name": "DHCP",
            "test": "not member('BOOTP')"
        }
    ],
    "subnet4": [
        {
            "subnet": "192.0.2.0/24",
            "pools": [
            {
                // BOOTP clients will be handled here
                "pool": "192.0.2.200 - 192.0.2.254",
                "client-class": "BOOTP"
            },
            {
                // Regular DHCP clients will be handled here
                "pool": "192.0.2.1 - 192.0.2.199",
                "client-class": "DHCP"
            }],
            ...
        },
        ...
    ],
    ...
}
```

16.12.1 BOOTP Hooks Limitation

Currently the BOOTP library has the following limitation:

- A basic BOOTP as defined in RFC 951 is not supported. Kea only supports the BOOTP with vendor information extensions. Depending on the demand, this may or may not be implemented in the future.

16.13 class_cmds: Class Commands

This section describes the Class Commands hooks library, which exposes several control commands for manipulating client classes (part of the Kea DHCP servers' configurations) without the need to restart those servers. Using these commands it is possible to add, update, delete, and list client classes configured for a given server.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

The Class Commands hooks library is currently available only to ISC customers with a paid support contract.

16.13.1 The class-add Command

The `class-add` command adds a new client class to the DHCP server configuration. This class is appended at the end of the list of classes used by the server and may depend on any of the already-configured client classes.

The following example demonstrates how to add a new client class to the DHCPv4 server configuration:

```
{
    "command": "class-add",
    "arguments": {
        "client-classes": [
            {
                "name": "ipxe_efi_x64",
                "test": "option[93].hex == 0x0009",
                "next-server": "192.0.2.254",
                "server-hostname": "hal9000",
                "boot-file-name": "/dev/null"
            }
        ]
    }
}
```

Note that the `client-classes` parameter is a JSON list, but it allows only a single client class to be present.

Here is the response to the `class-add` command in our example:

```
{
    "result": 0,
    "text": "Class 'ipxe_efi_x64' added."
}
```

16.13.2 The class-update Command

The `class-update` command updates an existing client class in the DHCP server configuration. If the client class with the given name does not exist, the server returns the result code of 3, which means that the server configuration is not modified and the client class does not exist. The `class-add` command must be used instead to create the new client class.

The `class-update` command has the same argument structure as the `class-add` command:

```
{
    "command": "class-update",
    "arguments": {
        "client-classes": [
            {
                "name": "ipxe_efi_x64",
                "test": "option[93].hex == 0x0017",
                "next-server": "0.0.0.0",
                "server-hostname": "xfce",
                "boot-file-name": "/dev/null"
            }
        ]
    }
}
```

Here is the response for our example:

```
{
    "result": 0,
    "text": "Class 'ipxe_efi_x64' updated."
}
```

Any parameter of the client class can be modified with this command, except `name`. There is currently no way to rename the class, because the class name is used as a key for searching the class to be updated. To achieve a similar effect to renaming the class, an existing class can be removed with the `class-del` command and then added again with a different name using `class-add`. Note, however, that the class with the new name will be added at the end of the list of configured classes.

16.13.3 The class-del Command

The `class-del` command is used to remove a particular class from the server configuration. The class to be removed is identified by name. The class is not removed if there are other classes depending on it; to remove such a class, the dependent classes must be removed first.

The following is a sample command removing the `ipxe_efi_x64` class:

```
{
    "command": "class-del",
    "arguments": {
        {
            "name": "ipxe_efi_x64"
        }
    }
}
```

Here is the response to the `class-del` command in our example, when the specified client class has been found:

```
{
    "result": 0,
    "text": "Class 'ipxe_efi_x64' deleted."
}
```

If the class does not exist, the result of 3 is returned.

16.13.4 The class-list Command

`class-list` is used to retrieve a list of all client classes. This command includes no arguments:

```
{
    "command": "class-list"
}
```

Here is the response of the server in our example, including the list of client classes:

```
{
    "result": 0,
    "text": "2 classes found",
    "arguments": {
        "client-classes": [
            {
                "name": "ipxe_efi_x64"
            },
            {
                "name": "pxeclient"
            }
        ]
    }
}
```

Note that the returned list does not contain full class definitions, but merely class names. To retrieve full class information, the `class-get` command should be used.

16.13.5 The class-get Command

`class-get` is used to retrieve detailed information about a specified class. The command structure is very simple:

```
{
    "command": "class-get",
    "arguments": {
        "name": "pxeclient"
    }
}
```

If the class with the specified name does not exist, the status code of 3 is returned. If the specified client class exists, the class details are returned in the following format:

```
{
    "result": 0,
    "text": "Class 'pxeclient' definition returned",
    "arguments": {
        "client-classes": [
            {
                "name": "pxeclient",
                "only-if-required": true,
                "test": "option[vendor-class-identifier].text == 'PXEClient'",
                "option-def": [
                    {
                        "name": "configfile",
                        "code": 209,
                        "type": "string"
                    }
                ],
                "option-data": [ ],
                "next-server": "0.0.0.0",
                "server-hostname": "xfce",
                "boot-file-name": "/dev/null"
            }
        ]
    }
}
```

Note that the example above is DHCPv4-specific; the last three parameters are only returned by the DHCPv4 server and are never returned by the DHCPv6 server. Also, some of the parameters provided in this example may not be returned if they are not specified for the class. Specifically, `only-if-required`, `test`, and `option-def` are not returned if they are not specified for the class.

16.14 cb_cmds: Configuration Backend Commands

This section describes the `cb_cmds` hooks library, which is used to manage Kea servers' configurations in the Configuration Backends. This library must be used in conjunction with the available CB hooks libraries implementing the common APIs to create, read, update, and delete (CRUD) the configuration information in the respective databases. For example: the `mysql_cb` hooks library, released in Kea 1.6.0, implements this API for MySQL. In order to manage the configuration information in the MySQL database, both the `mysql_cb` and `cb_cmds` libraries must be loaded by the server used for the configuration management.

The `cb_cmds` library is only available to ISC customers with a paid support contract.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

16.14.1 Commands Structure

There are 5 types of commands supported by this library:

- `del` - delete the selected object from the database, e.g. `remote-global-parameter4-del`.
- `get` - fetch the selected object from the database, e.g. `remote-subnet4-get`.
- `get-all` - fetch all objects of the particular type from the database, e.g. `remote-option-def4-get-all`.
- `list` - list all objects of the particular type in the database, e.g. `remote-network4-list`; this class of commands returns brief information about each object comparing to the output of `get-all`.
- `set` - creates or replaces an object of the given type in the database, e.g. `remote-option4-global-set`.

All types of commands accept an optional `remote` map which selects the database instance to which the command refers. For example:

```
{
    "command": "remote-subnet4-list",
    "arguments": {
        "remote": {
            "type": "mysql",
            "host": "192.0.2.33",
            "port": 3302
        }
    }
}
```

selects the MySQL database, running on host 192.0.2.33 and port 3302, to fetch the list of subnets from. All parameters in the `remote` argument are optional. The `port` parameter can be only specified in conjunction with the `host`. If no options in the `remote` parameter are to be specified, the parameter should be omitted. In this case, the server will use the first backend listed in the `config-control` map within the configuration of the server receiving the command.

Note: As of the Kea 1.6.0 release, it is possible to configure the Kea server to use only one configuration backend. Strictly speaking, it is possible to point the Kea server to at most one MySQL database using the `config-control` parameter. That's why, in this release, the `remote` parameter may be omitted in the commands and the cb_cmds hooks library will use the sole backend by default.

16.14.2 Control Commands for DHCP Servers

This section describes and gives some examples of the control commands implemented by the `cb_cmds` hooks library, to manage the configuration information of the DHCPv4 and DHCPv6 servers. Many of the commands are almost identical between DHCPv4 and DHCPv6; they only differ by the command name. Other commands differ slightly by the structure of the inserted data; for example, the structure of the IPv4 subnet information is different than that of the IPv6 subnet. Nevertheless, they still share the structure of their command arguments and thus it makes sense to describe them together.

In the following sections, various commands are described and some usage examples are provided. In the sections jointly describing the DHCPv4 and DHCPv6 variants of the particular command, we sometimes use the following notation: the `remote-subnet[46]-set` is the wildcard name for the two commands: `remote-subnet4-set` and `remote-subnet6-set`.

In addition, whenever the text in the subsequent sections refers to a DHCP command or DHCP parameter, it refers to both DHCPv4 and DHCPv6 variants. The text specific to the particular server type refers to them as: DHCPv4 command, DHCPv4 parameter, DHCPv6 command, DHCPv6 parameter, etc.

16.14.3 Metadata

The typical response to the `get` or `list` command includes a list of returned objects (e.g. subnets), and each such object contains the `metadata` map with some database-specific information describing this object. In other words, the metadata contains any information about the fetched object which may be useful for the administrator, but which is not part of the object specification from the DHCP server standpoint. In the Kea 1.6.0 release, the metadata is limited to the `server-tag`, which describes the association of the object with a particular server or all servers.

The following is the example response to the `remote-network4-list` command, which includes the metadata:

```
{
    "result": 0,
    "text": "1 IPv4 shared network(s) found.",
    "arguments": {
        "shared-networks": [
            {
                "name": "level3",
                "metadata": {
                    "server-tags": [ "all" ]
                }
            }
        ],
        "count": 1
    }
}
```

Client implementations must not assume that the metadata contains only the `server-tags` parameter. In future releases, this map will be extended with additional information, e.g. object modification time, log message created during the last modification, etc.

16.14.4 remote-server4-del, remote-server6-del commands

This command is used to delete the information about a selected DHCP server from the configuration database. The server is identified by a unique case insensitive server tag. For example:

```
{
    "command": "remote-server4-del",
    "arguments": {
        "servers": [
            {
                "server-tag": "server1"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

As a result of this command, the user defined server called *server1* is removed from the database. All associations of the configuration information with this server are automatically removed from the database. The non-shareable configuration information, such as: global parameters, option definitions and global options associated with the server are removed from the database. The shareable configuration information, i.e. the configuration elements which may be associated with more than one server, is preserved. In particular, the subnets and shared networks associated with the deleted servers are preserved. If any of the shareable configuration elements was associated only with the deleted server, this object becomes unassigned (orphaned). For example: if a subnet has been created and associated with the *server1* using the *remote-subnet4-set* command and the server1 is subsequently deleted, the subnet remains in the database but none of the servers can use this subnet. The subnet can be updated using the *remote-subnet4-set* and associated with some other server or with all servers using the special server tag "all". Such subnet can be also deleted from the database using the *remote-subnet4-del-by-id* or *remote-subnet4-del-by-prefix*, if it is no longer needed.

The following is the successful response to the *remote-server4-del* command:

```
{
    "result": 0,
    "text": "1 DHCPv4 server(s) deleted."
    "arguments": {
        "count": 1
    }
}
```

Note: The *remote-server4-del* and *remote-server6-del* commands must be used with care, because an accidental deletion of the server causes some parts of the existing configurations to be lost permanently from the database. This operation is not reversible. Re-creation of the accidentally deleted server does not revert the lost configuration for that server and such configuration must be re-created manually by the user.

16.14.5 remote-server4-get, remote-server6-get commands

This command is used to fetch the information about the selected DHCP server from the configuration database. For example:

```
{
    "command": "remote-server6-get"
    "arguments": {
        "servers": [
            {
                "server-tag": "server1"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

This command fetches the information about the DHCPv6 server identified by the server tag *server1*. The server tag is case insensitive. A successful response returns basic information about the server, such as server tag and the user's description of the server:

```
{
    "result": 0,
    "text": "DHCP server server1 found.",
    "arguments": {
        "servers": [
            {
                "server-tag": "server1",
                "description": "A DHCPv6 server located on the first floor."
            }
        ],
        "count": 1
    }
}
```

16.14.6 remote-server4-get-all, remote-server6-get-all commands

This command is used to fetch all user defined DHCPv4 or DHCPv6 servers from the database. The command structure is very simple:

```
{
    "command": "remote-server4-get-all"
    "arguments": {
        "remote": {
            "type": "mysql"
        }
    }
}
```

The response includes basic information about each server, such as its server tag and description:

```
{
    "result": 0,
    "text": "DHCPv4 servers found.",
    "arguments": {
        "servers": [
            {
                "server-tag": "server1",
                "description": "A DHCP server located on the first floor."
            },
            {
                "server-tag": "server2",
                "description": "An old DHCP server to be soon replaced."
            }
        ],
       "count": 2
    }
}
```

16.14.7 remote-server4-set, remote-server6-set commands

This command is used to create or replace an information about a DHCP server in the database. The information about the server must be created when there is a need to differentiate the configurations used by various Kea instances connecting to the same database. Various configuration elements, e.g. global parameters, subnets etc. may be explicitly associated with the selected servers (using server tags as identifiers), allowing only these servers to use the respective configuration elements. Using the particular server tag to make such associations is only possible when the server information has been stored in the database via the *remote-server4-set* or *remote-server6-set* commands. The following command creates a new (or updates an existing) DHCPv6 server in the database:

```
{
    "command": "remote-server6-set"
    "arguments": {
        "servers": [
            {
                "server-tag": "server1",
                "description": "A DHCP server on the ground floor."
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The server tag must be unique accross all servers in the database. When the server information under the given server tag already exists, it is replaced with the new information. The specified server tag is case insensitive. The maximum length of the server tag is 256 characters. The following keywords are reserved and must not be used as server tags: “all” and “any”.

The following is the example response to the above command:

```
{
    "result": 0,
    "text": "DHCPv6 server successfully set.",
    "arguments": {
        "servers": [
            {
                "server-tag": "server1",
                "description": "A DHCP server on the ground floor."
            }
        ]
    }
}
```

16.14.8 The remote-global-parameter4-del, remote-global-parameter6-del Commands

These commands are used to delete a global DHCP parameter from the configuration database. When the parameter is deleted from the database, the server will use the value specified in the configuration file for this parameter, or a default value if the parameter is not specified in the configuration file.

The following command attempts to delete the DHCPv4 `renew-timer` parameter common for all servers from the database:

```
{
    "command": "remote-global-parameter4-del",
    "arguments": {
        "parameters": [ "renew-timer" ],
        "remote": {
            "type": "mysql"
         },
        "server-tags": [ "all" ]
    }
}
```

If the server specific parameter is to be deleted, the *server-tags* list must contain the tag of the appropriate server. There must be exactly one server tag specified in this list.

16.14.9 The remote-global-parameter4-get, remote-global-parameter6-get Commands

These commands are used to fetch a scalar global DHCP parameter from the configuration database.

The following command attempts to fetch the `boot-file-name` parameter for the “server1”:

```
{
    "command": "remote-global-parameter4-get",
    "arguments": {
        "parameters": [ "boot-file-name" ],
         "remote": {
             "type": "mysql"
         },
         "server-tags": [ "server1" ]
    }
}
```

The returned value has one of the four scalar types: string, integer, real, or boolean. Non-scalar global configuration parameters, such as map or list, are not returned by this command.

In the case of the example above, the string value is returned, e.g.:

```
{
    "result": 0,
    "text": "1 DHCPv4 global parameter found.",
    "arguments": {
        "parameters": {
            "boot-file-name": "/dev/null",
            "metadata": {
                "server-tags": [ "all" ]
            }
        },
        "count": 1
    }
}
```

Note that the response above indicates that the returned parameter is associated with "all" servers rather than "server1" used in the command. This indicates that there is no server1 specific value in the database. Therefore, the value shared by all servers is returned. If there was the server1 specific value in the database this value would be returned instead.

The example response for the integer value is:

```
{
    "result": 0,
    "text": "1 DHCPv4 global parameter found.",
    "arguments": {
        "parameters": {
            "renew-timer": 2000,
            "metadata": {
                "server-tags": [ "server1" ]
            }
        },
        "count": 1
    }
}
```

The real value:

```
{
    "result": 0,
    "text": "1 DHCPv4 global parameter found.",
    "arguments": {
        "parameters": {
            "t1-percent": 0.85,
            "metadata": {
                "server-tags": [ "all" ]
            }
        },
        "count": 1
    }
}
```

Finally, the boolean value:

```
{
    "result": 0,
    "text": "1 DHCPv4 global parameter found.",
    "arguments": {
        "parameters": {
            "match-client-id": true,
            "metadata": {
                "server-tags": [ "server2" ]
            }
        },
        "count": 1
    }
}
```

16.14.10 The remote-global-parameter4-get-all, remote-global-parameter6-get-all Commands

These commands are used to fetch all global DHCP parameters from the database for the specified server. The following example demonstrates how to fetch all global parameters to be used by the server "server1":

```
{
    "command": "remote-global-parameter4-get-all",
    "arguments": {
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1" ]
    }
}
```

The example response may look as follows:

```
{
    "result": 0,
    "text": "DHCPv4 global parameters found.",
    "arguments": {
        "parameters": [
            {
                "boot-file-name": "/dev/null",
                "metadata": {
                    "server-tags": [ "server1" ]
                }
            },
            {
                "match-client-id": true,
                "metadata": {
                    "server-tags": [ "all" ]
                }
            }
        ],
        "count": 2
    }
}
```

The example response contains two parameters, one string parameter and one boolean parameter. The metadata returned for each parameter indicates if this parameter is specific to the "server1" or all servers. Since the *match-client-id* value is associated with "all" servers it indicates that there is no server1 specific setting for this parameter. Each parameter always has exactly one server tag associated with it, because the global parameters are non-shareable configuration elements.

Note: If the server tag is set to "all" in the command, the response will contain only the global parameters associated with the logical server "all". When the server tag points to the specific server (as in the example above), the returned list combines parameters associated with this server and all servers, but the former take precedence.

16.14.11 The remote-global-parameter4-set, remote-global-parameter6-set Commands

This command is used to create scalar global DHCP parameters in the database. If any of the parameters already exists, its value is replaced as a result of this command. It is possible to set multiple parameters within a single command, each having one of the four types: string, integer, real, or boolean. For example:

```
{
    "command": "remote-global-parameter4-set"
    "arguments": {
        "parameters": {
            "boot-file-name": "/dev/null",
            "renew-timer": 2000,
            "t1-percent": 0.85,
            "match-client-id": true
        },
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1" ]
    }
}
```

An error is returned if any of the parameters is not supported by the DHCP server or its type does not match. Care should be taken when multiple parameters are specified in a single command, because it is possible that only some of the parameters are stored successfully and some fail. If an error occurs when processing this command, it is recommended to use `remote-global-parameter[46]-get-all` to check which of the parameters have been stored/updated successfully and which have failed.

The *server-tags* list is mandatory and it must contain a single server tag or the keyword “all”. In the example above, all specified parameters are associated with the “server1” server.

16.14.12 The remote-network4-del, remote-network6-del Commands

These commands are used to delete an IPv4 or IPv6 shared network from the database. The optional parameter `subnets-action` determines whether the subnets belonging to the deleted shared network should also be deleted or preserved. The `subnets-action` parameter defaults to `keep`, which preserves the subnets. If it is set to `delete`, the subnets are deleted along with the shared network.

The following command:

```
{
    "command": "remote-network6-del",
    "arguments": {
        "shared-networks": [
            {
                "name": "level3"
            }
        ],
        "subnets-action": "keep",
        "remote": {
            "type": "mysql"
        }
    }
}
```

deletes the "level3" IPv6 shared network. The subnets are preserved, but they are disassociated from the deleted shared network and become global. This behavior corresponds to the behavior of the `network[46]-del` commands with respect to the `subnets-action` parameter.

Note that the *server-tags* parameter must not be used for this command.

16.14.13 The remote-network4-get, remote-network6-get Commands

These commands are used to retrieve information about an IPv4 or IPv6 shared network. The optional parameter `subnets-include` denotes whether the subnets belonging to the shared network should also be returned. This parameter defaults to `no`, in which case the subnets are not returned. If this parameter is set to `full`, the subnets are returned together with the shared network.

The following command fetches the "level3" IPv6 shared network along with the full information about the subnets belonging to it:

```
{
    "command": "remote-network6-get",
    "arguments": {
        "shared-networks": [
            {
                "name": "level3"
            }
        ],
        "subnets-include": "full",
        "remote": {
            "type": "mysql"
        }
    }
}
```

Note that the *server-tags* parameter must not be used for this command.

16.14.14 The remote-network4-list, remote-network6-list Commands

These commands are used to list all IPv4 or IPv6 shared networks for a server.

The following command retrieves all shared networks to be used by the "server1" and "server2":

```
{
    "command": "remote-network4-list"
    "arguments": {
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1", "server2" ]
    }
}
```

The *server-tags* parameter is mandatory and it contains one or more server tags. It may contain the keyword "all" to fetch the shared networks associated with all servers. When the *server-tags* list contains the *null* value the returned response contains a list of unassigned shared networks, i.e. the networks which are associated with no servers. For example:

```
{
    "command": "remote-network4-list"
    "arguments": {
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ null ]
    }
}
```

The example response to this command when non-null server tags are specified looks similar to this:

```
{
    "result": 0,
    "text": "3 IPv4 shared network(s) found.",
    "arguments": {
        "shared-networks": [
            {
                "name": "ground floor",
                "metadata": {
                    "server-tags": [ "all" ]
                }
            },
            {
                "name": "floor2",
                "metadata": {
                    "server-tags": [ "server1" ]
                }
            },
            {
                "name": "floor3",
                "metadata": {
                    "server-tags": [ "server2" ]
                }
            }
        ],
        "count": 3
    }
}
```

The returned information about each shared network merely contains the shared network name and the metadata. In order to fetch the detailed information about the selected shared network, use the *remote-network[46]-get* command.

The example response above contains three shared networks. One of the shared networks is associated will all servers, so it is included in the list of shared networks to be used by the “server1” and “server2”. The remaining two shared networks are returned because one of them is associated with the “server1” and another one is associated with the “server2”.

When listing unassigned shared networks, the response will look similar to this:

```
{
    "result": 0,
    "text": "1 IPv4 shared network(s) found.",
    "arguments": {
        "shared-networks": [
            {
                "name": "fancy",
                "metadata": {
                    "server-tags": [ null ]
                }
            }
        ],
        "count": 1
    }
}
```

The *null* value in the metadata indicates that the returned shared network is unassigned.

16.14.15 The remote-network4-set, remote-network6-set Commands

These commands create a new or replace an existing IPv4 or IPv6 shared network in the database. The structure of the shared network information is the same as in the Kea configuration file (see *Shared Networks in DHCPv4* and *Shared Networks in DHCPv6* for details), except that specifying subnets along with the shared network information is not allowed. Including the `subnet4` or `subnet6` parameter within the shared network information will result in an error.

These commands are intended to be used for managing the shared network-specific information and DHCP options. In order to associate and disassociate the subnets with the shared networks, the `remote-subnet[46]-set` commands should be used.

The following command adds the IPv6 shared network "level3" to the database:

```
{
    "command": "remote-network6-set",
    "arguments": {
        "shared-networks": [
            {
                "name": "level3",
                "interface": "eth0",
                "option-data": [ {
                    "name": "sntp-servers",
                    "data": "2001:db8:1::1"
                } ],
            }
        ],
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "all" ]
    }
}
```

This command includes the `interface` parameter, which sets the shared network-level interface name. Any remaining shared network-level parameters, which are not specified with the command, will be marked as "unspecified" in the database. The DHCP server will use the global values for unspecified parameters or, if the global values are not specified, the default values will be used.

The *server-tags* list is mandatory for this command and it must include one or more server tags. As a result the shared network is associated with all listed servers. The shared network may be associated with all servers connecting to the database when the keyword "all" is included.

Note: As with other "set" commands, this command replaces all the information about the given shared network in the database, if the shared network already exists. Therefore, when sending this command, make sure to always include all parameters that must be specified for the updated shared-network instance. Any unspecified parameter will be marked unspecified in the database, even if its value was present prior to sending the command.

16.14.16 The remote-option-def4-del, remote-option-def6-del Commands

These commands are used to delete a DHCP option definition from the database. The option definition is identified by an option code and option space. For example:

```
{
    "command": "remote-option-def6-del",
    "arguments": {
        "option-defs": [
            {
                "code": 1,
                "space": "isc"
            }
        ],
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1" ]
    }
}
```

deletes the definition of the option associated with the "server1", having the code of 1 and belonging to the option space "isc". The default option spaces are "dhcp4" and "dhcp6" for the DHCPv4 and DHCPv6 top level options respectively. If there is no such option explicitly associated with the server1, no option is deleted. In order to delete an option belonging to "all" servers, the keyword "all" must be used as server tag. The *server-tags* list must contain exactly one tag. It must not include the *null* value.

16.14.17 The remote-option-def4-get, remote-option-def6-get Commands

These commands are used to fetch a specified DHCP option definition from the database. The option definition is identified by the option code and option space. The default option spaces are "dhcp4" and "dhcp6" for the DHCPv4 and DHCPv6 top-level options, respectively.

The following command retrieves a DHCPv4 option definition associated with all servers, having the code of 1 and belonging to the option space "isc":

```
{
    "command": "remote-option-def4-get"
    "arguments": {
        "option-defs": [
            {
                "code": 1,
                "space": "isc"
            }
        ],
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "all" ]
    }
}
```

The *server-tags* list must include exactly one server tag or the keyword "all". It must not contain the *null* value.

16.14.18 The remote-option-def4-get-all, remote-option-def6-get-all Commands

These commands are used to fetch all DHCP option definitions from the database for the particular server or all servers. For example:

```
{
    "command": "remote-option-def6-get-all"
    "arguments": {
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "all" ]
    }
}
```

This command attempts to fetch all DHCPv6 option definitions associated with "all" servers. The *server-tags* list is mandatory for this command and it must include exactly one server tag or the keyword "all". It must not include the *null* value.</para>

The following is the example response to this command:

```
{
    "result": 0,
    "text": "1 DHCPv6 option definition(s) found.",
    "arguments": {
        "option-defs": [
            {
                "name": "bar",
                "code": 1012,
                "space": "dhcp6",
                "type": "record",
                "array": true,
                "record-types": "ipv6-address, uint16",
                "encapsulate": "",
                "metadata": {
                    "server-tags": [ "all" ]
                }
            }
        ],
        "count": 1
    }
}
```

The response contains an option definition associated with all servers as indicated by the metadata.

16.14.19 The remote-option-def4-set, remote-option-def6-set Commands

These commands create a new DHCP option definition or replace an existing option definition in the database. The structure of the option definition information is the same as in the Kea configuration file (see *Custom DHCPv4 Options* and *Custom DHCPv6 Options*). The following command creates the DHCPv4 option definition in the top-level "dhcp4" option space and associates it with the "server1":

```
{
    "command": "remote-option-def4-set",
    "arguments": {
        "option-defs": [
            {
                "name": "foo",
                "code": 222,
                "type": "uint32",
                "array": false,
                "record-types": "",
                "space": "dhcp4",
                "encapsulate": ""
            }
        ],
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1" ]
    }
}
```

The *server-tags* list must include exactly one server tag or the keyword “all”. It must not contain the *null* value.</para>

16.14.20 The remote-option4-global-del, remote-option6-global-del Commands

These commands are used to delete a global DHCP option from the database. The option is identified by an option code and option space. For example:

```
{
    "command": "remote-option4-global-del",
    "arguments": {
        "options": [
            {
                "code": 5
                "space": "dhcp4"
            }
        ],
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1" ]
    }
}
```

The “dhcp4” is the top-level option space where the standard DHCPv4 options belong. The *server-tags* is mandatory and it must include a single option tag or the keyword “all”. If the explicit server tag is specified then this command attempts to delete a global option associated with this server. If there is no such option associated with the given server, no option is deleted. In order to delete the option associated with all servers, the keyword “all” must be specified.

16.14.21 The remote-option4-global-get, remote-option6-global-get Commands

These commands are used to fetch a global DHCP option from the database. The option is identified by the code and option space. The top-level option spaces where DHCP standard options belong are called “dhcp4” and “dhcp6” for the DHCPv4 and DHCPv6 servers, respectively.

The following command retrieves the IPv6 “DNS Servers” (code 23) option associated with all servers:

```
{
    "command": remote-option6-global-get",
    "arguments": {
        "options": [
            {
                "code": 23,
                "space": "dhcp6"
            }
        ],
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "all" ]
    }
}
```

The *server-tags* is mandatory and it must include exactly one server tag or the keyword “all”. It must not contain the *null* value.

16.14.22 The remote-option4-global-get-all, remote-option6-global-get-all Commands

These commands are used to fetch all global DHCP options from the configuration database for the particular server or for all servers. The following command fetches all global DHCPv4 options for the “server1”:

```
{
    "command": "remote-option6-global-get-all",
    "arguments": {
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1" ]
    }
}
```

The *server-tags* list is mandatory for this command and it must contain exactly one server tag or a keyword “all”. It must not contain the *null* value. The following is the example response to this command with a single option being associated with the “server1” returned:

```
{
    "result": 0,
    "text": "DHCPv4 options found.",
    "arguments": {
        "options": [
            {
                "name": "domain-name-servers",
                "code": 6,
                "space": "dhcp4",
                "csv-format": false,
                "data": "192.0.2.3",
                "metadata": {
                    "server-tags": [ "server1" ]
                }
            }
        ],
        "count": 1
    }
}
```

16.14.23 The remote-option4-global-set, remote-option6-global-set Commands

These commands create a new global DHCP option or replace an existing option in the database. The structure of the option information is the same as in the Kea configuration file (see *Standard DHCPv4 Options* and *Standard DHCPv6 Options*). For example:

```
{
    "command": "remote-option6-global-set",
    "arguments": {
        "options": [
            {
                "name": "dns-servers",
                "data": "2001:db8:1::1"
            }
        ],
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1" ]
    }
}
```

The *server-tags* list is mandatory for this command and it must include exactly one server tag or the keyword "all". It must not include the *null* value. The command above associates the option with the "server1" server.

Note that specifying an option name instead of the option code only works reliably for the standard DHCP options. When specifying a value for the user-defined DHCP option, the option code should be specified instead of the name. For example:

```
{
    "command": "remote-option6-global-set",
    "arguments": {
        "options": [
            {
                "code": 1,
                "space": "isc",
                "data": "2001:db8:1::1"
            }
        ],
        "server-tags": [ "server1" ]
    }
}
```

16.14.24 The remote-option4-network-del, remote-option6-network-del Commands

These commands are used to delete a shared network specific DHCP option from the database. The option is identified by an option code and option space and these two parameters are passed within the *options* list. Another list, *shared-networks*, contains a map with the name of the shared network from which the option is to be deleted. If the option is not explicitly specified for this shared network, no option is deleted. In particular, the given option may be present for a subnet belonging to the shared network. Such option instance is not affected by this command as this command merely deletes shared network level option. In order to delete subnet level option the *remote-option[46]-subnet-del* command must be used instead.

The following command attempts to delete an option having the option code 5 in the top-level option space from the shared network "fancy".

```
{
    "command": "remote-option4-network-del",
    "arguments": {
        "shared-networks": [
            {
                "name": "fancy"
            }
        ],
        "options": [
            {
                "code": 5,
                "space": "dhcp4"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The “dhcp4” is the top-level option space where the standard DHCPv4 options belong. The *server-tags* parameter must not be specified for this command.

16.14.25 The remote-option4-network-set, remote-option6-network-set Commands

These commands create a new shared network specific DHCP option or replace an existing option in the database. The structure of the option information is the same as in the Kea configuration file (see *Standard DHCPv4 Options* and *Standard DHCPv6 Options*). The option information is carried in the *options* list. Another list, *shared-networks*, contains a map with the name of the shared network for which the option is to be set. If such option already exists for the shared network, it is replaced with the new instance.

```
{
    "command": "remote-option6-network-set",
    "arguments": {
        "shared-networks": [
            {
                "name": "fancy"
            }
        ],
        "options": [
            {
                "name": "dns-servers",
                "data": "2001:db8:1::1"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The *sever-tags* parameter must not be specified for this command.

Specifying an option name instead of the option code only works reliably for the standard DHCP options. When specifying a value for the user-defined DHCP option, the option code should be specified instead of the name.

16.14.26 The remote-option6-pd-pool-del Command

This command is used to delete a prefix delegation pool specific DHCPv6 option from the database. The option is identified by an option code and option space and these two parameters are passed within the *options* list. Another list, *pd-pools*, contains a map with the prefix delegation pool prefix and length identifying the pool. If the option is not explicitly specified for this pool, no option is deleted. In particular, the given option may exist for a subnet containing the specified pool. Such option instance is not affected by this command as this command merely deletes a prefix delegation pool level option. In order to delete subnet level option the *remote-option6-subnet-del* command must be used instead.

```
{
    "command": "remote-option6-pd-pool-del",
    "arguments": {
        "pd-pools": [
            {
                "prefix": "3000::",
                "prefix-len": 64
            }
        ],
        "options": [
            {
                "code": 23,
                "space": "dhcp6"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The "dhcp6" is the top-level option space where the standard DHCPv6 options belong. The *server-tags* parameter must not be specified for this command.

16.14.27 The remote-option6-pd-pool-set Command

This command creates a new prefix delefation pool specific DHCPv6 option or replaces an existing option in the database. The structure of the option information is the same as in the Kea configuration file (see *Standard DHCPv4 Options* and *Standard DHCPv6 Options*). The option information is carried in the *options* list. Another list, *pd-pools*, contains a map with the prefix delegation pool prefix and the prefix length identifying the pool. If such option already exists for the prefix delegation pool, it is replaced with the new instance.

For example:

```
{
    "command": "remote-option6-pd-pool-set",
    "arguments": {
        "pd-pools": [
            {
                "prefix": "3001:1::",
                "length": 64
            }
        ],
        "options": [
            {
                "name": "dns-servers",
                "data": "2001:db8:1::1"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The *sever-tags* parameter must not be specified for this command.

Specifying an option name instead of the option code only works reliably for the standard DHCP options. When specifying a value for the user-defined DHCP option, the option code should be specified instead of the name.

16.14.28 The remote-option4-pool-del, remote-option6-pool-del Commands

These commands are used to delete an address pool specific DHCP option from the database. The option is identified by an option code and option space and these two parameters are passed within the *options* list. Another list, *pools*, contains a map with the IP address range or prefix identifying the pool. If the option is not explicitly specified for this pool, no option is deleted. In particular, the given option may exist for a subnet containing the specified pool. Such option instance is not affected by this command as this command merely deletes a pool level option. In order to delete subnet level option the *remote-option[46]-subnet-del* command must be used instead.

The following command attempts to delete an option having the option code 5 in the top-level option space from an IPv4 address pool:

```
{
    "command": "remote-option4-pool-del",
    "arguments": {
        "pools": [
            {
                "pool": "192.0.2.10 - 192.0.2.100"
            }
        ],
        "options": [
            {
                "code": 5,
                "space": "dhcp4"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The "dhcp4" is the top-level option space where the standard DHCPv4 options belong. The *server-tags* parameter must not be specified for this command.

16.14.29 The remote-option4-pool-set, remote-option6-pool-set Commands

These commands create a new address pool specific DHCP option or replace an existing option in the database. The structure of the option information is the same as in the Kea configuration file (see *Standard DHCPv4 Options* and *Standard DHCPv6 Options*). The option information is carried in the *options* list. Another list, *pools*, contains a map with the IP address range or prefix identifying the pool. If such option already exists for the pool, it is replaced with the new instance.

For example:

```
{
    "command": "remote-option4-pool-set",
    "arguments": {
        "pools": [
            {
                "pool": "192.0.2.10 - 192.0.2.100"
            }
        ],
        "options": [
            {
                "name": "domain-name-servers",
                "data": "10.0.0.1"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The *sever-tags* parameter must not be specified for this command.

Specifying an option name instead of the option code only works reliably for the standard DHCP options. When specifying a value for the user-defined DHCP option, the option code should be specified instead of the name.

16.14.30 The remote-option4-subnet-del, remote-option6-subnet-del Commands

These commands are used to delete a subnet specific DHCP option from the database. The option is identified by an option code and option space and these two parameters are passed within the *options* list. Another list, *subnets*, contains a map with the identifier of the subnet from which the option is to be deleted. If the option is not explicitly specified for this subnet, no option is deleted.

The following command attempts to delete an option having the option code 5 in the top-level option space from the subnet having an identifer of 123.

```
{
    "command": "remote-option4-subnet-del",
    "arguments": {
        "subnets": [
            {
                "id": 123
            }
        ],
        "options": [
            {
                "code": 5,
                "space": "dhcp4"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The “dhcp4” is the top-level option space where the standard DHCPv4 options belong. The *server-tags* parameter must not be specified for this command.

16.14.31 The remote-option4-subnet-set, remote-option6-subnet-set Commands

These commands create a new subnet specific DHCP option or replace an existing option in the database. The structure of the option information is the same as in the Kea configuration file (see *Standard DHCPv4 Options* and *Standard DHCPv6 Options*). The option information is carried in the *options* list. Another list, *subnets*, contains a map with the identifier of the subnet for which the option is to be set. If such option already exists for the subnet, it is replaced with the new instance.

```
{
    "command": "remote-option6-subnet-set",
    "arguments": {
        "subnets": [
            {
                "id": 123
            }
        ],
        "options": [
            {
                "name": "dns-servers",
                "data": "2001:db8:1::1"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The *sever-tags* parameter must not be specified for this command.

Specifying an option name instead of the option code only works reliably for the standard DHCP options. When specifying a value for the user-defined DHCP option, the option code should be specified instead of the name.

16.14.32 The remote-subnet4-del-by-id, remote-subnet6-del-by-id Commands

This is the first variant of the commands used to delete an IPv4 or IPv6 subnet from the database. It uses the subnet ID to identify the subnet. For example, to delete the IPv4 subnet with an ID of 5:

```
{
    "command": "remote-subnet4-del-by-id",
    "arguments": {
        "subnets": [
            {
                "id": 5
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The *server-tags* parameter must not be used with this command.

16.14.33 The remote-subnet4-del-by-prefix, remote-subnet6-del-by-prefix Commands

This is the second variant of the commands used to delete an IPv4 or IPv6 subnet from the database. It uses the subnet prefix to identify the subnet. For example:

```
{
    "command": "remote-subnet6-del-by-prefix",
    "arguments": {
        "subnets": [
            {
                "subnet": "2001:db8:1::/64"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The *server-tags* parameter must not be used with this command.

16.14.34 The remote-subnet4-get-by-id, remote-subnet6-get-by-id Commands

This is the first variant of the commands used to fetch an IPv4 or IPv6 subnet from the database. It uses a subnet ID to identify the subnet. For example:

```
{
    "command": "remote-subnet4-get-by-id",
    "arguments": {
        "subnets": [
            {
                "id": 5
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The *server-tags* parameter must not be used with this command.

16.14.35 The remote-subnet4-get-by-prefix, remote-subnet6-get-by-prefix Commands

This is the second variant of the commands used to fetch an IPv4 or IPv6 subnet from the database. It uses a subnet prefix to identify the subnet. For example:

```
{
    "command": "remote-subnet6-get-by-prefix",
    "arguments": {
        "subnets": [
            {
                "subnet": "2001:db8:1::/64"
            }
        ],
        "remote": {
            "type": "mysql"
        }
    }
}
```

The *server-tags* parameter must not be used with this command.

16.14.36 The remote-subnet4-list, remote-subnet6-list Commands

These commands are used to list all IPv4 or IPv6 subnets from the database for selected servers or all servers. The following command retrieves all servers to be used by the “server1” and “server2”:

```
{
    "command": "remote-subnet4-list"
    "arguments": {
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "server1", "server2" ]
    }
}
```

The *server-tags* parameter is mandatory and it contains one or more server tags. It may contain the keyword “all” to fetchg the subnets associated with all servers. When the *server-tags* list contains the *null* value the returned response contains a list of unassigned subnets, i.e. the subnets which are associated with no servers. For example:

```
{
    "command": "remote-subnet4-list"
    "arguments": {
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ null ]
    }
}
```

The example response to this command when non-null server tags are specified looks similar to this:

```
{
    "result": 0,
    "text": "2 IPv4 subnet(s) found.",
    "arguments": {
        "subnets": [
            {
                "id": 1,
                "subnet": "192.0.2.0/24",
                "shared-network-name": null,
                "metadata": {
                    "server-tags": [ "server1", "server2" ]
                }
            },
            {
                "id": 2,
                "subnet": "192.0.3.0/24",
                "shared-network-name": null,
                "metadata": {
                    "server-tags": [ "all" ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned information about each subnet is limited to subnet identifier, prefix and associated shared network name. In order to retrieve full information about the selected subnet use the *remote-subnet[46]-get-by-id* or *remote-subnet[46]-get-by-prefix*.

The example response above contains two subnets. One of the subnets is associated with both servers: "server1" and "server2". The second subnet is associated with all servers, thus it is also present in the configuration for the "server1" and "server2".

When listing unassigned subnets, the response will look similar to this:

```
{
    "result": 0,
    "text": "1 IPv4 subnet(s) found.",
    "arguments": {
        "subnets": [
            {
                "id": 3,
                "subnet": "192.0.4.0/24",
                "shared-network-name": null,
                "metadata": {
                    "server-tags": [ null ]
                }
            }
        ],
        "count": 1
    }
}
```

The *null* value in the metadata indicates that the returned subnet is unassigned.

16.14.37 The remote-subnet4-set, remote-subnet6-set Commands

These commands are used to create a new IPv4 or IPv6 subnet or replace an existing subnet in the database. Setting the subnet also associates or disassociates the subnet with a shared network.

The structure of the subnet information is similar to the structure used in the configuration file (see *DHCPv4 Server Configuration* and *DHCPv6 Server Configuration*). The subnet information conveyed in the `remote-subnet[46]-set` command must include the additional parameter `shared-network-name`, which denotes whether the subnet belongs to a shared network.

Consider the following example:

```
{
    "command": "remote-subnet4-set",
    "arguments": {
        "subnets": [
            {
                "id": 5,
                "subnet": "192.0.2.0/24",
                "shared-network-name": "level3",
                "pools": [ { "pool": "192.0.2.100-192.0.2.200" } ],
                "option-data": [ {
                    "name": "routers",
                    "data": "192.0.2.1"
                } ]
            }
        ],
        "remote": {
            "type": "mysql"
        },
        "server-tags": [ "all" ]
    }
}
```

It creates the subnet and associates it with the “level3” shared network. The “level3” shared network must be created with the `remote-network4-set` command prior to creating the subnet.

If the created subnet must be global - that is, not associated with any shared network - the `shared-network-name` must be explicitly set to `null`:

```
{
    "command": "remote-subnet4-set",
    "arguments": {
        "subnets": [
            {
                "id": 5,
                "subnet": "192.0.2.0/24",
                "shared-network-name": null,
                "pools": [ { "pool": "192.0.2.100-192.0.2.200" } ],
                "option-data": [ {
                    "name": "routers",
                    "data": "192.0.2.1"
                } ]
           }
        ],
        "server-tags": [ "all" ]
    }
}
```

The subnet created in the previous example is replaced with the new subnet having the same parameters, but it becomes global.

The `shared-network-name` parameter is mandatory for the `remote-subnet4-set` command. The *server-tags* list is mandatory and it must include one or more server tags. As a result, the subnet is associated with all of the listed servers. It may also be associated with “all” servers connecting to the database when the keyword “all” is used as the server tag.</para>

Note: As with other “set” commands, this command replaces all the information about the particular subnet in the database, if the subnet information is already present. Therefore, when sending this command, make sure to always include all parameters that must be specified for the updated subnet instance. Any unspecified parameter will be marked as unspecified in the database, even if its value was present prior to sending the command.

16.15 ha: High Availability

This section describes the High Availability hooks library, which can be loaded on a pair of DHCPv4 or DHCPv6 servers to increase the reliability of the DHCP service in the event of an outage of one of the servers. This library was previously only available to ISC's paid subscribers, but is now part of the open source Kea, available to all users.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

High Availability (HA) of the DHCP service is provided by running multiple cooperating server instances. If any of these instances becomes unavailable for any reason (DHCP software crash, Control Agent software crash, power outage, hardware failure), a surviving server instance can continue providing reliable service to clients. Many DHCP server implementations include the "DHCP Failover" protocol, whose most significant features are communication between the servers, partner failure detection, and lease synchronization between the servers. However, the DHCPv4 failover standardization process was never completed by the IETF. The DHCPv6 failover standard (RFC 8156) was published, but it is complex, difficult to use, has significant operational constraints, and is different than its v4 counterpart. Although it may be useful for some users to use a "standard" failover protocol, it seems that most Kea users are simply interested in a working solution which guarantees high availability of the DHCP service. Therefore, the Kea HA hook library derives major concepts from the DHCP Failover protocol but uses its own solutions for communication and configuration. It offers its own state machine, which greatly simplifies its implementation and generally fits better into Kea, and it provides the same features in both DHCPv4 and DHCPv6. This document intentionally uses the term "High Availability" rather than "Failover" to emphasize that it is not the Failover protocol implementation.

The following sections describe the configuration and operation of the Kea HA hook library.

16.15.1 Supported Configurations

The Kea HA hook library supports three configurations, also known as HA modes: load-balancing, hot-standby and passive-backup. In the load-balancing mode, two servers respond to DHCP requests. The load-balancing function is implemented as described in RFC 3074, with each server responding to half the received DHCP queries. When one of the servers allocates a lease for a client, it notifies the partner server over the control channel (the RESTful API), so the partner can save the lease information in its own database. If the communication with the partner is unsuccessful, the DHCP query is dropped and the response is not returned to the DHCP client. If the lease update is successful, the response is returned to the DHCP client by the server which has allocated the lease. By exchanging lease updates, both servers get a copy of all leases allocated by the entire HA setup, and either server can be switched to handle the entire DHCP traffic if its partner becomes unavailable.

In the load-balancing configuration, one of the servers must be designated as `"primary"` and the other as `"secondary."` Functionally, there is no difference between the two during normal operation. This distinction is required when the two servers are started at (nearly) the same time and have to synchronize their lease databases. The primary server synchronizes the database first. The secondary server waits for the primary server to complete the lease database synchronization before it starts the synchronization.

In the hot-standby configuration, one of the servers is also designated as "primary" and the second as "secondary." However, during normal operation, the primary server is the only one that responds to DHCP requests. The secondary or standby server receives lease updates from the primary over the control channel; however, it does not respond to any DHCP queries as long as the primary is running or, more accurately, until the secondary considers the primary to be offline. If the secondary server detects the failure of the primary, it starts responding to all DHCP queries.

In the configurations described above, the primary and secondary/standby are referred to as `"active"` servers, because they receive lease updates and can automatically react to the partner's failures by responding to the DHCP queries which would normally be handled by the partner. The HA hook library supports another server type/role: `"backup"` server. The use of a backup server is optional, and can be implemented in both load-balancing and hot-standby setup, in addition to the active servers. There is no limit on the number of backup servers in the HA setup; however, the presence of backup servers may increase the latency of DHCP responses, because not only do active servers send lease updates to each other, but also to the backup servers. As of Kea 1.7.8 the active servers no longer expect the acknowledgments from the backup servers before responding to the DHCP clients, therefore the overhead of sending the lease updates to the backup servers is minimized comparing to the earlier Kea versions.

The last supported configuration, passive-backup, has been introduced in Kea 1.7.8 release. In this configuration there is only one active server and typically one or more backup servers. A passive-backup configuration with no backup servers is also accepted but it is no different than running a single server with no HA function at all.

The passive-backup configuration is used in situations when an administrator wants to take advantage of the backup servers as an additional storage for leases without a need for running the fully blown failover setup. In this case, if the primary server fails, the DHCP service is lost and it requires that the administrator manually starts the primary to resume the DHCP service. The administrator may also configure one of the backup servers to provide the DHCP service to the clients as these servers should have accurate or nearly accurate information about the allocated leases. The major advantage of the passive-backup mode is that it provides some redundancy of the lease information but with better performance of the primary server responding to the DHCP queries. Since Kea 1.7.8 release, the primary server does not have to wait for the acknowledgments to the lease updates from the backup servers before it sends a response to the DHCP client. This reduces the response time comparing to the load-balancing and hot-standby cases in which the server responding to the DHCP query has to wait for the acknowledgment from the other active server before it can respond to the client.

Note: An interesting use case for a single active server running in the passive-backup mode is a notification service in which a software pretending to be a backup server receives live notifications about allocated and deleted leases from the primary server and can display them on the monitoring screen, trigger alerts etc.

16.15.2 Clocks on Active Servers

Synchronized clocks are essential for the HA setup to operate reliably. The servers share lease information via lease updates and during synchronization of the databases. The lease information includes the time when the lease was allocated and when it expires. Some clock skew between the servers participating in the HA setup usually exists; this is acceptable as long as the clock skew is relatively low, compared to the lease lifetimes. However, if the clock skew becomes too high, the different lease expiration times on different servers may cause the HA system to malfunction. For example, one server may consider a lease to be expired when it is actually still valid. The lease reclamation process may remove a name associated with this lease from the DNS, causing problems when the client later attempts to renew the lease.

Each active server monitors the clock skew by comparing its current time with the time returned by its partner in response to the heartbeat command. This gives a good approximation of the clock skew, although it doesn't take into account the time between sending the response by the partner and receiving this response by the server which sent the heartbeat command. If the clock skew exceeds 30 seconds, a warning log message is issued. The administrator may correct this problem by synchronizing the clocks (e.g. using NTP); the servers should notice the clock skew correction and stop issuing the warning.

If the clock skew is not corrected and exceeds 60 seconds, the HA service on each of the servers is terminated, i.e. the state machine enters the `terminated` state. The servers will continue to respond to DHCP clients (as in the load-balancing or hot-standby mode), but will exchange neither lease updates nor heartbeats and their lease databases will diverge. In this case, the administrator should synchronize the clocks and restart the servers.

Note: Prior to Kea 1.7.8 release, in order to recover from the terminated state, the administrator had to shutdown both servers and then start both of them. Since Kea 1.7.8 release it is allowed to restart the servers sequentially, i.e. restart one server and then restart another one. The clocks must be in sync before restarting the servers.

Note: The clock skew is only assessed between two active servers and only the active servers may enter the terminated state if it is too high. As of Kea 1.7.8, the clock skew between the active and the backup servers is not assessed because the active servers do not exchange heartbeat messages with the backup servers.

16.15.3 Server States

A DHCP server operating within an HA setup runs a state machine, and the state of the server can be retrieved by its peers using the `ha-heartbeat` command sent over the RESTful API. If the partner server doesn't respond to the `ha-heartbeat` command within the specified amount of time, the communication is considered interrupted and the server may, depending on the configuration, use additional measures (described later in this document) to verify that the partner is still operating. If it finds that the partner is not operating, the server transitions to the `partner-down` state to handle all the DHCP traffic directed to the system.

In this case, the surviving server continues to send the `ha-heartbeat` command to detect when the partner wakes up. At that time, the partner synchronizes the lease database and when it is again ready to operate, the surviving server returns to normal operation, i.e. the `load-balancing` or `hot-standby` state.

The following is the list of all possible server states:

- `backup` - normal operation of the backup server. In this state it receives lease updates from the active servers.
- `hot-standby` - normal operation of the active server running in the hot-standby mode; both the primary and the standby server are in this state during their normal operation. The primary server responds to DHCP queries and sends lease updates to the standby server and to any backup servers that are present.
- `load-balancing` - normal operation of the active server running in the load-balancing mode; both the primary and the secondary server are in this state during their normal operation. Both servers respond to DHCP queries and send lease updates to each other and to any backup servers that are present.
- `in-maintenance` - an active server transitions to this state as a result of being notified by its partner that the administrator requested maintenance of the HA setup. The administrator requests the maintenance by sending the `ha-maintenance-start` to the server which is supposed to take over the responsibility for responding to the DHCP clients while the other server is taken offline for maintenance. If the server is in the `in-maintenance` state it can be safely shut down. The partner is in the `partner-in-maintenance` state from which it will transition to the `partner-down` state immediately after it finds that the server in maintenance was shut down.
- `partner-down` - an active server transitions to this state after detecting that its partner (another active server) is offline. The server does not transition to this state if only a backup server is unavailable. In the `partner-down` state the active server responds to all DHCP queries, including those queries which are normally handled by the server that is now unavailable.
- `partner-in-maintenance` - an active server transitions to this state after receiving a `ha-maintenance-start` command from the administrator. The server in this state becomes responsible for responding to all DHCP requests. The server sends `ha-maintenance-notify` command to the partner which is supposed to enter the `in-maintenance` state. If that is the case, the server remaining in the `partner-in-maintenance` state keeps sending lease updates to the partner until it finds that the parter stops responding to those lease updates, heartbeats or any other commands. In this case, the server in the `partner-in-maintenance` state transitions to the `partner-down` state and keeps responding to the queries, but no longer sends lease updates.

- `passive-backup` - a primary server running in the passive-backup HA mode transitions to this state immediately after it is booted up. The primary server being in this state responds to the entire DHCP traffic and sends lease updates to the backup servers it is connected to. By default, the primary server does not wait for the acknowledgments from the backup servers and responds to the DHCP query right after sending the lease updates to all backup servers. If any of the lease updates fails, a backup server misses such lease update but the DHCP client is still provisioned. This default configuration can be changed by setting the `wait-backup-ack` configuration parameter to `true`, in which case the primary server always waits for the acknowledgements and drops the DHCP query if sending any of the correponding lease updates fails. This improves lease database consistency between the primary and the secondary. However, if a communication failure between the active server and any of the backups occurs, it effectively causes the failure of the DHCP service from the DHCP clients' perspective.
- `ready` - an active server transitions to this state after synchronizing its lease database with an active partner. This state indicates to the partner - which may be in the `partner-down` state - that it should return to normal operation. If and when it does, the server in the `ready` state will also start normal operation.
- `syncing` - an active server transitions to this state to fetch leases from the active partner and update the local lease database. When in this state, the server issues the `dhcp-disable` command to disable the DHCP service of the partner from which the leases are fetched. The DHCP service is disabled for a maximum time of 60 seconds, after which it is automatically re-enabled, in case the syncing partner was unable to re-enable the service. If the synchronization is completed, the syncing server issues the `dhcp-enable` command to re-enable the DHCP service of its partner. The syncing operation is synchronous; the server waits for an answer from the partner and does nothing else while the lease synchronization takes place. A server that is configured not to synchronize the lease database with its partner, i.e. when the `sync-leases` configuration parameter is set to `false`, will never transition to this state. Instead, it will transition directly from the `waiting` state to the `ready` state.
- `terminated` - an active server transitions to this state when the High Availability hooks library is unable to further provide reliable service and a manual intervention of the administrator is required to correct the problem. Various issues with the HA setup may cause the server to transition to this state. While in this state, the server continues responding to DHCP clients based on the HA mode selected (load-balancing or hot-standby), but the lease updates are not exchanged and the heartbeats are not sent. Once a server has entered the "terminated" state, it will remain in this state until it is restarted. The administrator must correct the issue which caused this situation prior to restarting the server (e.g. synchronize the clocks); otherwise, the server will return to the "terminated" state once it finds that the issue persists.
- `waiting` - each started server instance enters this state. The backup server transitions directly from this state to the `backup` state. An active server sends a heartbeat to its partner to check its state; if the partner appears to be unavailable, the server transitions to the `partner-down` state. If the partner is available, the server transitions to the `syncing` or `ready` state, depending on the setting of the `sync-leases` configuration parameter. If both servers appear to be in the `waiting` state (concurrent startup), the primary server transitions to the next state first. The secondary or standby server remains in the `waiting` state until the primary transitions to the `ready` state.

Note: Currently, restarting the HA service from the `terminated` state requires restarting the DHCP server or reloading its configuration.

Whether the server responds to the DHCP queries and which queries it responds to is a matter of the server's state, if no administrative action is performed to configure the server otherwise. The following table provides the default behavior for various states.

The `DHCP Server Scopes` denote what group of received DHCP queries the server responds to in the given state. An in-depth explanation of the scopes can be found below.

Table 16.2: Default Behavior of the Server in Various HA States

State	Server Type	DHCP Service	DHCP Service Scopes
backup	backup server	disabled	none
hot-standby	primary or standby (hot-standby mode)	enabled	`HA_server1` if primary, none otherwise
load-balancing	primary or secondary (load-balancing mode)	enabled	`HA_server1` or `HA_server2`
in-maintenance	active server	disabled	none
partner-down	active server	enabled	all scopes
partner-in-maintenance	active server	enabled	all scopes
passive-backup	active server	enabled	all scopes
ready	active server	disabled	none
syncing	active server	disabled	none
terminated	active server	enabled	same as in the load-balancing or hot-standby state
waiting	any server	disabled	none

The DHCP service scopes require some explanation. The HA configuration must specify a unique name for each server within the HA setup. This document uses the following convention within the provided examples: `server1` for a primary server, `server2` for the secondary or standby server, and `server3` for the backup server. In real life any names can be used as long as they remain unique.

In the load-balancing mode there are two scopes specified for the active servers: `HA_server1` and `HA_server2`. The DHCP queries load-balanced to `server1` belong to the `HA_server1` scope and the queries load-balanced to `server2` belong to the `HA_server2` scope. If either of the servers is in the `partner-down` state, the active partner is responsible for serving both scopes.

In the hot-standby mode, there is only one scope - `HA_server1` - because only `server1` is responding to DHCP queries. If that server becomes unavailable, `server2` becomes responsible for this scope.

The backup servers do not have their own scopes. In some cases they can be used to respond to queries belonging to the scopes of the active servers. Also, a server which is neither in the partner-down state nor in normal operation serves no scopes.

The scope names can be used to associate pools, subnets, and networks with certain servers, so only these servers can allocate addresses or prefixes from those pools, subnets, or networks. This is done via the client classification mechanism (see *Load Balancing with Advanced Classification* for more details).

16.15.4 Scope Transition in a Partner-Down Case

When one of the servers finds that its partner is unavailable, it starts serving clients from both its own scope and the scope of the unavailable partner. This is straightforward for new clients, i.e. those sending DHCPDISCOVER (DHCPv4) or Solicit (DHCPv6), because those requests are not sent to any particular server. The available server will respond to all such queries when it is in the `partner-down` state.

When a client renews a lease, it sends its DHCPREQUEST (DHCPv4) or Renew (DHCPv6) message directly to the server which has allocated the lease being renewed. If this server is no longer available, the client will get no response. In that case, the client continues to use its lease and attempts to renew until the rebind timer (T2) elapses. The client then enters the rebinding phase, in which it sends a DHCPREQUEST (DHCPv4) or Rebind (DHCPv6) message to any available server. The surviving server will receive the rebinding request and will typically extend the lifetime of the lease. The client then continues to contact that new server to renew its lease as appropriate.

If and when the other server once again becomes available, both active servers will eventually transition to the `load-balancing` or `hot-standby` state, in which they will again be responsible for their own scopes. Some clients belonging to the scope of the restarted server will try to renew their leases via the surviving server, but this server will not respond to them anymore; the client will eventually transition back to the correct server via the rebinding mechanism.

16.15.5 Load-Balancing Configuration

The following is the configuration snippet to enable high availability on the primary server within the load-balancing configuration. The same configuration should be applied on the secondary and backup servers, with the only difference that `this-server-name` should be set to `server2` and `server3` on those servers, respectively.

```
"Dhcp4": {
    "hooks-libraries": [{
        "library": "/usr/lib/kea/hooks/libdhcp_lease_cmds.so",
        "parameters": { }
    }, {
        "library": "/usr/lib/kea/hooks/libdhcp_ha.so",
        "parameters": {
            "high-availability": [{
                "this-server-name": "server1",
                "mode": "load-balancing",
                "heartbeat-delay": 10000,
                "max-response-delay": 10000,
                "max-ack-delay": 5000,
                "max-unacked-clients": 5,
                "peers": [{
                    "name": "server1",
                    "url": "http://192.168.56.33:8000/",
                    "role": "primary",
                    "auto-failover": true
                }, {
                    "name": "server2",
                    "url": "http://192.168.56.66:8000/",
                    "role": "secondary",
                    "auto-failover": true
                }, {
                    "name": "server3",
                    "url": "http://192.168.56.99:8000/",
                    "role": "backup",
                    "basic-auth-user": "foo",
                    "basic-auth-password": "bar",
                    "auto-failover": false
                }]
            }]
        }
    }],

    "subnet4": [{
        "subnet": "192.0.3.0/24",
        "pools": [{
            "pool": "192.0.3.100 - 192.0.3.150",
            "client-class": "HA_server1"
         }, {
            "pool": "192.0.3.200 - 192.0.3.250",
            "client-class": "HA_server2"
         }],
        "option-data": [{
            "name": "routers",
            "data": "192.0.3.1"
        }],

        "relay": { "ip-address": "10.1.2.3" }
    }]
}
```

Two hooks libraries must be loaded to enable HA: `libdhcp_lease_cmds.so` and `libdhcp_ha.so`. The latter implements the HA feature, while the former enables control commands required by HA to fetch and manipulate leases on the remote servers. In the example provided above, it is assumed that Kea libraries are installed in the `/usr/lib` directory. If Kea is not installed in the /usr directory, the hooks libraries locations must be updated accordingly.

The HA configuration is specified within the scope of `libdhcp_ha.so`. Note that the top-level parameter `high-availability` is a list, even though it currently contains only one entry.

The following are the global parameters which control the server's behavior with respect to HA:

- `this-server-name` - is a unique identifier of the server within this HA setup. It must match with one of the servers specified within the `peers` list.
- `mode` - specifies an HA mode of operation. Currently supported modes are `load-balancing` and `hot-standby`.
- `heartbeat-delay` - specifies a duration in milliseconds between sending the last heartbeat (or other command sent to the partner) and the next heartbeat. The heartbeats are sent periodically to gather the status of the partner and to verify whether the partner is still operating. The default value of this parameter is 10000 ms.
- `max-response-delay` - specifies a duration in milliseconds since the last successful communication with the partner, after which the server assumes that communication with the partner is interrupted. This duration should be greater than the `heartbeat-delay`. Usually it is greater than the duration of multiple `heartbeat-delay` values. When the server detects that communication is interrupted, it may transition to the `partner-down` state (when `max-unacked-clients` is 0) or trigger the failure-detection procedure using the values of the two parameters below. The default value of this parameter is 60000 ms.
- `max-ack-delay` - is one of the parameters controlling partner failure-detection. When communication with the partner is interrupted, the server examines the values of the `secs` field (DHCPv4) or `Elapsed Time` option (DHCPv6), which denote how long the DHCP client has been trying to communicate with the DHCP server. This parameter specifies the maximum time in milliseconds for the client to try to communicate with the DHCP server, after which this server assumes that the client failed to communicate with the DHCP server (is "unacked"). The default value of this parameter is 10000.
- `max-unacked-clients` - specifies how many "unacked" clients are allowed (see `max-ack-delay`) before this server assumes that the partner is offline and transitions to the `partner-down` state. The special value of 0 is allowed for this parameter, which disables the failure-detection mechanism. In this case, a server that can't communicate with its partner over the control channel assumes that the partner server is down and transitions to the `partner-down` state immediately. The default value of this parameter is 10.

The values of `max-ack-delay` and `max-unacked-clients` must be selected carefully, taking into account the specifics of the network in which the DHCP servers are operating. Note that the server in question may not respond to some DHCP clients because these clients are not to be serviced by this server according to administrative policy. The server may also drop malformed queries from clients. Therefore, selecting too low a value for the `max-unacked-clients` parameter may result in a transition to the `partner-down` state even though the partner is still operating. On the other hand, selecting too high a value may result in never transitioning to the `partner-down` state if the DHCP traffic in the network is very low (e.g. at nighttime), because the number of distinct clients trying to communicate with the server could be lower than the `max-unacked-clients` setting.

In some cases it may be useful to disable the failure-detection mechanism altogether, if the servers are located very close to each other and network partitioning is unlikely, i.e. failure to respond to heartbeats is only possible when the partner is offline. In such cases, set the `max-unacked-clients` to 0.

The `peers` parameter contains a list of servers within this HA setup. This configuration must contain at least one primary and one secondary server. It may also contain an unlimited number of backup servers. In this example, there is one backup server which receives lease updates from the active servers.

Since Kea version 1.9.0 the basic HTTP authentication is available to protect the Kea control agent against local attackers.

These are the parameters specified for each of the peers within this list:

- `name` - specifies a unique name for the server.
- `url` - specifies the URL to be used to contact this server over the control channel. Other servers use this URL to send control commands to that server.
- `basic-auth-user` - specifies the user id for basic HTTP authentication. If not specified or specified as an empty string no authentication header will be added to HTTP transactions. Must not contain the colon (:) character.
- `basic-auth-password` - specifies the password for basic HTTP authentication. Ignored when the user id is not specified or empty. The password is optional: if not specified an empty password is used.
- `role` - denotes the role of the server in the HA setup. The following roles are supported in the load-balancing configuration: `primary`, `secondary`, and `backup`. There must be exactly one primary and one secondary server in the load-balancing setup.
- `auto-failover` - a boolean value which denotes whether a server detecting a partner's failure should automatically start serving the partner's clients. The default value of this parameter is true.

In our example configuration, both active servers can allocate leases from the subnet "192.0.3.0/24". This subnet contains two address pools: "192.0.3.100 - 192.0.3.150" and "192.0.3.200 - 192.0.3.250", which are associated with HA server scopes using client classification. When `server1` processes a DHCP query, it uses the first pool for lease allocation. Conversely, when `server2` processes a DHCP query it uses the second pool. When either of the servers is in the `partner-down` state, it can serve leases from both pools and it selects the pool which is appropriate for the received query. In other words, if the query would normally be processed by `server2` but this server is not available, `server1` will allocate the lease from the pool of "192.0.3.200 - 192.0.3.250". The Kea control agent in front of the `server3` requires basic HTTP authentication and authorizes the user id "foo" with the password "bar".

16.15.6 Load Balancing with Advanced Classification

In the previous section, we provided an example of a load-balancing configuration with client classification limited to the `HA_server1` and `HA_server2` classes, which are dynamically assigned to the received DHCP queries. In many cases, HA will be needed in deployments which already use some other client classification.

Suppose there is a system which classifies devices into two groups: phones and laptops, based on some classification criteria specified in the Kea configuration file. Both types of devices are allocated leases from different address pools. Introducing HA in the load-balancing mode results in a further split of each of those pools, as each server allocates leases for some phones and some laptops. This requires each of the existing pools to be split between `HA_server1` and `HA_server2`, so we end up with the following classes:

- phones_server1
- laptops_server1
- phones_server2
- laptops_server2

The corresponding server configuration using advanced classification (and the `member` expression) is provided below. For brevity's sake, the HA hook library configuration has been removed from this example.

```
"Dhcp4": {
    "client-classes": [{
        "name": "phones",
        "test": "substring(option[60].hex,0,6) == 'Aastra'",
    }, {
        "name": "laptops",
        "test": "not member('phones')"
    }, {
        "name": "phones_server1",
        "test": "member('phones') and member('HA_server1')"
    }, {
        "name": "phones_server2",
        "test": "member('phones') and member('HA_server2')"
    }, {
        "name": "laptops_server1",
        "test": "member('laptops') and member('HA_server1')"
    }, {
        "name": "laptops_server2",
        "test": "member('laptops') and member('HA_server2')"
    }],

    "hooks-libraries": [{
        "library": "/usr/lib/kea/hooks/libdhcp_lease_cmds.so",
        "parameters": { }
    }, {
        "library": "/usr/lib/kea/hooks/libdhcp_ha.so",
        "parameters": {
            "high-availability": [{
                ...
            }]
        }
    }],

    "subnet4": [{
        "subnet": "192.0.3.0/24",
        "pools": [{
            "pool": "192.0.3.100 - 192.0.3.125",
            "client-class": "phones_server1"
        }, {
            "pool": "192.0.3.126 - 192.0.3.150",
            "client-class": "laptops_server1"
        }, {
            "pool": "192.0.3.200 - 192.0.3.225",
            "client-class": "phones_server2"
        }, {
            "pool": "192.0.3.226 - 192.0.3.250",
            "client-class": "laptops_server2"
        }],

        "option-data": [{
            "name": "routers",
            "data": "192.0.3.1"
        }],

        "relay": { "ip-address": "10.1.2.3" }
    }],
}
```

The configuration provided above splits the address range into four pools: two pools dedicated to server1 and two to server2. Each server can assign leases to both phones and laptops. Both groups of devices are assigned addresses from different pools. The `HA_server1` and `HA_server2` classes are built-in (see *Built-in Client Classes*) and do not need to be declared. They are assigned dynamically by the HA hook library as a result of the load-balancing algorithm. `phones_*` and `laptop_*` evaluate to "true" when the query belongs to a given combination of other classes, e.g. `HA_server1` and `phones`. The pool is selected accordingly as a result of such an evaluation.

Consult *Client Classification* for details on how to use the `member` expression and class dependencies.

16.15.7 Hot-Standby Configuration

The following is an example configuration of the primary server in the hot-standby configuration:

```
"Dhcp4": {
    "hooks-libraries": [{
        "library": "/usr/lib/kea/hooks/libdhcp_lease_cmds.so",
        "parameters": { }
    }, {
        "library": "/usr/lib/kea/hooks/libdhcp_ha.so",
        "parameters": {
            "high-availability": [{
                "this-server-name": "server1",
                "mode": "hot-standby",
                "heartbeat-delay": 10000,
                "max-response-delay": 10000,
                "max-ack-delay": 5000,
                "max-unacked-clients": 5,
                "peers": [{
                    "name": "server1",
                    "url": "http://192.168.56.33:8000/",
                    "role": "primary",
                    "auto-failover": true
                }, {
                    "name": "server2",
                    "url": "http://192.168.56.66:8000/",
                    "role": "standby",
                    "auto-failover": true
                }, {
                    "name": "server3",
                    "url": "http://192.168.56.99:8000/",
                    "basic-auth-user": "foo",
                    "basic-auth-password": "bar",
                    "role": "backup",
                    "auto-failover": false
                }]
            }]
        }
    }],

    "subnet4": [{
        "subnet": "192.0.3.0/24",
        "pools": [{
            "pool": "192.0.3.100 - 192.0.3.250",
            "client-class": "HA_server1"
        }],
        "option-data": [{
            "name": "routers",
            "data": "192.0.3.1"
        }],

        "relay": { "ip-address": "10.1.2.3" }
    }]
}
```

This configuration is very similar to the load-balancing configuration described in *Load-Balancing Configuration*, with a few notable differences.

The `mode` is now set to `hot-standby`, in which only one server responds to DHCP clients. If the primary server is online, it responds to all DHCP queries. The `standby` server takes over all DHCP traffic if it discovers that the primary is unavailable.

In this mode, the non-primary active server is called `standby` and that is its role.

Finally, because there is always one server responding to DHCP queries, there is only one scope - `HA_server1` - in use within pools definitions. In fact, the `client-class` parameter could be removed from this configuration without harm, because there can be no conflicts in lease allocations by different servers as they do not allocate leases concurrently. The `client-class` remains in this example mostly for demonstration purposes, to highlight the differences between the hot-standby and load-balancing modes of operation.

16.15.8 Passive-Backup Configuration

The following is an example configuration of the primary server in the passive-backup configuration:

```
"Dhcp4": {
    "hooks-libraries": [{
        "library": "/usr/lib/kea/hooks/libdhcp_lease_cmds.so",
        "parameters": { }
    }, {
        "library": "/usr/lib/kea/hooks/libdhcp_ha.so",
        "parameters": {
            "high-availability": [{
                "this-server-name": "server1",
                "mode": "passive-backup",
                "wait-backup-ack": false,
                "peers": [{
                    "name": "server1",
                    "url": "http://192.168.56.33:8000/",
                    "role": "primary"
                }, {
                    "name": "server2",
                    "url": "http://192.168.56.66:8000/",
                    "role": "backup"
                }, {
                    "name": "server3",
                    "url": "http://192.168.56.99:8000/",
                    "basic-auth-user": "foo",
                    "basic-auth-password": "bar",
                    "role": "backup"
                }]
            }]
        }
    }],

    "subnet4": [{
        "subnet": "192.0.3.0/24",
        "pools": [{
            "pool": "192.0.3.100 - 192.0.3.250",
        }],

        "option-data": [{
            "name": "routers",
            "data": "192.0.3.1"
        }],

        "relay": { "ip-address": "10.1.2.3" }
    }]
}
```

The configurations of three peers are included, one for the primary and two for the backup servers. Many of the parameters present in the load-balancing and hot-standby configuration examples are not relevant in the passive-backup mode, thus they are not specified here. For example: `heartbeat-delay`, `max-unacked-clients` and others related to the automatic failover mechanism should not be specified in the passive-backup mode. The `wait-backup-ack` is a boolean parameter not present in previous examples. It defaults to `false` and must not be modified in the load-balancing and hot-standby modes. In the passive-backup mode this parameter can be set to `true` which causes the primary server to expect acknowledgments to the lease updates from the backup servers prior to responding to the DHCP client. It ensures that the lease was propagated to all servers before the client is given the lease, but it poses a risk of loosing a DHCP service if there is a communication problem with one of the backup servers. This setting also increases the latency of the DHCP response because of the time that the primary spends waiting for the acknowledgements. We recommend that the `wait-backup-ack` is left at its default value if the DHCP service reliability is more important than consistency of the lease information between the primary and the backups and in all cases when the DHCP service latency should be minimal.

16.15.9 Lease Information Sharing

An HA-enabled server informs its active partner about allocated or renewed leases by sending appropriate control commands, and the partner updates the lease information in its own database. When the server starts up for the first time or recovers after a failure, it synchronizes its lease database with its partner. These two mechanisms guarantee consistency of the lease information between the servers and allow the designation of one of the servers to handle the entire DHCP traffic load if the other server becomes unavailable.

In some cases, though, it is desirable to disable lease updates and/or database synchronization between the active servers, if the exchange of information about the allocated leases is performed using some other mechanism. Kea supports various database types that can be used to store leases, including MySQL, PostgreSQL, and Cassandra. Those databases include built-in solutions for data replication which are often used by Kea administrators to provide redundancy.

The HA hook library supports such scenarios by disabling lease updates over the control channel and/or lease database synchronization, leaving the server to rely on the database replication mechanism. This is controlled by the two boolean parameters `send-lease-updates` and `sync-leases`, whose values default to true:

```
{
"Dhcp4": {

    ...

    "hooks-libraries": [
        {
            "library": "/usr/lib/kea/hooks/libdhcp_lease_cmds.so",
            "parameters": { }
        },
        {
            "library": "/usr/lib/kea/hooks/libdhcp_ha.so",
            "parameters": {
                "high-availability": [ {
                    "this-server-name": "server1",
                    "mode": "load-balancing",
                    "send-lease-updates": false,
                    "sync-leases": false,
                    "peers": [
                        {
                            "name": "server1",
                            "url": "http://192.168.56.33:8000/",
                            "role": "primary"
                        },
                        {
                            "name": "server2",
                            "url": "http://192.168.56.66:8000/",
                            "role": "secondary"
                        }
                    ]
                } ]
            }
        }
    ],

    ...

}
```

In the most typical use case, both parameters are set to the same value, i.e. both are `false` if database replication is in use, or both are `true` otherwise. Introducing two separate parameters to control lease updates and lease-database synchronization is aimed at possible special use cases; for example, when synchronization is performed by copying a lease file (therefore `sync-leases` is set to `false`), but lease updates should be conducted as usual (`send-lease-updates` is set to `true`). It should be noted that Kea does not natively support such use cases, but users may develop their own scripts and tools around Kea to provide such mechanisms. The HA hooks library configuration is designed to maximize flexibility of administration.

16.15.10 Controlling Lease-Page Size Limit

An HA-enabled server initiates synchronization of the lease database after downtime or upon receiving the `ha-sync` command. The server uses commands described in *The lease4-get-page, lease6-get-page Commands* and *The lease4-get-page, lease6-get-page Commands* to fetch leases from its partner server (lease queries). The size of the results page (the maximum number of leases to be returned in a single response to one of these commands) can be controlled via configuration of the HA hooks library. Increasing the page size decreases the number of lease queries sent to the partner server, but it causes the partner server to generate larger responses, which lengthens transmission time as well as increases memory and CPU utilization on both servers. Decreasing the page size helps to decrease resource utilization, but requires more lease queries to be issued to fetch the entire lease database.

The default value of the `sync-page-limit` command controlling the page size is 10000. This means that the entire lease database can be fetched with a single command if the size of the database is equal to or less than 10000 lines.

16.15.11 Timeouts

In deployments with a large number of clients connected to the network, lease-database synchronization after a server failure may be a time-consuming operation. The synchronizing server must gather all leases from its partner, which yields a large response over the RESTful interface. The server receives leases using the paging mechanism described in *Controlling Lease-Page Size Limit*. Before the page of leases is fetched, the synchronizing server sends a `dhcp-disable` command to disable the DHCP service on the partner server. If the service is already disabled, this command will reset the timeout for the DHCP service being disabled. This timeout value is by default set to 60 seconds. If fetching a single page of leases takes longer than the specified time, the partner server will assume that the synchronizing server died and will resume its DHCP service. The connection of the synchronizing server with its partner is also protected by the timeout. If the synchronization of a single page of leases takes longer than the specified time, the synchronizing server terminates the connection and the synchronization fails. Both timeout values are controlled by a single configuration parameter, `sync-timeout`. The following configuration snippet demonstrates how to modify the timeout for automatic re-enabling of the DHCP service on the partner server and how to increase the timeout for fetching a single page of leases from 60 seconds to 90 seconds:

```
{
"Dhcp4": {

    ...

    "hooks-libraries": [
        {
            "library": "/usr/lib/kea/hooks/libdhcp_lease_cmds.so",
            "parameters": { }
        },
        {
            "library": "/usr/lib/kea/hooks/libdhcp_ha.so",
            "parameters": {
                "high-availability": [ {
                    "this-server-name": "server1",
                    "mode": "load-balancing",
                    "sync-timeout": 90000,
                    "peers": [
                        {
                            "name": "server1",
                            "url": "http://192.168.56.33:8000/",
                            "role": "primary"
                        },
                        {
                            "name": "server2",
                            "url": "http://192.168.56.66:8000/",
                            "role": "secondary"
                        }
                    ]
                } ]
            }
        }
    ],

    ...

}
```

It is important to note that extending this `sync-timeout` value may sometimes be insufficient to prevent issues with timeouts during lease-database synchronization. The control commands travel via the Control Agent, which also monitors incoming (with a synchronizing server) and outgoing (with a DHCP server) connections for timeouts. The

DHCP server also monitors the connection from the Control Agent for timeouts. Those timeouts cannot currently be modified via configuration; extending these timeouts is only possible by modifying them in the Kea code and recompiling the server. The relevant constants are located in the Kea source at: `src/lib/config/timeouts.h`.

16.15.12 Pausing the HA State Machine

The high-availability state machine includes many different states described in detail in *Server States*. The server enters each state when certain conditions are met, most often taking into account the partner server's state. In some states the server performs specific actions, e.g. synchronization of the lease database in the `syncing` state or responding to DHCP queries according to the configured mode of operation in the `load-balancing` and `hot-standby` states.

By default, transitions between the states are performed automatically and the server administrator has no direct control when the transitions take place; in most cases, the administrator does not need such control. In some situations, however, the administrator may want to "pause" the HA state machine in a selected state to perform some additional administrative actions before the server transitions to the next state.

Consider a server failure which results in the loss of the entire lease database. Typically, the server will rebuild its lease database when it enters the `syncing` state by querying the partner server for leases, but it is possible that the partner was also experiencing a failure and lacks lease information. In this case, it may be required to reconstruct lease databases on both servers from some external source, e.g. a backup server. If the lease database is to be reconstructed via the RESTful API, the servers should be started in the initial, i.e. `waiting`, state and remain in this state while leases are being added. In particular, the servers should not attempt to synchronize their lease databases nor start serving DHCP clients.

The HA hooks library provides configuration parameters and a command to control when the HA state machine should be paused and resumed. The following configuration causes the HA state machine to pause in the `waiting` state after server startup.

```
"Dhcp4": {

    ...

    "hooks-libraries": [
        {
            "library": "/usr/lib/kea/hooks/libdhcp_lease_cmds.so",
            "parameters": { }
        },
        {
            "library": "/usr/lib/kea/hooks/libdhcp_ha.so",
            "parameters": {
                "high-availability": [ {
                    "this-server-name": "server1",
                    "mode": "load-balancing",
                    "peers": [
                        {
                            "name": "server1",
                            "url": "http://192.168.56.33:8000/",
                            "role": "primary"
                        },
                        {
                            "name": "server2",
                            "url": "http://192.168.56.66:8000/",
                            "role": "secondary"
                        }
                    ],
                    "state-machine": {
                        "states":  [
                            {
                                "state": "waiting",
                                "pause": "once"
                            }
                        ]
                    }
                } ]
            }
        }
    ],

    ...

}
```

The `pause` parameter value `once` denotes that the state machine should be paused upon the first transition to the `waiting` state; later transitions to this state will not cause the state machine to pause. Two other supported values of the `pause` parameter are `always` and `never`. The latter is the default value for each state, which instructs the server never to pause the state machine.

In order to "unpause" the state machine, the `ha-continue` command must be sent to the paused server. This command does not take any arguments. See *Control Commands for High Availability* for details about commands specific to the HA hooks library.

It is possible to configure the state machine to pause in more than one state. Consider the following configuration:

```
"Dhcp4": {

    ...

    "hooks-libraries": [
        {
            "library": "/usr/lib/kea/hooks/libdhcp_lease_cmds.so",
            "parameters": { }
        },
        {
            "library": "/usr/lib/kea/hooks/libdhcp_ha.so",
            "parameters": {
                "high-availability": [ {
                    "this-server-name": "server1",
                    "mode": "load-balancing",
                    "peers": [
                        {
                            "name": "server1",
                            "url": "http://192.168.56.33:8000/",
                            "role": "primary"
                        },
                        {
                            "name": "server2",
                            "url": "http://192.168.56.66:8000/",
                            "role": "secondary"
                        }
                    ],
                    "state-machine": {
                        "states": [
                            {
                                "state": "ready",
                                "pause": "always"
                            },
                            {
                                "state": "partner-down",
                                "pause": "once"
                            }
                        ]
                    }
                } ]
            }
        }
    ],

    ...

}
```

This configuration instructs the server to pause the state machine every time it transitions to the `ready` state and upon the first transition to the `partner-down` state.

Refer to *Server States* for a complete list of server states. The state machine can be paused in any of the supported states; however, it is not practical for the `backup` and `terminated` states because the server never transitions out of these states anyway.

Note: In the `syncing` state the server is paused before it makes an attempt to synchronize the lease database with a partner. To pause the state machine after lease-database synchronization, use the `ready` state instead.

Note: The state of the HA state machine depends on the state of the cooperating server. Therefore, it must be taken into account that pausing the state machine of one server may affect the operation of the partner server. For example: if the primary server is paused in the `waiting` state, the partner server will also remain in the `waiting` state until the state machine of the primary server is resumed and that server transitions to the `ready` state.

16.15.13 Control Agent Configuration

The Kea Control Agent describes in detail the Kea daemon, which provides a RESTful interface to control the Kea servers. The same functionality is used by the High Availability hooks library to establish communication between the HA peers. Therefore, the HA library requires that the Control Agent (CA) be started for each DHCP instance within the HA setup. If the Control Agent is not started, the peers will not be able to communicate with the particular DHCP server (even if the DHCP server itself is online) and may eventually consider this server to be offline.

The following is an example configuration for the CA running on the same machine as the primary server. This configuration is valid for both the load-balancing and the hot-standby cases presented in previous sections.

```
{
"Control-agent": {
    "http-host": "192.168.56.33",
    "http-port": 8000,

    "control-sockets": {
        "dhcp4": {
            "socket-type": "unix",
            "socket-name": "/tmp/kea-dhcp4-ctrl.sock"
        },
        "dhcp6": {
            "socket-type": "unix",
            "socket-name": "/tmp/kea-dhcp6-ctrl.sock"
        }
    }
}
}
```

Since Kea version 1.9.0 the basic HTTP authentication is supported.

16.15.14 Controlled Shutdown and Maintenance of DHCP servers

Having a pair of servers providing High Availability allows for controlled shutdown and maintenance of those servers without disrupting the DHCP service. For example, an administrator can perform an upgrade of one of the servers while the other one continues to respond to the DHCP queries. When the upgraded server is back online, the upgrade can be performed for the second server. The typical problem reported for the earlier versions of the High Availability hooks library was that the administrator did not have a direct control over the state of the DHCP server. Shutting down one of the servers for maintenance didn't necessarily cause the other server to start reponding to all DHCP queries because the failure detection algorithm described in *Scope Transition in a Partner-Down Case* requires that the partner does not respond for a configured period of time and, depending on the configuration, may also require that a number of DHCP requests are not responded for a configured period of time. The maintenance procedure, however, requires that the administrator is able to instruct one of the servers to instantly start serving all DHCP clients and the other server to instantly stop serving any DHCP clients so as it can be safely shut down.

The maintenance feature of the High Availability hooks library addresses this problem. The `ha-maintenance-start` command was introduced to allow the administrator to put the pair of the active servers in states in which one of them is responding to all DHCP queries and the other one is awaiting a shutdown.

Suppose that the HA setup includes two active servers, e.g. `server1` and `server2` and the latter needs to be shut down for the maintenance. The administrator should send the `ha-maintenance-start` to server1, as this is the server which is going to handle the DHCP traffic while the other one is offline. The server1 may respond with an error if its state or the partner's state does not allow for the maintenance. For example, the maintenance is not supported for the backup server or the server being in the terminated state. Also, an error will be returned if the maintenance request was already sent to the other server.

Upon receiving the `ha-maintenance-start` command, the server1 will send the `ha-maintenance-notify` command to the server2 to put this server in the `in-maintenance` state. If the server2 confirms, the server1 will transition to the `partner-in-maintenance` state. This is similar to the `partner-down` state, except that in the `partner-in-maintenance` state the server1 continues to send lease updates to the server2 until the administrator shuts down the server2. The server1 now responds to all DHCP queries.

The administrator may safely shut down the server2 being in the `in-maintenance` state and perform necessary maintenance actions. When the server2 is offline, the server1 will encounter communication issues with the partner and will immediately transition to the `partner-down` state in which it will continue to respond to all DHCP queries but will no longer send lease updates to the server2. Starting the server2 after the maintenance will trigger normal state negotiation, lease database synchronization and, ultimately, a transition to the load-balancing or hot-standby state. The maintenance can now be performed for the server1. It should be initiated by sending the `ha-maintenance-start` to the server2.

If the `ha-maintenance-start` command was sent to the server and the server has transitioned to the `partner-in-maintenance` state it is possible to transition it and its partner back to the previous states to resume the normal operation of the HA pair. This is achieved by sending the `ha-maintenance-cancel` command to the server being in the `partner-in-maintenance` state. However, if the server has already transitioned to the `partner-down` state as a result of detecting that the partner is offline, canceling the maintenance is no longer possible.

16.15.15 Upgrading from Older HA Versions

The maintenance mechanism was first introduced in the Kea 1.7.4 release. In order to upgrade the HA hooks library from the older version the administrator must shut down one of the servers and rely on the failover mechanism to get the online server to transition to the partner-down state and start serving all DHCP clients. After the successful upgrade of one of the servers to the version supporting the maintenance mechanism it is possible to benefit from this mechanism during the upgrade of the second server.

In such case, shut down the server running the old version. Next, send the `ha-maintenance-start` to the server that has been upgraded and supports the maintenance mechanism. This server should immediately transition to the partner-down state as it cannot communicate with the partner being offline. In the partner-down state the server will be responding to all DHCP requests.

Note: Do not send the `ha-maintenance-start` command while the server running the old version is still online. The server receiving this command will return an error seeing that the partner does not support the maintenance mechanism.

16.15.16 Control Commands for High Availability

Even though the HA hook library is designed to automatically resolve issues with DHCP service interruptions by redirecting the DHCP traffic to a surviving server and synchronizing the lease database when required, it may be useful for the administrator to have more control over the server behavior. In particular, it may be useful to be able to trigger lease-database synchronization on demand. It may also be useful to manually set the HA scopes that are being served.

Note that the backup server can sometimes be used to handle DHCP traffic if both active servers are down. The backup server does not perform failover function automatically; thus, in order to use the backup server to respond to DHCP queries, the server administrator must enable this function manually.

The following sections describe commands supported by the HA hooks library which are available for the administrator.

The ha-sync Command

The `ha-sync` command instructs the server to synchronize its local lease database with the selected peer. The server fetches all leases from the peer and updates those locally stored leases which are older than those fetched. It also creates new leases when any of those fetched do not exist in the local database. All leases that are not returned by the peer but are in the local database are preserved. The database synchronization is unidirectional; only the database on the server to which the command has been sent is updated. In order to synchronize the peer's database a separate `ha-sync` command must be issued to that peer.

Database synchronization may be triggered for both active and backup server types. The `ha-sync` command has the following structure (DHCPv4 server case):

```
{
    "command": "ha-sync",
    "service": [ "dhcp4 "],
    "arguments": {
        "server-name": "server2",
        "max-period": 60
    }
}
```

When the server receives this command it first disables the DHCP service of the server from which it will be fetching leases, by sending the `dhcp-disable` command to that server. The `max-period` parameter specifies the maximum duration (in seconds) for which the DHCP service should be disabled. If the DHCP service is successfully disabled, the synchronizing server fetches leases from the remote server by issuing one or more `lease4-get-page` commands. When the lease-database synchronization is complete, the synchronizing server sends the `dhcp-enable` command to the peer to re-enable its DHCP service.

The `max-period` value should be sufficiently long to guarantee that it does not elapse before the synchronization is completed. Otherwise, the DHCP server will automatically enable its DHCP function while the synchronization is still in progress. If the DHCP server subsequently allocates any leases during the synchronization, those new (or updated) leases will not be fetched by the synchronizing server, leading to database inconsistencies.

The ha-scopes Command

This command allows modification of the HA scopes that the server is serving. Consult *Load-Balancing Configuration* and *Hot-Standby Configuration* to learn what scopes are available for different HA modes of operation. The `ha-scopes` command has the following structure (DHCPv4 server case):

```
{
    "command": "ha-scopes",
    "service": [ "dhcp4" ],
    "arguments": {
        "scopes": [ "HA_server1", "HA_server2" ]
    }
}
```

This command configures the server to handle traffic from both the `HA_server1` and `HA_server2` scopes. To disable all scopes specify an empty list:

```
{
    "command": "ha-scopes",
    "service": [ "dhcp4 "],
    "arguments": {
        "scopes": [ ]
    }
}
```

The ha-continue Command

This command is used to resume the operation of the paused HA state machine, as described in *Pausing the HA State Machine*. It takes no arguments, so the command structure is as simple as:

```
{
    "command": "ha-continue",
    "service": [ "dhcp4" ]
}
```

The ha-heartbeat Command

The *Server States* describes how the `ha-heartbeat` command is used by the active HA servers to detect a failure of one of them. This command, however, can also be sent by the system administrator to one or both servers to check their state with regards to the HA relationship. This allows for hooking up a monitoring system to the HA enabled servers to periodically check if they are operational or if any manual intervention is required. The `ha-heartbeat` command takes no arguments, e.g.:

```
{
    "command": "ha-heartbeat",
    "service": [ "dhcp4" ]
}
```

Upon successful communication with the server a response similar to this should be returned:

```
{
   "result": 0,
   "text": "HA peer status returned.",
   "arguments":
       {
           "state": "partner-down",
           "date-time": "Thu, 07 Nov 2019 08:49:37 GMT"
       }
}
```

The returned state value may be one of the values listed in *Server States*. In the example above the `partner-down` state is returned, which indicates that the server which responded to the command is assuming that its partner is offline, thus it is serving all DHCP requests sent to the servers. In order to ensure that the partner is indeed offline the administrator should send the `ha-heartbeat` command to the second server. If sending the command fails, e.g. as a result of inability to establish TCP connection to the Control Agent or the Control Agent reports issues with communication with the DHCP server, it is very likely that the server is not running.

The typical response returned by one of the servers when both servers are operational is:

```
{
   "result": 0,
   "text": "HA peer status returned.",
   "arguments":
       {
           "state": "load-balancing",
           "date-time": "Thu, 07 Nov 2019 08:49:37 GMT"
       }
}
```

In most cases it is desired to send the `ha-heartbeat` command to both HA enabled servers to verify the state of the entire HA setup. In particular, if the response sent to one of the servers indicates that the server is in the `load-balancing` state, it merely means that this server is operating as if the partner is still functional. When the partner dies it actually takes some time for the surviving server to realize it. The *Scope Transition in a Partner-Down Case* section describes the algorithm which the surviving server follows before it transitions to the `partner-down` state. If the `ha-heartbeat` command is sent during the time window between the failure of one of the servers and the transition of the surviving server to the `partner-down` state, the response from the surviving server doesn't reflect the failure. Sending the command to the failing server allows for detecting the failure.

Note: Remember! Always send the `ha-heartbeat` command to both active HA servers to check the state of the entire HA setup. Sending it to only one of the servers may not reflect issues with one of the servers that just began.

The status-get Command

The `status-get` is the general purpose command supported by several Kea deamons, not only DHCP servers. However, when sent to the DHCP server with HA enabled, it can be used to get insight into the details of the HA specific status information of the servers being in the HA configuration. Not only does the response contain the status information of the server receiving this command but also the information about its partner, if this information is available.

The following is the example response to the `status-get` command including the HA status of two load balancing servers:

```
{
    "result": 0,
    "text": "",
    "arguments": {
        "pid": 1234,
        "uptime": 3024,
        "reload": 1111,
        "high-availability": [
            {
                "ha-mode": "load-balancing",
                "ha-servers": {
                    "local": {
                        "role": "primary",
                        "scopes": [ "server1" ],
                        "state": "load-balancing"
                    },
                    "remote": {
                        "age": 10,
                        "in-touch": true,
                        "role": "secondary",
                        "last-scopes": [ "server2" ],
                        "last-state": "load-balancing",
                        "communication-interrupted": true,
                        "connecting-clients": 2,
                        "unacked-clients": 1,
                        "unacked-clients-left": 2,
                        "analyzed-packets": 8
                    }
                }
            }
        ],
        "multi-threading-enabled": true,
        "thread-pool-size": 4,
        "packet-queue-size": 64
    }
}
```

The `high-availability` argument is a list which currently always comprises one element. There are plans to extend the HA implementation to facilitate multiple HA relationships for a single server instance. In that case, the returned list will comprise more elements, each describing the status of a different relationship in which the server participates. Currently, it is only one status.

Note: In Kea 1.7.8 an incompatible change was introduced to the syntax of the `status-get` response. Previously, the HA status for a single relationship was returned within the `arguments` map. As of Kea 1.7.8, the returned status is enclosed in the list as described above. Any existing code relying on the previous syntax must be updated to work with the new Kea versions.

The `ha-servers` map contains two structures: `local` and `remote`. The former contains the status information of the server which received the command. The latter contains the status information known to the local server about the partner. The `role` of the partner server is gathered from the local configuration file, therefore it should always be available. The remaining status information such as `last-scopes` and `last-state` is not available until the local server communicates with the remote by successfully sending the `ha-heartbeat` command. If at least one such communication took place, the returned value of `in-touch` parameter is set to `true`. By examining this value, the command sender can determine whether the information about the remote server is reliable.

The `last-scopes` and `last-state` contain the information about the HA scopes served by the partner and its state. Note that this information is gathered during the heartbeat command exchange, so it may not be accurate if the communication problem occur between the partners and this status information is not refreshed. In such case, it may be useful to send the `status-get` command to the partner server directly to check its current state. The `age` parameter specifies the number of seconds since the information from the partner was gathered (the age of this information).

The `communication-interrupted` boolean value indicates if the server receiving the `status-get` command (local server) has been unable to communicate with the partner longer than the duration specified as `max-response-delay`. In such a situation we say that active servers are in the communication interrupted state or that the communication between them is interrupted. At this point, the local server may start monitoring the DHCP traffic directed to the partner to see if the partner is responding to this traffic. More about the failover procedure can be found in *Load-Balancing Configuration*.

The `connecting-clients`, `unacked-clients`, `unacked-clients-left` and `analyzed-packets` parameters have been introduced together with the `communication-interrupted` parameter in the Kea 1.7.8 release and they convey useful information about the state of the DHCP traffic monitoring in the communication interrupted state. If the server leaves the communication interrupted state these parameters are all reset to 0.

These parameters have the following meaning in the communication interrupted state:

- `connecting-clients` - number of different clients which have attempted to get a lease from the remote server. The clients are differentiated by their MAC address and client identifier (in DHCPv4) or DUID (in DHCPv6). This number includes both “unacked” clients (for which “secs” field or “elapsed time” value exceeded the `max-response-delay`).
- `unacked-clients` - number of different clients which have been considered “unacked”, i.e. the clients which have been trying to get the lease long enough, so as the value of the “secs” field or “elapsed time” exceeded the `max-response-delay`.
- `unacked-clients-left` - number of additional clients which have to be considered “unacked” before the server enters the partner-down state. This value decreases when the `unacked-clients` value increases. The local server will enter the `partner-down` state when this value decreases to 0.
- `analyzed-packets` - total number of all packets directed to the partner server and analyzed by the local server since entering the communication interrupted state. It includes retransmissions from the same clients.

Monitoring these values helps to predict when the local server will enter the partner-down state or why the server hasn’t yet entered this state.

The last parameter introduced in the Kea 1.7.8 release was the `ha-mode`. It returns the HA mode of operation selected using the `mode` parameter in the configuration file. It can hold one of the following values: `load-balancing`, `hot-standby` or `passive-backup`.

The `status-get` response has the format described above only in the `load-balancing` and `hot-standby` modes. In the `passive-backup` mode the `remote` map is not included in the response because in this mode there is only one active server (local). The response comprises no information about the status of the backup servers.

The ha-maintenance-start Command

This command is used to initiate transition of the server’s partner into the `in-maintenance` state and the transition of the server receiving the command into the `partner-in-maintenance` state. See the *Controlled Shutdown and Maintenance of DHCP servers* for the details.

```
{
    "command": "ha-maintenance-start",
    "service": [ "dhcp4" ]
}
```

The ha-maintenance-cancel Command

This command is used to cancel the maintenance previously initiated using the `ha-maintenance-start` command. The server receiving this command will first send the `ha-maintenance-notify` with the cancel flag set to true to its partner. Next, the server will revert from the `partner-in-maintenance` state to the previous state. See the *Controlled Shutdown and Maintenance of DHCP servers* for the details.

```
{
    "command": "ha-maintenance-cancel",
    "service": [ "dhcp4" ]
}
```

The ha-maintenance-notify Command

This command is sent by the server receiving the `ha-maintenance-start` or the `ha-maintenance-cancel` command to its partner to cause the partner to transition to the `in-maintenance` state or to revert from this state to a previous state. See the *Controlled Shutdown and Maintenance of DHCP servers* for the details.

```
{
    "command": "ha-maintenance-notify",
    "service": [ "dhcp4" ],
    "arguments": {
        "cancel": false
    }
}
```

Warning: The `ha-maintenance-notify` command is not meant to be used by the system administrators. It is used for internal communication between a pair of HA enabled DHCP servers. Direct use of this command is not supported and may produce unintended consequences.

16.16 stat_cmds: Supplemental Statistics Commands

This library provides additional commands for retrieving lease statistics from Kea DHCP servers. These commands were added to address an issue with obtaining accurate lease statistics in deployments running multiple Kea servers that use a shared lease backend. The in-memory statistics kept by individual servers only track lease changes made by that server; thus, in a deployment with multiple servers (e.g. two kea-dhcp6 servers using the same PostgreSQL database for lease storage), these statistics are incomplete. The MySQL and PostgreSQL backends in Kea track lease allocation changes as they occur via database triggers. Additionally, all four lease backends were extended to support retrieving lease statistics for all subnets, a single subnet, or a range of subnets. Finally, this library was constructed to provide commands for retrieving these statistics.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

The commands currently provided by this library are:

- `stat-lease4-get` - fetches DHCPv4 lease statistics.
- `stat-lease6-get` - fetches DHCPv6 lease statistics.

The stat commands library is part of the open source code and is available to every Kea user.

All commands use JSON syntax and can be issued directly to the servers via either the control channel (see *Management API*) or the Control Agent (see *The Kea Control Agent*).

This library may be loaded by both the kea-dhcp4 and kea-dhcp6 servers. It is loaded in the same way as other libraries and currently has no parameters:

```
"Dhcp6": {
    "hooks-libraries": [
        {
            "library": "/path/libdhcp_stat_cmds.so"
        }
        ...
    ]
}
```

In a deployment with multiple Kea DHCP servers sharing a common lease storage, this hooks library may be loaded by any or all of the servers. However, one thing to keep in mind is that a server's response to a stat-lease[46]-get command will only contain data for subnets known to that server. In other words, if a subnet does not appear in a server's configuration, Kea will not retrieve statistics for it.

16.16.1 The stat-lease4-get, stat-lease6-get Commands

The `stat-lease4-get` and `stat-lease6-get` commands fetch lease statistics for a range of known subnets. The range of subnets is determined through the use of optional command input parameters:

- `subnet-id` - the ID of the subnet for which lease statistics should be fetched. Use this to get statistics for a single subnet. If the subnet does not exist, the command result code is 3 (i.e. CONTROL_RESULT_EMPTY).
- `subnet-range` - a pair of subnet IDs which describe an inclusive range of subnets for which statistics should be retrieved. The range may include one or more IDs that correspond to no subnet; in this case, the command will only output lease statistics for those that exist. However, if the range does not include any known subnets, the command result code is 3 (i.e. CONTROL_RESULT_EMPTY).
    - `first-subnet-id` - the ID of the first subnet in the range.
    - `last-subnet-id` - the ID of the last subnet in the range.

The use of subnet-id and subnet-range are mutually exclusive. If no parameters are given, the result will contain data for all known subnets. Note that in configurations with large numbers of subnets, this can result in a large response.

The following command fetches lease statistics for all known subnets from a kea-dhcp4 server:

```
{
  "command": "stat-lease4-get"
}
```

The following command fetches lease statistics for subnet ID 10 from a kea-dhcp6 server:

```
{
  "command": "stat-lease6-get",
  "arguments": {
    "subnet-id" : 10
  }
}
```

The following command fetches lease statistics for all subnets with IDs in the range 10 through 50 from a kea-dhcp4 server:

```
{
  "command": "stat-lease4-get",
  "arguments": {
    "subnet-range" {
      "first-subnet-id": 10,
      "last-subnet-id": 50,
    }
  }
}
```

The response to either command will contain three elements:

- `result` - a numeric value indicating the outcome of the command where:
  - `0` - the command was successful;
  - `1` - an error occurred, and an explanation will be the "text" element; or
  - `2` - the fetch found no matching data.
- `text` - an explanation of the command outcome. When the command succeeds it will contain the command name along with the number of rows returned.
- `arguments` - a map containing the data returned by the command as the element "result-set", which is patterned after SQL statement responses:
  - `columns` - a list of text column labels. The columns returned for DHCPv4 are:
    - `subnet-id` - the ID of the subnet.
    - `total-addresses` - the total number of addresses available for DHCPv4 management in the subnet. In other words, this is the sum of all addresses in all the configured pools in the subnet.
    - `cumulative-assigned-addresses` - the cumulative number of addresses in the subnet that have been assigned to a client by the server since it started.
    - `assigned-addresses` - the number of addresses in the subnet that are currently assigned to a client.
    - `declined-addresses` - the number of addresses in the subnet that are currently declined and are thus unavailable for assignment.
  - The columns returned for DHCPv6 are:
    - `subnet-id` - the ID of the subnet.
    - `total-nas` - the number of NA addresses available for DHCPv6 management in the subnet. In other words, this is the sum of all the NA addresses in all the configured NA pools in the subnet.
    - `cumulative-assigned-nas` - the cumulative number of NA addresses in the subnet that have been assigned to a client by the server since it started.
    - `assigned-nas` - the number of NA addresses in the subnet that are currently assigned to a client.
    - `declined-nas` - the number of NA addresses that are currently declined and are thus unavailable for assignment.

    - `total-pds` - the total number of PD prefixes available of DHCPv6 management in the subnet. In other words, this is the sum of all prefixes in all the configured prefix pools in the subnet.
    - `cumulative-assigned-pds` - the cumulative number of PD prefixes in the subnet that have been assigned to a client by the server since it started.
    - `assigned-pds` - the number of PD prefixes in the subnet that are currently assigned to a client.
  - `rows` - a list of rows, one per subnet ID. Each row contains a data value corresponding to and in the same order as each column listed in "columns" for a given subnet.
  - `timestamp` - the textual date and time the data were fetched, expressed as GMT.

The response to a DHCPv4 command might look as follows:

```
{
  "result": 0,
  "text": "stat-lease4-get: 2 rows found",
  "arguments": {
    "result-set": {
      "columns": [ "subnet-id", "total-addresses", "cumulative-assigned-addresses",
↪"assigned-addresses", "declined-addresses" ]
      "rows": [
        [ 10, 256, 300, 111, 0 ],
        [ 20, 4098, 2034, 2034, 4 ]
      ],
    "timestamp": "2018-05-04 15:03:37.000000"
    }
  }
}
```

The response to a DHCPv6 command might look as follows (subnet 10 has no prefix pools, subnet 20 has no NA pools, and subnet 30 has both NA and PD pools):

```
{
  "result": 0,
  "text": "stat-lease6-get: 2 rows found",
  "arguments": {
    "result-set": {
      "columns": [ "subnet-id", "total-nas", "cumulative-assigned-nas", "assigned-nas
↪", "declined-nas", "total-pds", "cumulative-assigned-pds", "assigned-pds" ]
      "rows": [
        [ 10, 4096, 5000, 2400, 3, 0, 0, 0],
        [ 20, 0, 0, 0, 0, 1048, 300, 233 ]
        [ 30, 256, 60, 60, 0, 1048, 15, 15 ]
      ],
    "timestamp": "2018-05-04 15:03:37.000000"
    }
  }
}
```

16.17 radius: RADIUS Server Support

The RADIUS hooks library allows Kea to interact with two types of RADIUS servers: access and accounting. Although the most common DHCP and RADIUS integration is done on the DHCP relay-agent level (DHCP clients send DHCP packets to DHCP relays; those relays contact the RADIUS server and depending on the response either send the packet to the DHCP server or drop it), it does require DHCP relay hardware to support RADIUS communication.

Also, even if the relay has the necessary support, it is often not flexible enough to send and receive additional RADIUS attributes. As such, the alternative looks more appealing: to extend the DHCP server to talk to RADIUS directly. That is the goal this library intends to fulfill.

Note: This library may only be loaded by the `kea-dhcp4` or the `kea-dhcp6` process.

The major feature of this hooks library is the ability to use RADIUS authorization. When a DHCP packet is received, the Kea server sends an Access-Request to the RADIUS server and waits for a response. The server then sends back either an Access-Accept with specific client attributes, or an Access-Reject. There are two cases supported here: first, the Access-Accept includes a Framed-IP-Address (for DHCPv4) or Framed-IPv6-Address (for DHCPv6), which will be interpreted by Kea as an instruction to assign that specified IPv4 or IPv6 address. This effectively means RADIUS can act as an address-reservation database.

The second case supported is the ability to assign clients to specific pools based on a RADIUS response. In this case, the RADIUS server sends back an Access-Accept with a Framed-Pool (IPv4) or Framed-IPv6-Pool (IPv6). In both cases, Kea interprets those attributes as client classes. With the addition of the ability to limit access to pools to specific classes (see *Configuring Pools With Class Information*), RADIUS can be used to force the client to be assigned a dynamic address from a specific pool. Furthermore, the same mechanism can be used to control what kind of options the client will get if there are DHCP options specified for a particular class.

16.17.1 Compilation and Installation of the RADIUS Hook

The following section describes how to compile and install the software on CentOS 7.0. Other systems may differ slightly.

Note: Starting with Kea 1.7.0, ISC now provides Kea software and hooks in convenient to use native DEB and RPM packages. This includes RADIUS hook and the required patched version of FreeRADIUS client library. The software compilation for RADIUS is complicated. unless you have specific reasons to compile it yourself, you should seriously consider using native packages.

STEP 1: Install dependencies

Several tools are needed to build the dependencies and Kea itself. The following commands should install them:

```
$ sudo rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
$ sudo yum install gcc-g++ openssl-devel log4cplus-devel wget git
```

STEP 2: Install FreeRADIUS

The Kea RADIUS hooks library uses the FreeRADIUS client library to conduct RADIUS communication. Unfortunately, the standard 1.1.7 release available from the project website https://freeradius.org/sub_projects/ has several serious deficiencies; ISC engineers observed a segmentation fault during testing. Also, the base version of the library does not offer asynchronous transmissions, which are essential for effective accounting implementation. Both of these issues were addressed by ISC engineers, and the changes have been reported to the FreeRADIUS client project. Acceptance of those changes is outside of ISC's control, so until those are processed, it is strongly recommended to use the FreeRADIUS client with ISC's patches. To download and compile this version, please use the following steps:

```
$ git clone https://github.com/fxdupont/freeradius-client.git
$ cd freeradius-client/
$ git checkout iscdev
$ ./configure
$ make
$ sudo make install
```

Additional parameters may be passed to the configure script, if needed. Once installed, the FreeRADIUS client will be installed in /usr/local. This is the default path where Kea will be looking for it. It can be installed in a different directory; if so, make sure to add that path to the configure script when compiling Kea.

STEP 3: Install recent BOOST version

Kea requires a reasonably recent Boost version. Unfortunately, the version available in CentOS 7 is too old, so a newer Boost version is necessary. Furthermore, CentOS 7 has an old version of the g++ compiler that does not handle the latest Boost versions. Fortunately, Boost 1.65 meets both requirements; it is both recent enough for Kea and able to be compiled using the g++ 4.8 version in CentOS.

To download and compile Boost 1.65, please use the following commands:

```
$ wget -nd https://dl.bintray.com/boostorg/release/1.65.1/source/boost_1_65_1.tar.gz
$ tar zxvf boost_1_65_1.tar.gz
$ cd boost_1_65_1/
$ ./bootstrap.sh
$ ./b2 --without-python
$ sudo ./b2 install
```

Note that the b2 script may optionally take extra parameters; one of them specifies the destination path where the sources are to be compiled.

STEP 4: Compile and install Kea

Obtain the Kea sources either by downloading them from the git repository or extracting the tarball. Use one of those commands to obtain the Kea sources.

Choice 1: get from github

```
$ git clone https://github.com/isc-projects/kea
```

Choice 2: get a tarball and extract it

```
$ tar zxvf kea-1.9.0.tar.gz
```

The next step is to extract the premium Kea package that contains the RADIUS repository into the Kea sources. After the tarball is extracted, the Kea sources should have a premium/ subdirectory.

```
$ cd kea
$ tar zxvf ../kea-premium-radius-1.9.0.tar.gz
```

Once this is done, verify that the Kea sources look similar to this:

```
$ ls -l
total 952
-rw-r--r--   1 thomson  staff    6192 Apr 25 17:38 AUTHORS
-rw-r--r--   1 thomson  staff   29227 Apr 25 17:38 COPYING
-rw-r--r--   1 thomson  staff  360298 Apr 25 20:00 ChangeLog
-rw-r--r--   1 thomson  staff     645 Apr 25 17:38 INSTALL
-rw-r--r--   1 thomson  staff    5015 Apr 25 17:38 Makefile.am
-rw-r--r--   1 thomson  staff     587 Apr 25 17:38 README
-rw-r--r--   1 thomson  staff   62323 Apr 25 17:38 configure.ac
drwxr-xr-x  12 thomson  staff     408 Apr 26 19:04 doc
drwxr-xr-x   7 thomson  staff     238 Apr 25 17:38 examples
drwxr-xr-x   5 thomson  staff     170 Apr 26 19:04 ext
drwxr-xr-x   8 thomson  staff     272 Apr 26 19:04 m4macros
drwxr-xr-x  20 thomson  staff     680 Apr 26 11:22 premium
drwxr-xr-x  10 thomson  staff     340 Apr 26 19:04 src
drwxr-xr-x  14 thomson  staff     476 Apr 26 19:04 tools
```

The makefiles must be regenerated using `autoreconf`.

The next step is to configure Kea, and there are several essential steps necessary here. Running `autoreconf -if` is necessary to compile the premium package that contains RADIUS. Also, the –with-freeradius option is necessary to tell Kea where the FreeRADIUS client sources can be found. Also, since the non-standard Boost is used, the path to it must be specified.

```
$ autoreconf -i
$ ./configure --with-freeradius=/path/to/freeradius --with-boost-include=/path/to/
↪boost --with-boost-lib-dir=/path/to/boost/state/lib
```

For example, assuming the FreeRADIUS client was installed in the default directory (/usr/local) and the Boost 1.65 sources were compiled in /home/thomson/devel/boost1_65_1, the configure path should look as follows:

```
$ ./configure --with-freeradius=/usr/local \
      --with-boost-include=/home/thomson/devel/boost_1_65_1 \
      --with-boost-lib-dir=/home/thomson/devel/boost_1_65_1/stage/lib
```

After some checks, the configure script should print a report similar to the following:

```
       Kea source configure results:
    -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

Package:
  Name:              kea
  Version:           1.9.0
  Extended version:  1.9.0 (tarball)
  OS Family:         Linux

  Hooks directory:   /usr/local/lib/kea/hooks
  Premium hooks:     yes
  Included Hooks:    forensic_log flex_id host_cmds subnet_cmds radius host_cache

C++ Compiler:
  CXX:             g++ --std=c++11
  CXX_VERSION:     g++ (GCC) 4.8.5 20150623 (Red Hat 4.8.5-16)
  CXX_STANDARD:    201103
  DEFS:            -DHAVE_CONFIG_H
  CPPFLAGS:         -DOS_LINUX  -DBOOST_ASIO_HEADER_ONLY
  CXXFLAGS:        -g -O2
  LDFLAGS:          -lpthread
  KEA_CXXFLAGS:     -Wall -Wextra -Wnon-virtual-dtor -Wwrite-strings -Woverloaded-virtual -

Python:
  PYTHON_VERSION:  not needed (because kea-shell is disabled)

Boost:
  BOOST_VERSION:   1.65.1
  BOOST_INCLUDES:  -I/home/thomson/devel/boost_1_65_1
  BOOST_LIBS:      -L/home/thomson/devel/boost_1_65_1/stage/lib  -lboost_system

OpenSSL:
  CRYPTO_VERSION:  OpenSSL 1.0.2k  26 Jan 2017
  CRYPTO_CFLAGS:
  CRYPTO_INCLUDES:
  CRYPTO_LDFLAGS:
```

```
  CRYPTO_LIBS:     -lcrypto

Botan: no

Log4cplus:
  LOG4CPLUS_VERSION:  1.1.3
  LOG4CPLUS_INCLUDES: -I/usr/include
  LOG4CPLUS_LIBS:     -L/usr/lib -L/usr/lib64 -llog4cplus

Flex/bison:
  FLEX:  flex
  BISON: bison -y

MySQL:
  no

PostgreSQL:
  no

Cassandra CQL:
  no
Google Test:
  no
Google Benchmark:
  no

FreeRADIUS client:
  FREERADIUS_INCLUDE:    -I/usr/local/include
  FREERADIUS_LIB:        -L/usr/local/lib -lfreeradius-client
  FREERADIUS_DICTIONARY: /usr/local/etc/radiusclient/dictionary

Developer:
  Enable Debugging:       no
  Google Tests:           no
  Valgrind:               not found
  C++ Code Coverage:      no
  Logger checks:          no
  Generate Documentation: no
  Parser Generation:      no
  Kea-shell:              no
  Perfdhcp:               no
```

Please make sure that the compilation includes the following:

- RADIUS listed in Included Hooks;
- FreeRADIUS client directories printed and pointing to the right directories;
- Boost version at least 1.65.1. The versions available in CentOS 7 (1.48 and and 1.53) are too old.

Once the configuration is complete, compile Kea using make. If the system has more than one core, using the "-j N" option is recommended to speed up the build.

```
$ make -j5
$ sudo make install
```

16.17.2 RADIUS Hook Configuration

The RADIUS hook is a library that has to be loaded by either DHCPv4 or DHCPv6 Kea servers. Unlike some other available hooks libraries, this one takes many parameters. For example, this configuration could be used:

```
"Dhcp4": {

# Your regular DHCPv4 configuration parameters here.

"hooks-libraries": [
{
    # Note that RADIUS requires host-cache for proper operation,
    # so that library is loaded as well.
    "library": "/usr/local/lib/kea/hooks/libdhcp_host_cache.so"
},
{
    "library": "/usr/local/lib/kea/hooks/libdhc_radius.so",
    "parameters": {

        # Specify where FreeRADIUS dictionary could be located
        "dictionary": "/usr/local/etc/freeradius/dictionary",

        # Specify which address to use to communicate with RADIUS servers
        "bindaddr": "*",

        # more RADIUS parameters here
    }
} ]
```

RADIUS is a complicated environment. As such, it is not feasible to provide a default configuration that works for everyone. However, we do have one example that showcases some of the more common features. Please see doc/examples/kea4/hooks-radius.json in the Kea sources.

The RADIUS hook library supports the following global configuration flags, which correspond to FreeRADIUS client library options:

- `bindaddr` (default "*") - specifies the address to be used by the hooks library in communication with RADIUS servers. The "*" special value tells the kernel to choose the address.
- `canonical-mac-address` (default false) - specifies whether MAC addresses in attributes follow the canonical RADIUS format (lowercase pairs of hexadecimal digits separated by '-').
- `client-id-pop0` (default false) - used with flex-id, removes the leading zero (or pair of zeroes in DHCPv6) type in client-id (aka duid in DHCPv6). Implied by client-id-printable.
- `client-id-printable` (default false) - checks whether the client-id/duid content is printable and uses it as is instead of in hexadecimal. Implies client-id-pop0 and extract-duid as 0 and 255 are not printable.
- `deadtime` (default 0) is a mechanism to try unresponsive servers after responsive servers. Its value specifies the number of seconds after which a server is considered not to have answered, so 0 disables the mechanism. As the asynchronous communication does not use locks or atomics, it is recommended that you do not use this feature when running in this mode.
- `dictionary` (default set by configure at build time) - is the attribute and value dictionary. Note that it is a critical parameter.
- `extract-duid` (default true) - extracts the embedded duid from an RFC 4361-compliant DHCPv4 client-id. Implied by client-id-printable.
- `identifier-type4` (default client-id) - specifies the identifier type to build the User-Name attribute. It

should be the same as the host identifier, and when the flex-id hook library is used the replace-client-id must be set to true; client-id will be used with client-id-pop0.

- `identifier-type6` (default duid) - specifies the identifier type to build the User-Name attribute. It should be the same as the host identifier, and when the flex-id hook library is used the replace-client-id must be set to true; duid will be used with client-id-pop0.
- `realm` (default "") - is the default realm.
- `reselect-subnet-address` (default false) - uses the Kea reserved address/RADIUS Framed-IP-Address or Framed-IPv6-Address to reselect subnets where the address is not in the subnet range.
- `reselect-subnet-pool` (default false) - uses the Kea client-class/RADIUS Frame-Pool to reselect subnets where no available pool can be found.
- `retries` (default 3) - is the number of retries before trying the next server. Note that it is not supported for asynchronous communication.
- `session-history` (default "") - is the name of the file providing persistent storage for accounting session history.
- `timeout` (default 10) - is the number of seconds during which a response is awaited.

When `reselect-subnet-pool` or `reselect-subnet-address` is set to true at the reception of RADIUS Access-Accept, the selected subnet is checked against the client-class name or the reserved address; if it does not match, another subnet is selected among matching subnets.

Two services are supported:

- `access` - the authentication service.
- `accounting` - the accounting service.

Configuration of services is divided into two parts:

- Servers that define RADIUS servers that the library is expected to contact. Each server may have the following items specified:
  - `name` - specifies the IP address of the server (it is possible to use a name which will be resolved, but it is not recommended).
  - `port` (default RADIUS authentication or accounting service) - specifies the UDP port of the server. Note that the FreeRADIUS client library by default uses ports 1812 (authorization) and 1813 (accounting). Some server implementations use 1645 (authorization) and 1646 (accounting). The "port" parameter may be used to adjust as needed.
  - `secret` - authenticates messages.

  There may be up to eight servers. Note that when no server is specified, the service is disabled.
- Attributes which define additional information that the Kea server will send to a RADIUS server. The parameter must be identified either by a name or type. Its value can be specified in one of three possible ways: data (which defines a plain text value), raw (which defines the value in hex), or expr (which defines an expression, which will be evaluated for each incoming packet independently).
  - `name` - the name of the attribute.
  - `type` - the type of the attribute. Either the type or the name must be provided, and the attribute must be defined in the dictionary.
  - `data` - the first of three ways to specify the attribute content. The data entry is parsed by the FreeRADIUS library, so values defined in the dictionary of the attribute may be used.

  - `raw` - the second of three ways to specify the attribute content; it specifies the content in hexadecimal. Note that it does not work with integer-content attributes (date, integer, and IPv4 address); a string-content attribute (string, IPv6 address, and IPv6 prefix) is required.
  - `expr` - the last way to specify the attribute content. It specifies an evaluation expression which must return a not-empty string when evaluated with the DHCP query packet. Currently this is restricted to the access service.

For example, to specify a single access server available on localhost that uses "xyz123" as a secret, and tell Kea to send three additional attributes (Password, Connect-Info, and Configuration-Token), the following snippet could be used:

```
"parameters": {

    # Other RADIUS parameters here

    "access": {

        # This starts the list of access servers
        "servers": [
        {
            # These are parameters for the first (and only) access server
            "name": "127.0.0.1",
            "port": 1812,
            "secret": "xyz123"
        }
        # Additional access servers could be specified here
        ],

        # This defines a list of additional attributes Kea will send to each
        # access server in Access-Request.
        "attributes": [
        {
            # This attribute is identified by name (must be present in the
            # dictionary) and has static value (i.e. the same value will be
            # sent to every server for every packet)
            "name": "Password",
            "data": "mysecretpassword"
        },
        {
            # It is also possible to specify an attribute using its type,
            # rather than a name. 77 is Connect-Info. The value is specified
            # using hex. Again, this is a static value. It will be sent the
            # same for every packet and to every server.
            "type": 77,
            "raw": "65666a6a71"
        },
        {
            # This example shows how an expression can be used to send dynamic
            # value. The expression (see Section 13) may take any value from
            # the incoming packet or even its metadata (e.g. the interface
            # it was received over from)
            "name": "Configuration-Token",
            "expr": "hexstring(pkt4.mac,':')"
        }
        ] # End of attributes
    } # End of access

    # Accounting parameters.
    "accounting": {
        # This starts the list of accounting servers
        "servers": [
        {
            # These are parameters for the first (and only) accounting server
            "name": "127.0.0.1",
            "port": 1813,
            "secret": "sekret"
        }
        # Additional accounting servers could be specified here
        ]
    }

}
```

For the RADIUS hooks library to operate properly in DHCPv4, the Host Cache hooks library must also be loaded. The reason for this is somewhat complex. In a typical deployment, the DHCP clients send their packets via DHCP relay which inserts certain Relay Agent Information options, such as circuit-id or remote-id. The values of those options are then used by the Kea DHCP server to formulate the necessary attributes in the Access-Request message sent to the RADIUS server. However, once the DHCP client gets its address, it then renews by sending packets directly to the DHCP server. As a result, the relays are not able to insert their RAI options, and the DHCP server cannot send the Access-Request queries to the RADIUS server by using just the information from incoming packets. Kea needs to keep the information received during the initial Discover/Offer exchanges and use it again later when sending accounting messages.

This mechanism is implemented based on user context in host reservations. (See *Comments and User Context* and *User Contexts in Hooks* for details.) The host cache mechanism allows the information retrieved by RADIUS to be stored and later used for sending accounting and access queries to the RADIUS server. In other words, the host-cache mechanism is mandatory, unless administrators do not want RADIUS communication for messages other than Discover and the first Request from each client.

16.18 host_cache: Caching Host Reservations

Some database backends, such as RADIUS, are considered slow and may take a long time to respond. Since Kea in general is synchronous, backend performance directly affects DHCP performance. To minimize the impact and improve performance, the Host Cache library provides a way to cache information from the database locally. This includes negative caching, i.e. the ability to remember that there is no client information in the database.

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

In principle, this hooks library can be used with any backend that may introduce performance degradation (MySQL, PostgreSQL, Cassandra, or RADIUS). Host Cache must be loaded for the RADIUS accounting mechanism to work.

The Host Cache hooks library is currently very simple. It takes only one optional parameter ("maximum"), which defines the maximum number of hosts to be cached. If not specified, the default value of 0 is used, which means there is no limit. This hooks library can be loaded the same way as any other hooks library; for example, this configuration could be used:

```
"Dhcp4": {

# Your regular DHCPv4 configuration parameters here.

"hooks-libraries": [
{
    "library": "/usr/local/lib/kea/hooks/libdhc_host_cache.so",
    "parameters": {

        # Tells Kea to never cache more than 1000 hosts.
        "maximum": 1000

    }
} ]
```

Once loaded, the Host Cache hooks library provides a number of new commands which can be used either over the control channel (see *Using the Control Channel*) or the RESTful API (see *Overview of the Kea Control Agent*). An example RESTful API client is described in *Overview of the Kea Shell*. The following sections describe the commands available.

16.18.1 The cache-flush Command

This command allows removal of a specified number of cached host entries. It takes one parameter, which defines the number of hosts to be removed. An example usage looks as follows:

```
{
    "command": "cache-flush",
    "arguments": 1000
}
```

This command will remove 1000 hosts. To delete all cached hosts, please use cache-clear instead. The hosts are stored in FIFO (first-in, first-out) order, so the oldest entries are always removed.

16.18.2 The cache-clear Command

This command allows removal of all cached host entries. An example usage looks as follows:

```
{
    "command": "cache-clear"
}
```

This command will remove all hosts. To delete only a certain number of cached hosts, please use cache-flush instead.

16.18.3 The cache-size Command

This command returns the number of host entries. An example usage looks as follows:

```
{
    "command": "cache-size"
}
```

16.18.4 The cache-write Command

In general, the cache content is considered a runtime state and the server can be shut down or restarted as usual; the cache will then be repopulated after restart. However, there are some cases when it is useful to store the contents of the cache. One such case is RADIUS, where the cached hosts also retain additional cached RADIUS attributes; there is no easy way to obtain this information again, because renewing clients send their packet to the DHCP server directly.

Another use case is when an administrator wants to restart the server and, for performance reasons, wants it to start with a hot (populated) cache.

This command allows writing the contents of the in-memory cache to a file on disk. It takes one parameter, which defines the filename. An example usage looks as follows:

```
{
    "command": "cache-write",
    "arguments": "/tmp/kea-host-cache.json"
}
```

This causes the contents to be stored in the /tmp/kea-host-cache.json file. That file can then be loaded with the cache-load command or processed by any other tool that is able to understand JSON format.

16.18.5 The cache-load Command

See the previous section for a discussion of use cases where it may be useful to write and load contents of the host cache to disk.

This command allows the contents of a file on disk to be loaded into an in-memory cache. It takes one parameter, which defines the filename. An example usage looks as follows:

```
{
    "command": "cache-load",
    "arguments": "/tmp/kea-host-cache.json"
}
```

This command will store the contents to the /tmp/kea-host-cache.json file. That file can then be loaded with the cache-load command or processed by any other tool that is able to understand JSON format.

16.18.6 The cache-get Command

This command is similar to cache-write, but instead of writing the cache contents to disk, it returns the contents to whoever sent the command.

This command allows the contents of a file on disk to be loaded into an in-memory cache. It takes one parameter, which defines the filename. An example usage looks as follows:

```
{
    "command": "cache-get"
}
```

This command will return all the cached hosts. Note that the response may be large.

16.18.7 The cache-get-by-id Command

This command is similar to cache-get, but instead of returning the whole content it returns only the entries matching the given identifier.

It takes one parameter, which defines the identifier of wanted cached host reservations. An example usage looks as follows:

```
{
    "command": "cache-get-by-id",
    "arguments": {
        "hw-address": "01:02:03:04:05:06"
    }
}
```

This command will return all the cached hosts with the given hardware address.

16.18.8 The cache-insert Command

This command may be used to manually insert a host into the cache; there are very few use cases when this command might be useful. This command expects its arguments to follow the usual syntax for specifying host reservations (see *Host Reservation in DHCPv4* or *Host Reservation in DHCPv6*), with one difference: the subnet-id value must be specified explicitly.

An example command that will insert an IPv4 host into the host cache looks as follows:

```
{
    "command": "cache-insert",
    "arguments": {
        "hw-address": "01:02:03:04:05:06",
        "subnet-id4": 4,
        "subnet-id6": 0,
        "ip-address": "192.0.2.100",
        "hostname": "somehost.example.org",
        "client-classes4": [ ],
        "client-classes6": [ ],
        "option-data4": [ ],
        "option-data6": [ ],
        "next-server": "192.0.0.2",
        "server-hostname": "server-hostname.example.org",
        "boot-file-name": "bootfile.efi",
        "host-id": 0
    }
}
```

An example command that will insert an IPv6 host into the host cache looks as follows:

```
{
    "command": "cache-insert",
    "arguments": {
        "hw-address": "01:02:03:04:05:06",
        "subnet-id4": 0,
        "subnet-id6": 6,
        "ip-addresses": [ "2001:db8::cafe:babe" ],
        "prefixes": [ "2001:db8:dead:beef::/64" ],
        "hostname": "",
        "client-classes4": [ ],
        "client-classes6": [ ],
        "option-data4": [ ],
        "option-data6": [ ],
        "next-server": "0.0.0.0",
        "server-hostname": "",
        "boot-file-name": "",
        "host-id": 0
    }
}
```

16.18.9 The cache-remove Command

Sometimes it is useful to remove a single entry from the host cache. A good use case is a situation where the device is up, Kea has already provided configuration, and the host entry is in cache. As a result of administrative action (e.g. the customer hasn't paid their bills or has perhaps been upgraded to better service), the information in the backend (e.g. MySQL or RADIUS) is being updated. However, since the cache is in use, Kea does not notice the change as the cached values are used. The cache-remove command can solve this problem by removing a cached entry after administrative changes.

The cache-remove command works similarly to the reservation-get command. It allows querying by two parameters: either subnet-id4 or subnet-id6; or ip-address (may be an IPv4 or IPv6 address), hw-address (specifies hardware/MAC address), duid, circuit-id, client-id, or flex-id.

An example command to remove an IPv4 host with reserved address 192.0.2.1 from a subnet with a subnet-id 123 looks as follows:

```
{
    "command": "cache-remove",
    "arguments": {
        "ip-address": "192.0.2.1",
        "subnet-id": 123
    }
}
```

Another example that removes an IPv6 host identifier by DUID and specific subnet-id is:

```
{
    "command": "cache-remove",
    "arguments": {
        "duid": "00:01:ab:cd:f0:a1:c2:d3:e4",
        "subnet-id": 123
    }
}
```

16.19 lease_query: Leasequery

This library provides support for DHCPv4 Leasequery as described in RFC 4388; and for DHCPv6 Lease Query (RFC 5007).

Note: This library may only be loaded by the `kea-dhcp4` or `kea-dhcp6` process.

The Leasequery library is only available to ISC customers with a paid support contract.

16.19.1 DHCPv4 Leasequery

DHCPv4 simple Leasequery provides a requester the ability to query for active lease information for either a single IP address or a single client. RFC 4388 calls for three such queries:

- Query by IP address

  The IP address of interest is contained within the `ciaddr` field of the query.

- Query by hardware address

The hardware address of interest is contained with the `chaddr` field of the query.

- Query by client identifier

  The client identifier of interest is sent in the dhcp-client-identifier option (61) of the query.

The inbound DHCPLEASEQUERY packet must supply only one of the three values above. Queries which supply more than one of these values are dropped.

In addition, the query must contain the IP address of the requester in `giaddr`. This value will be used not only as the destination for the query response but also to validate the requester against a known list of IP addresses which are permitted to query. This list of valid requester addresses is specified as part of the Leasequery hook library's configuration (See the section on configuration below).

In response to a valid query, the server will return one of three message types:

- DHCPLEASEUNKNOWN

  Returned when the IP address of interest is not one the server knows about (query by IP address); or there are no active leases for the client of interest (query by hardware address or client id).

- DHCPLEASEUNASSIGNED

  Returned when the IP address is one the server knows of but for which there are no active leases (applies only to query by IP address).

- DHCPLEASEACTIVE

  Returned when there is at least one active lease found matching the criteria.

For both DHCPLEASEUNKNOWN and DHCPLEASEUNASSIGNED responses, the only information sent back to the requester in response is the query parameter itself (i.e. one of: IP address, hardware address, or client identifier).

For DHCPLEASEACTIVE the server will provide the following information for the newest active lease that matches the criteria, in the response:

- ciaddr - set to the lease's IP address
- chaddr - set to the lease's hardware address

In addition, one or more of the following options will be included:

Table 16.3: DHCPLEASEACTIVE Options

Option	Code	Content
dhcp-client-identifier	61	copied from the lease (if one)
client-last-transaction-time	91	amount of time that has elapsed since the lease's client-last-transaction-time (CLTT) This value will also be used by the server to adjust life time and timer values.
dhcp-lease-time	51	lease's life time reduced by CLTT
dhcp-renewal-time	58	as controlled by kea-dhcp4 configuration and then reduced by CLTT
dhcp-rebind-time	59	as dictacted by kea-dhcp4 configuration and then reduced by CLTT
dhcp-agent-options	82	if stored on the lease. (See *Storing Extended Lease Information*)
associated-ip	92	a list of all other IP addresses for which the client has active leases. (Does not apply to query by IP address)

The dhcp-server-identifier option (54) will be returned in all responses in keeping with RFC 2131 section 4.3.1.

RFC 4388 allows requesters to ask for specific options via the dhcp-parameter-request-list (PRL, option 55). This is not currently supported but may be added at a future date.

16.19.2 DHCPv4 Leasequery Configuration

Configuring the Leasequery hook library for use is straight forward. It currently supports a single parameter, `requesters`, which is a list of IP addresses from which DHCPLEASEQUERY packets will be accepted. In other words, it is a list of known requesters. The following shows an example configuration with two requester addresses:

```
:
   "hooks-libraries": [
       {
           "library": "lib/kea/hooks/libdhcp_lease_query.so",
           "parameters": {
               "requesters": [ "192.0.1.1", "10.0.0.2" ]
           }
       }
   ],
:
```

Note: For security purposes, there is no way to specify wildcards. Each requester address must be explicitly listed.

16.19.3 DHCPv6 Leasequery

DHCPv6 simple Leasequery provides a requester the ability to query for active lease information for either a single IP address or a single client DUID. The query type and parameters are conveyed in an `lq-query` option (44) attached to a DHCPV6_LEASEQUERY message. Briefly,

- query-type

  This is either `query-by-address` (1) or `query-by-clientid` (2)

- link-address

  Global link address, when not empty, instructs the query to be limited to leases within that "link". Kea uses this value to select only leases that belong to subnets whose prefix matches this value. Note that active leases for prefix delegations for a matched subnet will be included in the query reply, even if the delegated prefix itself falls outside the subnet prefix.

- query-options

  A single `iaaddr` option (12) must be supplied when querying by address. When querying by client ID, a single `clientid` option (1) must be supplied. RFC 5007 also calls for an optional, `oro` option (6), to request specific options be returned for matched leases. This is not currently implemented.

Note: RFC 5007, Section 3.3 states that querying by IP address should return either a lease (e.g. binding) for the address itself or a lease for a delegated prefix that contains the address. The latter is not currently implemented. Leases for delegated prefixes may only be returned when querying by client ID. See gitlab issue #1275

DHCPV6_LEASEQUERY queries will only be honored if the source address of the query matches an entry in a list of known IP addresses which are permitted to query. This list of valid requester addresses is specified as part of the Leasequery hook library's configuration (See the section on configuration below). Queries received from unknown requesters will be logged and dropped.

In response to a valid query, the server will carry out the requisite activities and return a DHCPV6_LEASEQUERY_REPLY. All replies will contain at least a `status-code` option (13) that indicates the outcome of the query as detailed in the following table:

Table 16.4: DHCPV6_LEASEQUERY_REPLY Status Option Values per Query Outcome

Query Outcome	Status Label	Status Code	Status Text
Invalid query type field	STATUS_UnknownQueryType	7	“unknown query-ty
Query by IP address that does not contain an address option	STATUS_Malformed	10	“missing D6O_IAA
Query by IP address for an address that does fall within any configured pools	STATUS_NotConfigured	9	“address not in a co
Query by IP address which found only an inactive lease (e.g. expired, declined, reclaimed-expired)	STATUS_Success	0	“inactive lease exis
Query by IP address that found no leases (active or otherwise)	STATUS_Success	0	“no active lease”
Query by IP address that found an active lease for the address	STATUS_Success	0	“active lease found
Query by Client ID that does not contain a client ID option	STATUS_Malformed	10	“missing D6O_CL
Query by Client ID with a link address that does not match any configured subnets	STATUS_NotConfigured	9	“not a configured li
Query by client id which found no matching leases	STATUS_Success	0	“no active leases”
Query by client id which found one or more active leases	STATUS_Success	0	“active lease(s) fou

For those scenarios where the query was either invalid or no matching, active leases were found the DHCPV6_LEASEQUERY_REPLY will only contain the status-code option (12) per the above table.

When a query finds active leases in more than one subnet and query’s link-address is empty, then in addition to the status-code, the DHCPV6_LEASEQUERY_REPLY will contain an `lq-client-link` option (48). The lq-client-link will contain a list of IPv6 addresses, one for each subnet in which a lease was found (see RFC 5007, Section 4.1.2.5) If, however, the query’s link-address is not empty, the list of queries will be pruned to contain only leases that belong to that subnet.

When the query results in one or more active leases which all belong to a single subnet, in addition to the status-code, the DHCPV6_LEASEQUERY_REPLY will contain a client-data option (45) (see RFC 5007, Section 4.1.2.2) The client-data option will encapsulate the following options:

Table 16.5: OPTION_CLIENT_DATA returned when active lease(s) are found

Option	Code	Content
clientid	1	copied from the lease (if one)
clt-time	46	amount of time that has elapsed since the lease’s client-last-transaction-time (CLTT) This value will also be used by the server to adjust life time and timer values.
iaaddr	5	One option per matched address, fields in each option: - lease address - valid life time reduced by CLTT - preferred life time reduced by CLTT
iaprefix	26	One option per matched prefix, fields in each option: - prefix - prefix length - valid life time reduced by CLTT - preferred life time reduced by CLTT

If the lease with the most recent CLTT value (Client Last Transmission Time) has relay information in it’s user-context (see *Storing Extended Lease Information*), then an OPTION_LQ_RELAY_DATA option will be added to the reply (see RFC 5007, Section 4.1.2.4).

The relay information on the lease is a list with an entry for each relay layer the client packet (e.g. DHCPV6_REQUEST) traversed, with the first entry in list being the outermost layer (closest to the server). The `peer-address` field of the lq-rely-option is set to the peer address of this relay. The list of relays is then used to construct a DHCPV6_RELAY_FORW message equivalent to that which contained the client packet, minus the client packet. This message is stored in the `DHCP-relay-message` field of the lq-relay-data option.

16.19.4 DHCPv6 Leasequery Configuration

Configuring the Leasequery hook library for use is straight forward. It currently supports a single parameter, `requesters`, which is a list of IP addresses from which DHCPV6_LEASEQUERY packets will be accepted. In other words, it is a list of known requesters. The following shows an example configuration with two requester addresses:

```
:
   "hooks-libraries": [
       {
           "library": "lib/kea/hooks/libdhcp_lease_query.so",
           "parameters": {
               "requesters": [ "2001:db8:1::1", "2001:db8:2::1" ]
           }
       }
   ],
:
```

Note: For security purposes, there is no way to specify wildcards. Each requester address must be explicitly listed.

16.20 User Contexts in Hooks

Hooks libraries can have their own configuration parameters, which is convenient if the parameter applies to the whole library. However, sometimes it is very useful to extend certain configuration entities with additional configuration data. This is where the concept of user contexts comes in. A system administrator can define an arbitrary set of data and attach it to Kea structures, as long as the data are specified as a JSON map. In particular, it is possible to define fields that are integers, strings, boolean, lists, or maps. It is possible to define nested structures of arbitrary complexity. Kea does not use that data on its own; it simply stores it and makes it available for the hooks libraries.

Another use case for user contexts may be storing comments and other information that will be retained by Kea. Regular comments are discarded when the configuration is loaded, but user contexts are retained. This is useful if administrators want their comments to survive config-set or config-get operations, for example.

If user context is supported in a given context, the parser translates “comment” entries into user context with a “comment” entry. The pretty print of a configuration did the opposite operation and put “comment” entries at the beginning of maps, but this was withdrawn in 1.7.9.

As of Kea 1.3, the structures that allow user contexts are pools of all types (addresses and prefixes) and subnets. Kea 1.4 extended user context support to the global scope, interfaces config, shared networks, subnets, client classes, option datas and definitions, host reservations, control socket, dhcp ddns, loggers and server id. These are supported in both DHCPv4 and DHCPv6, with the exception of server id which is DHCPv6 only.

CHAPTER
SEVENTEEN

STATISTICS

17.1 Statistics Overview

Both Kea DHCP servers support statistics gathering. A working DHCP server encounters various events that can cause certain statistics to be collected. For example, a DHCPv4 server may receive a packet (the pkt4-received statistic increases by one) that after parsing is identified as a DHCPDISCOVER (pkt4-discover-received). The server processes it and decides to send a DHCPOFFER representing its answer (the pkt4-offer-sent and pkt4-sent statistics increase by one). Such events happen frequently, so it is not uncommon for the statistics to have values in the high thousands. They can serve as an easy and powerful tool for observing a server's and a network's health. For example, if the pkt4-received statistic stops growing, it means that the clients' packets are not reaching the server.

There are four types of statistics:

- *integer* - this is the most common type. It is implemented as a 64-bit integer (int64_t in C++), so it can hold any value between -2^63 to 2^63-1.
- *floating point* - this type is intended to store floating-point precision. It is implemented as a C++ double type.
- *duration* - this type is intended for recording time periods. It uses the 'boost::posix_time::time_duration type, which stores hours, minutes, seconds, and microseconds.
- *string* - this type is intended for recording statistics in textual form. It uses the C++ std::string type.

During normal operation, the DHCPv4 and DHCPv6 servers gather statistics. For a list of DHCPv4 and DHCPv6 statistics, see *Statistics in the DHCPv4 Server* and *Statistics in the DHCPv6 Server*, respectively.

To extract data from the statistics module, the control channel can be used. See *Management API* for details. It is possible to retrieve a single statistic or all statistics, reset statistics (i.e. set to a neutral value, typically zero), or even completely remove a single statistic or all statistics. See the section *Commands for Manipulating Statistics* for a list of statistics-oriented commands.

Statistics can be used by external tools to monitor Kea. One example of such a tool is Stork. See *Monitoring Kea with Stork* for details how to use it to retrieve statistics periodically (and use other data sources) to get better insight into Kea health and operational status.

17.2 Statistics Lifecycle

In Kea 1.6.0 version and earlier, when the Kea server is started some of the statistics are initially not initialized. For example, the `pkt4-received` statistic is not available until the first DHCP packet is received. In the later Kea versions, this behavior has been changed and all of the statistics supported by the servers are initialized upon the servers' startup and should be returned in response to the commands such as `statistic-get-all`. The runtime statistics concerning DHCP packets processed is initially set to 0 and is reset upon the server restart.

Per-subnet statistics are recalculated when reconfiguration takes place.

In general, once a statistic is initialized it is held in the manager until explicitly removed, by `statistic-remove` or `statistic-remove-all` being called, or when the server is shut down.

Removing a statistic that is updated frequently makes little sense, as it will be re-added when the server code next records that statistic. The `statistic-remove` and `statistic-remove-all` commands are intended to remove statistics that are not expected to be observed in the near future. For example, a misconfigured device in a network may cause clients to report duplicate addresses, so the server will report increasing values of pkt4-decline-received. Once the problem is found and the device is removed, the system administrator may want to remove the pkt4-decline-received statistic, so it will not be reported anymore. If a duplicate address is ever detected again, the server will add this statistic back.

17.3 Commands for Manipulating Statistics

There are several commands defined that can be used for accessing (-get), resetting to zero or a neutral value (-reset), or removing a statistic completely (-remove). We can change the statistics time based limit (-sample-age-set) and size based limit (-sample-count-set) which control how long or how many samples of the given statistic are retained.

The difference between reset and remove is somewhat subtle. The reset command sets the value of the statistic to zero or a neutral value, so after this operation, the statistic will have a value of 0 (integer), 0.0 (float), 0h0m0s0us (duration), or “” (string). When requested, a statistic with the values mentioned will be returned. `Remove` removes a statistic completely, so the statistic will no longer be reported. Please note that the server code may add it back if there is a reason to record it.

Note: The following sections describe commands that can be sent to the server; the examples are not fragments of a configuration file. For more information on sending commands to Kea, see *Management API*.

17.3.1 The statistic-get Command

The `statistic-get` command retrieves a single statistic. It takes a single-string parameter called `name`, which specifies the statistic name. An example command may look like this:

```
{
    "command": "statistic-get",
    "arguments": {
        "name": "pkt4-received"
    }
}
```

The server returns details of the requested statistic, with a result of 0 indicating success and the specified statistic as the value of the “arguments” parameter. If the requested statistic is not found, the response will contain an empty map, i.e. only { } as an argument, but the status code will still indicate success (0). An example response:

```
{
    "command": "statistic-get",
    "arguments": {
        "pkt4-received": [ [ 125, "2019-07-30 10:11:19.498739" ], [ 100, "2019-07-30␣
↪10:11:19.498662" ] ]
    },
    "result": 0
}
```

17.3.2 The statistic-reset Command

The `statistic-reset` command sets the specified statistic to its neutral value: 0 for integer, 0.0 for float, 0h0m0s0us for time duration, and “” for string type. It takes a single-string parameter called `name`, which specifies the statistic name. An example command may look like this:

```
{
    "command": "statistic-reset",
    "arguments": {
        "name": "pkt4-received"
    }
}
```

If the specific statistic is found and the reset is successful, the server responds with a status of 0, indicating success, and an empty parameters field. If an error is encountered (e.g. the requested statistic was not found), the server returns a status code of 1 (error) and the text field contains the error description.

17.3.3 The statistic-remove Command

The `statistic-remove` command attempts to delete a single statistic. It takes a single-string parameter called `name`, which specifies the statistic name. An example command may look like this:

```
{
    "command": "statistic-remove",
    "arguments": {
        "name": "pkt4-received"
    }
}
```

If the specific statistic is found and its removal is successful, the server responds with a status of 0, indicating success, and an empty parameters field. If an error is encountered (e.g. the requested statistic was not found), the server returns a status code of 1 (error) and the text field contains the error description.

17.3.4 The statistic-get-all Command

The `statistic-get-all` command retrieves all statistics recorded. An example command may look like this:

```
{
    "command": "statistic-get-all",
    "arguments": { }
}
```

The server responds with details of all recorded statistics, with a result set to 0 to indicate that it iterated over all statistics (even when the total number of statistics is zero). An example response returning all collected statistics:

```
{
    "command": "statistic-get-all",
    "arguments": {
        "cumulative-assigned-addresses": [ [ 0, "2019-07-30 10:04:28.386740" ] ],
        "declined-addresses": [ [ 0, "2019-07-30 10:04:28.386733" ] ],
        "reclaimed-declined-addresses": [ [ 0, "2019-07-30 10:04:28.386735" ] ],
        "reclaimed-leases": [ [ 0, "2019-07-30 10:04:28.386736" ] ],
        "subnet[1].assigned-addresses": [ [ 0, "2019-07-30 10:04:28.386740" ] ],
        "subnet[1].cumulative-assigned-addresses": [ [ 0, "2019-07-30 10:04:28.386740
↪" ] ],
        "subnet[1].declined-addresses": [ [ 0, "2019-07-30 10:04:28.386743" ] ],
        "subnet[1].reclaimed-declined-addresses": [ [ 0, "2019-07-30 10:04:28.386745"␣
↪] ],
        "subnet[1].reclaimed-leases": [ [ 0, "2019-07-30 10:04:28.386747" ] ],
        "subnet[1].total-addresses": [ [ 200, "2019-07-30 10:04:28.386719" ] ]
    },
    "result": 0
}
```

17.3.5 The statistic-reset-all Command

The `statistic-reset` command sets all statistics to their neutral values: 0 for integer, 0.0 for float, 0h0m0s0us for time duration, and “” for string type. An example command may look like this:

```
{
    "command": "statistic-reset-all",
    "arguments": { }
}
```

If the operation is successful, the server responds with a status of 0, indicating success, and an empty parameters field. If an error is encountered, the server returns a status code of 1 (error) and the text field contains the error description.

17.3.6 The statistic-remove-all Command

The `statistic-remove-all` command attempts to delete all statistics. An example command may look like this:

```
{
    "command": "statistic-remove-all",
    "arguments": { }
}
```

If the removal of all statistics is successful, the server responds with a status of 0, indicating success, and an empty parameters field. If an error is encountered, the server returns a status code of 1 (error) and the text field contains the error description.

17.3.7 The statistic-sample-age-set Command

The `statistic-sample-age-set` command sets time based limit for collecting samples for given statistic. It takes two parameters a string called `name`, which specifies the statistic name and integer value called `duration`, which specifies the time limit for given statistic in seconds. An example command may look like this:

```
{
    "command": "statistic-sample-age-set",
    "arguments": {
        "name": "pkt4-received",
        "duration": 1245
    }

}
```

The server will respond with message about successfully set limit for the given statistic, with a result set to 0 indicating success and an empty parameters field. If an error is encountered (e.g. the requested statistic was not found), the server returns a status code of 1 (error) and the text field contains the error description.

17.3.8 The statistic-sample-age-set-all Command

The `statistic-sample-age-set-all` command sets time based limits for collecting samples for all statistics. It takes single-integer parameter called `duration`, which specifies the time limit for given statistic in seconds. An example command may look like this:

```
{
    "command": "statistic-sample-age-set-all",
    "arguments": {
        "duration": 1245
    }

}
```

The server will respond with message about successfully set limit for all statistics, with a result set to 0 indicating success and an empty parameters field. If an error is encountered, the server returns a status code of 1 (error) and the text field contains the error description.

17.3.9 The statistic-sample-count-set Command

The `statistic-sample-count-set` command sets size based limit for collecting samples for given statistic. An example command may look like this:

```
{
    "command": "statistic-sample-count-set",
    "arguments": {
        "name": "pkt4-received",
        "max-samples": 100
    }

}
```

The server will respond with message about successfully set limit for the given statistic, with a result set to 0 indicating success and an empty parameters field. If an error is encountered (e.g. the requested statistic was not found), the server returns a status code of 1 (error) and the text field contains the error description.

17.3.10 The statistic-sample-count-set-all Command

The `statistic-sample-count-set-all` command sets size based limits for collecting samples for all statistics. An example command may look like this:

```
{
    "command": "statistic-sample-count-set-all",
    "arguments": {
        "max-samples": 100
    }

}
```

The server will respond with message about successfully set limit for all statistics, with a result set to 0 indicating success and an empty parameters field. If an error is encountered, the server returns a status code of 1 (error) and the text field contains the error description.

17.4 Time series

Previously, by default, each statistic held only a single data point. When Kea attempted to record a new value, the existing previous value was overwritten. That approach has the benefit of taking up little memory and it covers most cases reasonably well. However, there may be cases where you need to have many data points for some process. For example, some processes, such as received packet size, packet processing time or number of database queries needed to process a packet, are not cumulative and it would be useful to keep many data points, perhaps to do some form of statistical analysis afterwards.

Since Kea 1.6, by default, each statistic holds 20 data points. Setting such limit prevent unlimited memory consumption growth. There are two ways to define the limts: time based (e.g. keep samples from the last 5 minutes) and size based. It's possible to change the size based limit by using one of two commands: `statistic-sample-count-set`, to set size limit for single statistic and `statistic-sample-count-set-all` for setting size based limits for all statistics. To set time based limit for single statistic use `statistic-sample-age-set`, and `statistic-sample-age-set-all` to set time based limits for all statistics. For given statistic only one type of limit can be active. It means that storage is limited only by time based limit or size based, never by both of them.

CHAPTER

EIGHTEEN

MANAGEMENT API

A classic approach to daemon configuration assumes that the server's configuration is stored in configuration files and, when the configuration is changed, the daemon is restarted. This approach has the significant disadvantage of introducing periods of downtime when client traffic is not handled. Another risk is that if the new configuration is invalid for any reason, the server may refuse to start, which will further extend the downtime period until the issue is resolved.

To avoid such problems, the DHCPv4, DHCPv6, and D2 servers in Kea include support for a mechanism that allows online reconfiguration without requiring server shutdown. Both servers can be instructed to open control sockets, which is a communications channel. The server is able to receive commands on that channel, act on them, and report back status.

The DHCPv4, DHCPv6, and D2 servers receive commands over the UNIX domain sockets. For details on how to configure these sockets, see *Management API for the DHCPv4 Server* and *Management API for the DHCPv6 Server*. While it is possible to control the servers directly using UNIX domain sockets, that requires that the controlling client be running on the same machine as the server. SSH is usually used to connect remotely to the controlled machine.

Network administrators usually prefer using some form of a RESTful API to control the servers, rather than using UNIX domain sockets directly. Therefore, Kea includes a component called the Control Agent (or CA), which exposes a RESTful API to the controlling clients and can forward commands to the respective Kea services over the UNIX domain sockets. The CA configuration is described in *Configuration*.

The HTTP requests received by the CA contain the control commands encapsulated within HTTP requests. Simply speaking, the CA is responsible for stripping the HTTP layer from the received commands and forwarding the commands in a JSON format over the UNIX domain sockets to the respective services. Because the CA receives commands for all services, it requires additional "forwarding" information to be included in the client's messages. This forwarding information is carried within the `service` parameter of the received command. If the `service` parameter is not included, or if the parameter is a blank list, the CA will assume that the control command is targeted at the CA itself and will try to handle it on its own.

Control connections over both HTTP and UNIX domain sockets are guarded with timeouts. The default timeout value is set to 10 seconds and is not configurable.

This API can be used by external tools to manage and monitor Kea operation. An example of such a monitoring tool is ISC Stork. For details, see *Monitoring Kea with Stork*.

18.1 Data Syntax

Communication over the control channel is conducted using JSON structures. If configured, Kea will open a socket and listen for incoming connections. A process connecting to this socket is expected to send JSON commands structured as follows:

```
{
    "command": "foo",
    "service": [ "dhcp4" ]
    "arguments": {
        "param1": "value1",
        "param2": "value2",
        ...
    }
}
```

The same command sent over the RESTful interface to the CA will have the following structure:

```
POST / HTTP/1.1\r\n
Content-Type: application/json\r\n
Content-Length: 147\r\n\r\n
{
    "command": "foo",
    "service": [ "dhcp4" ]
    "arguments": {
        "param1": "value1",
        "param2": "value2",
        ...
    }
}
```

`command` is the name of the command to execute and is mandatory. `arguments` is a map of the parameters required to carry out the given command. The exact content and format of the map are command-specific.

`service` is a list of the servers at which the control command is targeted. In the example above, the control command is targeted at the DHCPv4 server. In most cases, the CA will simply forward this command to the DHCPv4 server for processing via a UNIX domain socket. Sometimes, the command including a service value may also be processed by the CA, if the CA is running a hooks library which handles such a command for the given server. As an example, the hooks library loaded by the CA may perform some operations on the database, such as adding host reservations, modifying leases, etc. An advantage of performing DHCPv4-specific administrative operations in the CA, rather than forwarding it to the DHCPv4 server, is the ability to perform these operations without disrupting the DHCPv4 service, since the DHCPv4 server doesn't have to stop processing DHCP messages to apply changes to the database. Nevertheless, these situations are rather rare and, in most cases, when the `service` parameter contains a name of the service the commands are simply forwarded by the CA. The forwarded command includes the `service` parameter but this parameter is ignored by the receiving server. This parameter is only meaningful to the CA.

If the command received by the CA does not include a `service` parameter or this list is empty, the CA simply processes this message on its own. For example, a `config-get` command which includes no service parameter returns the Control Agent's own configuration. The `config-get` command with a service value "dhcp4" is forwarded to the DHCPv4 server and returns the DHCPv4 server's configuration.

The following list shows the mapping of the values carried within the `service` parameter to the servers to which the commands are forwarded:

- `dhcp4` - the command is forwarded to the `kea-dhcp4` server.
- `dhcp6` - the command is forwarded to the `kea-dhcp6` server.
- `d2` - the command is forwarded to the `kea-d2` server.

The server processing the incoming command will send a response of the form:

```
{
    "result": 0|1|2|3,
    "text": "textual description",
    "arguments": {
        "argument1": "value1",
        "argument2": "value2",
        ...
    }
}
```

`result` indicates the outcome of the command. A value of 0 means success, while any non-zero value designates an error or a failure to complete the requested action. Currently 1 indicates a generic error, 2 means that a command is not supported, and 3 means that the requested operation was completed, but the requested object was not found. For example, a well-formed command that requests a subnet that exists in a server's configuration returns the result 0. If the server encounters an error condition, it returns 1. If the command asks for the IPv6 subnet, but was sent to a DHCPv4 server, it returns 2. If the query asks for a subnet-id and there is no subnet with such an id, the result is 3.

The `text` field typically appears when the result is non-zero and contains a description of the error encountered, but it often also appears for successful outcomes. The exact text is command-specific, but in general uses plain English to describe the outcome of the command. `arguments` is a map of additional data values returned by the server which are specific to the command issued. The map may be present, but that depends on the specific command.

Note: When sending commands via the Control Agent, it is possible to specify multiple services at which the command is targeted. CA forwards this command to each service individually. Thus, the CA response to the controlling client contains an array of individual responses.

18.2 Using the Control Channel

The easiest way to start interacting with the control API is to use common UNIX/Linux tools such as `socat` and `curl`.

In order to control the given Kea service via a UNIX domain socket, use `socat` in interactive mode as follows:

```
$ socat UNIX:/path/to/the/kea/socket -
```

or in batch mode, include the "ignoreeof" option as shown below to ensure `socat` waits long enough for the server to respond:

```
$ echo "{ some command...}" | socat UNIX:/path/to/the/kea/socket -,ignoreeof
```

where `/path/to/the/kea/socket` is the path specified in the `Dhcp4/control-socket/socket-name` parameter in the Kea configuration file. Text passed to `socat` is sent to Kea and the responses received from Kea are printed to standard output. This approach communicates with the specific server directly and bypasses the Control Agent.

It is also easy to open a UNIX socket programmatically. An example of a simple client written in C is available in the Kea Developer's Guide, in the Control Channel Overview chapter, in the Using Control Channel section.

To use Kea's RESTful API with `curl`, use the following:

```
$ curl -X POST -H "Content-Type: application/json" -d '{ "command": "config-get",
↪"service": [ "dhcp4" ] }' http://ca.example.org:8000/
```

This assumes that the Control Agent is running on host `ca.example.org` and is running the RESTful service on port 8000.

18.3 Commands Supported by Both the DHCPv4 and DHCPv6 Servers

18.3.1 The build-report Command

The `build-report` command returns on the control channel what the command line `-W` argument displays, i.e. the embedded content of the `config.report` file. This command does not take any parameters.

```
{
    "command": "build-report"
}
```

18.3.2 The config-get Command

The `config-get` command retrieves the current configuration used by the server. This command does not take any parameters. The configuration returned is roughly equal to the configuration that was loaded using the -c command line option during server start-up or later set using the `config-set` command. However, there may be certain differences, as comments are not retained. If the original configuration used file inclusion, the returned configuration will include all parameters from all the included files.

Note that the returned configuration is not redacted, i.e. it will contain database passwords in plain text if those were specified in the original configuration. Care should be taken not to expose the command channel to unprivileged users.

An example command invocation looks like this:

```
{
    "command": "config-get"
}
```

18.3.3 The config-reload Command

The `config-reload` command instructs Kea to load again the configuration file that was used previously. This operation is useful if the configuration file has been changed by some external source; for example, a sysadmin can tweak the configuration file and use this command to force Kea pick up the changes.

Caution should be taken when mixing this with `config-set` commands. Kea remembers the location of the configuration file it was started with, and this configuration can be significantly changed using the `config-set` command. When `config-reload` is issued after `config-set`, Kea will attempt to reload its original configuration from the file, possibly losing all changes introduced using `config-set` or other commands.

`config-reload` does not take any parameters. An example command invocation looks like this:

```
{
    "command": "config-reload"
}
```

If the configuration file is incorrect reloading it can raise an error which leaves the server in unusable state.. Look at *The config-set Command* what to do to recover a working server.

18.3.4 The config-test Command

The `config-test` command instructs the server to check whether the new configuration supplied in the command's arguments can be loaded. The supplied configuration is expected to be the full configuration for the target server, along with an optional Logger configuration. As for the `-t` command, some sanity checks are not performed, so it is possible a configuration which successfully passes this command will still fail in the `config-set` command or at launch time. The structure of the command is as follows:

```
{
    "command": "config-test",
    "arguments":  {
        "<server>": {
        }
     }
}
```

where <server> is the configuration element name for a given server such as “Dhcp4” or “Dhcp6”. For example:

```
{
    "command": "config-test",
    "arguments":  {
        "Dhcp6": {
            :
        }
     }
}
```

The server’s response will contain a numeric code, “result” (0 for success, non-zero on failure), and a string, “text”, describing the outcome:

```
{"result": 0, "text": "Configuration seems sane..." }

or

{"result": 1, "text": "unsupported parameter: BOGUS (<string>:16:26)" }
```

18.3.5 The config-write Command

The `config-write` command instructs the Kea server to write its current configuration to a file on disk. It takes one optional argument, called “filename”, that specifies the name of the file to write the configuration to. If not specified, the name used when starting Kea (passed as a -c argument) will be used. If a relative path is specified, Kea will write its files only in the directory it is running.

An example command invocation looks like this:

```
{
    "command": "config-write",
    "arguments": {
        "filename": "config-modified-2017-03-15.json"
    }
}
```

18.3.6 The leases-reclaim Command

The `leases-reclaim` command instructs the server to reclaim all expired leases immediately. The command has the following JSON syntax:

```
{
    "command": "leases-reclaim",
    "arguments": {
        "remove": true
    }
}
```

The `remove` boolean parameter is mandatory and indicates whether the reclaimed leases should be removed from the lease database (if true), or left in the "expired-reclaimed" state (if false). The latter facilitates lease affinity, i.e. the ability to re-assign an expired lease to the same client that used this lease before. See *Configuring Lease Affinity* for the details. Also, see *Lease Reclamation* for general information about the processing of expired leases (lease reclamation).

18.3.7 The libreload Command

The `libreload` command first unloads and then loads all currently loaded hooks libraries. This is primarily intended to allow one or more hooks libraries to be replaced with newer versions without requiring Kea servers to be reconfigured or restarted. Note that the hooks libraries are passed the same parameter values (if any) that were passed when they originally loaded.

```
{
    "command": "libreload",
    "arguments": { }
}
```

The server will respond with a result of either 0, indicating success, or 1, indicating failure.

18.3.8 The list-commands Command

The `list-commands` command retrieves a list of all commands supported by the server. It does not take any arguments. An example command may look like this:

```
{
    "command": "list-commands",
    "arguments": { }
}
```

The server responds with a list of all supported commands. The arguments element is a list of strings, each of which conveys one supported command.

18.3.9 The config-set Command

The `config-set` command instructs the server to replace its current configuration with the new configuration supplied in the command's arguments. The supplied configuration is expected to be the full configuration for the target server, along with an optional Logger configuration. While optional, the Logger configuration is highly recommended, as without it the server will revert to its default logging configuration. The structure of the command is as follows:

```
{
    "command": "config-set",
    "arguments":  {
        "<server>": {
        }
     }
}
```

where <server> is the configuration element name for a given server such as "Dhcp4" or "Dhcp6". For example:

```
{
    "command": "config-set",
    "arguments":  {
        "Dhcp6": {
            :
        }
     }
}
```

If the new configuration proves to be invalid, the server retains its current configuration but in some cases a fatal error message is logged indicating that the server no longer provides any service: a working configuration must be loaded as soon as possible. If the control channel is dead the configuration file can still be reloaded using the SIGHUP signal. Of course a last chance solution is to restart the server.

Please note that the new configuration is retained in memory only; if the server is restarted or a configuration reload is triggered via a signal, the server uses the configuration stored in its configuration file. The server's response contains a numeric code, "result" (0 for success, non-zero on failure), and a string, "text", describing the outcome:

```
{"result": 0, "text": "Configuration successful." }

or

{"result": 1, "text": "unsupported parameter: BOGUS (<string>:16:26)" }
```

18.3.10 The shutdown Command

The `shutdown` command instructs the server to initiate its shutdown procedure. It is the equivalent of sending a SIGTERM signal to the process. This command does not take any arguments. An example command may look like this:

```
{
    "command": "shutdown"
    "arguments": {
        "exit-value": 3
    }
}
```

The server responds with a confirmation that the shutdown procedure has been initiated. The optional parameter, "exit-value", specifies the numeric value with which the server process will exit to the system. The default value is zero.

The DDNS deamon supports an extra parameter "type" which controls the way the process cleans up on exit. The supported shutdown types are:

- "normal" - Stops the queue manager and finishes all current transactions before exiting. This is the default.
- "drain_first" - Stops the queue manager but continues processing requests from the queue until it is empty.
- "now" - Exits immediately.

An example command may look like this:

```
{
    "command": "shutdown"
    "arguments": {
        "exit-value": 3,
        "type": "drain_first"
    }
}
```

18.3.11 The dhcp-disable Command

The `dhcp-disable` command globally disables the DHCP service. The server continues to operate, but it drops all received DHCP messages. This command is useful when the server's maintenance requires that the server temporarily stop allocating new leases and renew existing leases. It is also useful in failover-like configurations during a synchronization of the lease databases at startup, or recovery after a failure. The optional parameter "max-period" specifies the time in seconds after which the DHCP service should be automatically re-enabled, if the `dhcp-enable` command is not sent before this time elapses.

```
{
    "command": "dhcp-disable",
    "arguments": {
        "max-period": 20
    }
}
```

18.3.12 The dhcp-enable Command

The `dhcp-enable` command globally enables the DHCP service.

```
{
    "command": "dhcp-enable"
}
```

18.3.13 The status-get Command

The `status-get` command returns server's runtime information:

- pid: process id.
- uptime: number of seconds since the start of the server.
- reload: number of seconds since the last configuration (re)load.
- high-availability: HA specific status information about the DHCP servers configured to use HA hooks library:
  - local: for the local server the state, the role (primary, secondary, ...) and scopes (i.e. what the server is actually processing).
  - remote: for the remote server the last known state, served HA scopes and the role of the server in HA relationship.
- multi-threading-enabled: flag indicating if multi-threading is enabled.
- thread-pool-size: number of dhcp service threads.
- packet-queue-size: size of the packet queue size.

The `high-availability` information is returned only when the command is sent to the DHCP servers being in the HA setup. This parameter is never returned when the `status-get` command is sent to the Control Agent or DDNS deamon.

The `thread-pool-size` and `packet-queue-size` parameters are returned only when the command is sent to DHCP servers with multi-threading enabled. These two parameters and `multi-threading-enabled` are never returned when the `status-get` command is sent to the Control Agent or DDNS deamon.

To learn more about the HA status information returned by the `status-get` command please refer to the the *The status-get Command* section.

18.3.14 The server-tag-get Command:

The `server-tag-get` command returns the configured server tag of the DHCPv4 or DHCPv6 server (*Configuration Sharing and Server Tags* explains the server tag concept)

18.3.15 The config-backend-pull Command:

The `config-backend-pull` command triggers the polling of Config Backends (which should be configured for this command to do something) explained in *Enabling Configuration Backend*.

18.3.16 The version-get Command

The `version-get` command returns extended information about the Kea version. It is the same information available via the `-V` command-line argument. This command does not take any parameters.

```
{
    "command": "version-get"
}
```

18.4 Commands Supported by the D2 Server

The D2 server supports only a subset of DHCPv4 / DHCPv6 server commands:

- build-report
- config-get
- config-reload
- config-set
- config-test
- config-write
- list-commands
- shutdown
- status-get
- version-get

18.5 Commands Supported by the Control Agent

The following commands listed in *Commands Supported by Both the DHCPv4 and DHCPv6 Servers* are also supported by the Control Agent, i.e. when the `service` parameter is blank, the commands are handled by the CA and they relate to the CA process itself:

- build-report
- config-get
- config-reload
- config-set
- config-test
- config-write
- list-commands
- shutdown
- status-get
- version-get

CHAPTER

NINETEEN

LOGGING

19.1 Logging Configuration

During its operation Kea may produce many messages. They differ in severity (some are more important than others) and source (different components, like hooks, produce different messages). It is useful to understand which log messages are critical and which are not, and to configure logging appropriately. For example, debug-level messages can be safely ignored in a typical deployment. They are, however, very useful when debugging a problem.

The logging system in Kea is configured through the loggers entry in the server section of your configuration file. In previous Kea releases this entry was in an independent Logging section; this was still supported for backward compatibility until Kea 1.7.9 included.

19.1.1 Loggers

Within Kea, a message is logged through an entity called a "logger." Different components log messages through different loggers, and each logger can be configured independently of the others. Some components, in particular the DHCP server processes, may use multiple loggers to log messages pertaining to different logical functions of the component. For example, the DHCPv4 server uses one logger for messages about packet reception and transmission, another logger for messages related to lease allocation, and so on. Some of the libraries used by the Kea server, such as libdhcpsrv, use their own loggers.

Users implementing hooks libraries (code attached to the server at runtime) are responsible for creating the loggers used by those libraries. Such loggers should have unique names, different from the logger names used by Kea. In this way the messages produced by the hooks library can be distinguished from messages issued by the core Kea code. Unique names also allow the loggers to be configured independently of loggers used by Kea. Whenever it makes sense, a hooks library can use multiple loggers to log messages pertaining to different logical parts of the library.

In the server section of a configuration file the configuration for zero or more loggers (including loggers used by the proprietary hooks libraries) can be specified. If there are no loggers specified, the code will use default values; these cause Kea to log messages of INFO severity or greater to standard output. There is a small time window after Kea has been started but before it has read its configuration; logging in this short period can be controlled using environment variables. For details, see *Logging During Kea Startup*.

The three main elements of a logger configuration are: `name` (the component that is generating the messages), `severity` (what to log), and `output_commands` (where to log). There is also a `debuglevel` element, which is only relevant if debug-level logging has been selected.

The name (string) Logger

Each logger in the system has a name: that of the component binary file using it to log messages. For instance, to configure logging for the DHCPv4 server, add an entry for a logger named "kea-dhcp4". This configuration will then be used by the loggers in the DHCPv4 server and all the libraries used by it, unless a library defines its own logger and there is a specific logger configuration that applies to that logger.

When tracking down an issue with the server's operation, use of DEBUG logging is required to obtain the verbose output needed for problem diagnosis. However, the high verbosity is likely to overwhelm the logging system in cases where the server is processing high-volume traffic. To mitigate this problem, Kea can use multiple loggers, for different functional parts of the server, that can each be configured independently. If the user is reasonably confident that a problem originates in a specific function of the server, or that the problem is related to a specific type of operation, they may enable high verbosity only for the relevant logger, thereby limiting the debug messages to the required minimum.

The loggers are associated with a particular library or binary of Kea. However, each library or binary may (and usually does) include multiple loggers. For example, the DHCPv4 server binary contains separate loggers for packet parsing, dropped packets, callouts, etc.

The loggers form a hierarchy. For each program in Kea, there is a "root" logger, named after the program (e.g. the root logger for kea-dhcp, the DHCPv4 server) is named kea-dhcp4. All other loggers are children of this logger and are named accordingly, e.g. the allocation engine in the DHCPv4 server logs messages using a logger called kea-dhcp4.alloc-engine.

This relationship is important, as each child logger derives its default configuration from its parent root logger. In the typical case, the root logger configuration is the only logging configuration specified in the configuration file and so applies to all loggers. If an entry is made for a given logger, any attributes specified override those of the root logger, whereas any not specified are inherited from it.

To illustrate this, suppose we are using the DHCPv4 server with the root logger "kea-dhcp4" logging at the INFO level. In order to enable DEBUG verbosity for DHCPv4 packet drops, we must create a configuration entry for the logger called "kea-dhcp4.bad-packets" and specify severity DEBUG for this logger. All other configuration parameters may be omitted for this logger if the logger should use the default values specified in the root logger's configuration.

If there are multiple logger specifications in the configuration that might match a particular logger, the specification with the more specific logger name takes precedence. For example, if there are entries for both "kea-dhcp4" and "kea-dhcp4.dhcpsrv", the main DHCPv4 server program — and all libraries it uses other than the dhcpsrv library (libdhcpsrv) — will log messages according to the configuration in the first entry ("kea-dhcp4"). Messages generated by the dhcpsrv library will be logged according to the configuration set by the second entry.

Currently defined loggers are defined in the following table. The "Software Package" column of this table specifies whether the particular loggers belong to the core Kea code (open source Kea binaries and libraries), or hooks libraries (open source or premium).

Table 19.1: List of Loggers Supported by Kea Servers and Hooks Libraries Shipped With Kea and Premium Packages

Logger Name	Software Package	Description
`kea-ctrl-agent`	core	The root logger for the Control Agent exposing the RESTful control API. All components used by the Control Agent inherit the settings from this logger.
`kea-ctrl-agent.http`	core	A logger which outputs log messages related to receiving, parsing, and sending HTTP messages.
`kea-dhcp4`	core	The root logger for the DHCPv4 server. All components used by the DHCPv4 server inherit the settings from this logger.
`kea-dhcp6`	core	The root logger for the DHCPv6 server. All components used by the DHCPv6 server inherit the settings from this logger.
`kea-dhcp4.alloc-engine`, `kea-dhcp6.alloc-engine`	core	Used by the lease allocation engine, which is responsible for managing leases in the lease database, i.e. creating, modifying, and removing DHCP leases as a result of processing messages from clients.

Continued on next page

Table 19.1 – continued from previous page

Logger Name	Software Package	Description
`kea-dhcp4.bad-packets`, `kea-dhcp6.bad-packets`	core	Used by the DHCP servers for logging inbound client packets that were dropped or to which the server responded with a DHCPNAK. It allows administrators to configure a separate log output that contains only packet drop and reject entries.
`kea-dhcp4.bootp-hooks`	libdhcp_bootp hook library	This logger is used to log messages related to the operation of the BOOTP hook library.
`kea-dhcp4.callouts`, `kea-dhcp6.callouts`	core	Used to log messages pertaining to the callouts registration and execution for the particular hook point.
`kea-dhcp4.commands`, `kea-dhcp6.commands`	core	Used to log messages relating to the handling of commands received by the DHCP server over the command channel.
`kea-dhcp4.database`, `kea-dhcp6.database`	core	Used to log messages relating to general operations on the relational databases and Cassandra.
`kea-dhcp4.ddns`, `kea-dhcp6.ddns`	core	Used by the DHCP server to log messages related to Client FQDN and Hostname option processing. It also includes log messages related to the relevant DNS updates.
`kea-dhcp4.dhcp4`	core	Used by the DHCPv4 server daemon to log basic operations.
`kea-dhcp4.dhcpsrv`, `kea-dhcp6.dhcpsrv`	core	The base loggers for the libkea-dhcpsrv library.
`kea-dhcp4.eval`, `kea-dhcp6.eval`	core	Used to log messages relating to the client classification expression evaluation code.
`kea-dhcp4.host-cache-hooks`, `kea-dhcp6.host-cache-hooks`	libdhcp_host_cache premium hook library	This logger is used to log messages related to the operation of the Host Cache hooks library.
`kea-dhcp4.flex-id-hooks`, `kea-dhcp6.flex-id-hooks`	libdhcp_flex_id premium hook library	This logger is used to log messages related to the operation of the Flexible Identifiers hooks library.
`kea-dhcp4.ha-hooks`, `kea-dhcp6.ha-hooks`	libdhcp_ha hook library	This logger is used to log messages related to the operation of the High Availability hooks library.
`kea-dhcp4.hooks`, `kea-dhcp6.hooks`	core	Used to log messages related to the management of hooks libraries, e.g. registration and deregistration of the libraries, and to the initialization of the callouts execution for various hook points within the DHCP server.
`kea-dhcp4.host-cmds-hooks`, `kea-dhcp6.host-cmds-hooks`	libdhcp_host_cmds premium hook library	This logger is used to log messages related to the operation of the Host Commands hooks library. In general, these will pertain to the loading and unloading of the library and the execution of commands by the library.
`kea-dhcp4.hosts`, `kea-dhcp6.hosts`	core	Used within the libdhcpsrv, it logs messages related to the management of DHCP host reservations, i.e. retrieving reservations and adding new reservations.
`kea-dhcp4.lease-cmds-hooks`, `kea-dhcp6.lease-cmds-hooks`	libdhcp_lease_cmds hook library	This logger is used to log messages related to the operation of the Lease Commands hooks library. In general, these will pertain to the loading and unloading of the library and the execution of commands by the library.
`kea-dhcp4.leases`, `kea-dhcp6.leases`	core	Used by the DHCP server to log messages related to lease allocation. The messages include detailed information about the allocated or offered leases, errors during the lease allocation, etc.
`kea-dhcp4.legal-log-hooks`, `kea-dhcp6.legal-log-hooks`	libdhcp_legal_log premium hook library	This logger is used to log messages related to the operation of the Forensic Logging hooks library.
`kea-dhcp4.options`, `kea-dhcp6.options`	core	Used by the DHCP server to log messages related to the processing of options in the DHCP messages, i.e. parsing options, encoding options into on-wire format, and packet classification using options contained in the received packets.

Continued on next page

Table 19.1 – continued from previous page

Logger Name	Software Package	Description
`kea-dhcp4.packets`, `kea-dhcp6.packets`	core	This logger is mostly used to log messages related to transmission of the DHCP packets, i.e. packet reception and the sending of a response. Such messages include information about the source and destination IP addresses and interfaces used to transmit packets. The logger is also used to log messages related to subnet selection, as this selection is usually based on the IP addresses, relay addresses, and/or interface names, which can be retrieved from the received packet even before the DHCP message carried in the packet is parsed.
`kea-dhcp4.radius-hooks`, `kea-dhcp6.radius-hooks`	libdhcp_radius premium hook library	This logger is used to log messages related to the operation of the RADIUS hooks library.
`kea-dhcp4.stat-cmds-hooks`, `kea-dhcp6.stat-cmds-hooks`	libdhcp_stat_cmds hook library	This logger is used to log messages related to the operation of the Statistics Commands hooks library. In general, these will pertain to loading and unloading the library and the execution of commands by the library.
`kea-dhcp4.subnet-cmds-hooks`, `kea-dhcp6.subnet-cmds-hooks`	libdhcp_subnet_cmds hook library	This logger is used to log messages related to the operation of the Subnet Commands hooks library. In general, these will pertain to loading and unloading the library and the execution of commands by the library.
`kea-dhcp4.mysql-cb-hooks`, `kea-dhcp6.mysql-cb-hooks`	libdhcp_mysql_cb_hooks hook library	This logger is used to log messages related to the operation of the MySQL Configuration Backend hooks library.
`kea-dhcp-ddns`	core	The root logger for the kea-dhcp-ddns daemon. All components used by this daemon inherit the settings from this logger unless there are configurations for more specialized loggers.
`kea-dhcp-ddns.dctl`	core	The logger used by the kea-dhcp-ddns daemon for logging basic information about the process, received signals, and triggered reconfigurations.
`kea-dhcp-ddns.dhcpddns`	core	The logger used by the kea-dhcp-ddns daemon for logging events related to DDNS operations.
`kea-dhcp-ddns.dhcp-to-d2`	core	Used by the kea-dhcp-ddns daemon for logging information about events dealing with receiving messages from the DHCP servers and adding them to the queue for processing.
`kea-dhcp-ddns.d2-to-dns`	core	Used by the kea-dhcp-ddns daemon for logging information about events dealing with sending and receiving messages to and from the DNS servers.
`kea-netconf`	core	The root logger for the NETCONF agent. All components used by NETCONF inherit the settings from this logger if there is no specialized logger provided.
`kea-dhcp4.lease-query-hooks`, `kea-dhcp6.lease-query-hooks`	libdhcp_lease_query hook library	This logger is used to log messages related to the operation of the Leasequery hooks library

Note that user-defined hook libraries should not use any of the loggers mentioned above, but should instead define new loggers with names that correspond to the libraries using them. Suppose that a user created a library called "libdhcp-packet-capture" to dump packets received and transmitted by the server to a file. An appropriate name for the logger could be `kea-dhcp4.packet-capture-hooks`. (Note that the hook library implementer only specifies the second part of this name, i.e. "packet-capture". The first part is a root-logger name and is prepended by the Kea logging system.) It is also important to note that since this new logger is a child of a root logger, it inherits the configuration from the root logger, something that can be overridden by an entry in the configuration file.

The easiest way to find a logger name is to configure all logging to go to a single destination and look there for specific logger names. See *Logging Message Format* for details.

The severity (string) Logger

This specifies the category of messages logged. Each message is logged with an associated severity, which may be one of the following (in descending order of severity):

- FATAL - associated with messages generated by a condition that is so serious that the server cannot continue executing.
- ERROR - associated with messages generated by an error condition. The server will continue executing, but the results may not be as expected.
- WARN - indicates an out-of-the-ordinary condition. However, the server will continue executing normally.
- INFO - an informational message marking some event.
- DEBUG - messages produced for debugging purposes.

When the severity of a logger is set to one of these values, it will only log messages of that severity and above (e.g. setting the logging severity to INFO will log INFO, WARN, ERROR, and FATAL messages). The severity may also be set to NONE, in which case all messages from that logger are inhibited.

Note: The `keactrl` tool, described in *Managing Kea with keactrl*, can be configured to start the servers in verbose mode. If this is the case, the settings of the logging severity in the configuration file will have no effect; the servers will use a logging severity of DEBUG regardless of the logging settings specified in the configuration file. To control severity via the configuration file, please make sure that the `kea_verbose` value is set to "no" within the `keactrl` configuration.

The debuglevel (integer) Logger

When a logger's severity is set to DEBUG, this value specifies what level of debug messages should be printed. It ranges from 0 (least verbose) to 99 (most verbose). If severity for the logger is not DEBUG, this value is ignored.

The output_options (list) Logger

Each logger can have zero or more `output_options`. These specify where log messages are sent and are explained in detail below.

The output (string) Option

This value determines the type of output. There are several special values allowed here: `stdout` (messages are printed on standard output), `stderr` (messages are printed on stderr), `syslog` (messages are logged to syslog using the default name), `syslog:name` (messages are logged to syslog using a specified name). Any other value is interpreted as a filename to which messages should be written.

The flush (true of false) Option

Flush buffers after each log message. Doing this will reduce performance but will ensure that if the program terminates abnormally, all messages up to the point of termination are output. The default is "true".

The maxsize (integer) Option

This option is only relevant when the destination is a file; this is the maximum size in bytes that a log file may reach. When the maximum size is reached, the file is renamed and a new file opened. For example, a ".1" is appended to the name; if a ".1" file exists, it is renamed ".2", etc. This is referred to as rotation.

The default value is 10240000 (10MB). The smallest value that can be specified without disabling rotation is 204800. Any value less than this, including 0, disables rotation.

Note: Due to a limitation of the underlying logging library (log4cplus), rolling over the log files (from ".1" to ".2", etc) may show odd results; there can be multiple small files at the timing of rollover. This can happen when multiple processes try to roll over the files simultaneously. Version 1.1.0 of log4cplus solved this problem, so if this version or later of log4cplus is used to build Kea, the issue should not occur. Even for older versions, it is normally expected to happen rarely unless the log messages are produced very frequently by multiple different processes.

The maxver (integer) Option

This option is only relevant when the destination is a file and rotation is enabled (i.e. maxsize is large enough). This is the maximum number of rotated versions that will be kept. Once that number of files has been reached, the oldest file, "log-name.maxver", will be discarded each time the log rotates. In other words, at most there will be the active log file plus maxver rotated files. The minimum and default value is 1.

The pattern (string) Option

This option can be used to specify the layout pattern of log messages for a logger. Kea logging is implemented using the Log4Cplus library and whose output formatting is based, conceptually, on the printf formatting from C and is discussed in detail in the the next section *Logging Message Format*.

Each output type (stdout, file, or syslog) has a default `pattern` which describes the content of its log messages. This parameter can be used to specifiy your own pattern. The pattern for each logger is governed individually so each configured logger can have it's own pattern. Omitting the `pattern` parameter or setting it to an empty string, "", will cause Kea to use the default pattern for that logger's output type.

In addition to the log text itself, the default patterns used for `stdout` and files contain information such as date and time, logger level, and process information. The default pattern for `syslog` is limited primarily to log level, source, and the log text. This avoids duplicating information which is usually supplied by syslog.

Warning: You are strongly encouraged to test your pattern(s) on a local, non-production instance of Kea, running in the foreground and logging to `stdout`.

19.1.2 Logging Message Format

As mentioned above, Kea log message content is controlled via a scheme similar to the C language's printf formatting. The "pattern" used for each message is described by a string containing one or more format components as part of a text string. In addition to the components the string may contain any other arbitrary text you find useful.

The Log4Cplus documentation provides a concise discussion of the supported components and formatting behavior and can be seen here:

> https://log4cplus.sourceforge.io/docs/html/classlog4cplus_1_1PatternLayout.html

It is probably easiest to understand this by examining the default pattern for stdout and files (currently they are the same). That pattern is shown below:

```
"%D{%Y-%m-%d %H:%M:%S.%q} %-5p [%c/%i.%t] %m\n";
```

and a typical log produced by this pattern would look somethng like this:

```
2019-08-05 14:27:45.871 DEBUG [kea-dhcp4.dhcpsrv/8475.12345] DHCPSRV_TIMERMGR_START_
↪TIMER starting timer: reclaim-expired-leases
```

That breaks down as like so:

- `%D{%Y-%m-%d %H:%M:%S.%q}` '%D' is the date and time in local time that the log message is generated, while everything between the curly braces, '{}' are date and time components. From the example log above this produces: `2019-08-05 14:27:45.871`
- `%-5p` The severity of message, output as a minimum of five characters, using right-padding with spaces. In our example log: `DEBUG`
- `%c` The log source. This includes two elements: the Kea process generating the message, in this case, `kea-dhcp4`; and the component within the program from which the message originated, `dhcpsrv` (e.g. the name of the library used by DHCP server implementations).
- `%i` The process ID. From the example log: `8475`
- `%t` The thread ID. From the example log: `12345`. Note the format of the thread ID is OS dependent: e.g. on some systems it is an address so is displayed in hexadecimal.
- `%m` The log message itself. Keg log messages all begin with a message identifier followed by arbitrary log text. Every message in Kea has a unique identifier, which can be used as an index to the Kea Messages Manual, where more information can be obtained. In our example log above, the identifier is `DHCPSRV_TIMERMGR_START_TIMER`. The log text is typically a brief description detailing the condition that caused the message to be logged. In our example, the information logged, `starting timer: reclaim-expired-leases`, explains that the timer for the expired lease reclamation cycle has been started.

> **Warning:** Omitting `%m` will omit the log message text from your output making it rather useless. You should consider `%m` mandatory.

Finally, note that spacing between components, the square brackets around the log source and PID, and the final carriage return 'n' are all literal text specified as part of the pattern.

> **Warning:** In order to ensure each log entry is a separate line, your patterns must end with an `\n`. There may be use cases where it is not desired so we do not enforce its inclusion. Be aware that if you omit it from your pattern that to common text tools or displays, the log entries will run together in one long, endless "line".

The default for pattern for syslog output is as follows:

```
"%-5p [%c.%t] %m\n";
```

You can see that it omits the date and time as well the process ID as this information is typically output by syslog. Note that Kea uses the pattern to construct the text it sends to syslog (or any other destination). It has no influence on the content syslog may add or formatting it may do.

Consult your OS documentation for "syslog" behavior as there are multiple implementations.

Example Logger Configurations

In this example we want to set the server logging to write to the console using standard output.

```
"Server": {
    "loggers": [
        {
            "name": "kea-dhcp4",
            "output_options": [
                {
                    "output": "stdout"
                }
            ],
            "severity": "WARN"
        }
    ]
}
```

In this second example, we want to store debug log messages in a file that is at most 2MB and keep up to eight copies of old logfiles. Once the logfile grows to 2MB, it will be renamed and a new file will be created.

```
"Server": {
    "loggers": [
        {
            "name": "kea-dhcp6",
            "output_options": [
                {
                    "output": "/var/log/kea-debug.log",
                    "maxver": 8,
                    "maxsize": 204800,
                    "flush": true
                    "pattern": "%d{%j %H:%M:%S.%q} %c %m\n"
                }
            ],
            "severity": "DEBUG",
            "debuglevel": 99
        }
   ]
}
```

Notice that the above configuration uses a custom pattern which produces output like this:

```
220 13:50:31.783 kea-dhcp4.dhcp4 DHCP4_STARTED Kea DHCPv4 server version 1.6.0-beta2-
↪git started
```

19.1.3 Logging During Kea Startup

The logging configuration is specified in the configuration file. However, when Kea starts, the configuration file is not read until partway into the initialization process. Prior to that, the logging settings are set to default values, although it is possible to modify some aspects of the settings by means of environment variables. Note that in the absence of any logging configuration in the configuration file, the settings of the (possibly modified) default configuration will persist while the program is running.

The following environment variables can be used to control the behavior of logging during startup:

KEA_LOCKFILE_DIR

Specifies a directory where the logging system should create its lock file. If not specified, it is prefix/var/run/kea, where “prefix” defaults to /usr/local. This variable must not end with a slash. There is one special value: “none”, which instructs Kea not to create a lock file at all. This may cause issues if several processes log to the same file.

KEA_LOGGER_DESTINATION

Specifies logging output. There are several special values:

`stdout` Log to standard output.

`stderr` Log to standard error.

`syslog[:fac]` Log via syslog. The optional fac (which is separated from the word “syslog” by a colon) specifies the facility to be used for the log messages. Unless specified, messages will be logged using the facility “local0”.

Any other value is treated as a name of the output file. If not specified otherwise, Kea will log to standard output.

CHAPTER TWENTY

THE KEA SHELL

20.1 Overview of the Kea Shell

Kea 1.2.0 introduced the Control Agent (CA, see *The Kea Control Agent*), which provides a RESTful control interface over HTTP. That API is typically expected to be used by various IPAMs and similar management systems. Nevertheless, there may be cases when an administrator wants to send a command to the CA directly, and the Kea shell provides a way to do this. It is a simple command-line, scripting-friendly, text client that is able to connect to the CA, send it commands with parameters, retrieve the responses, and display them.

As the primary purpose of the Kea shell is as a tool in a scripting environment, it is not interactive. However, following simple guidelines it can be run manually.

Kea 1.9.0 introduced basic HTTP authentication support.

20.2 Shell Usage

`kea-shell` is run as follows:

```
$ kea-shell [--host hostname] [--port number] [--path path] [--auth-user] [--auth-
↪password] [--timeout seconds] [--service service-name] [command]
```

where:

- `--host hostname` specifies the hostname of the CA. If not specified, "localhost" is used.
- `--port number` specifies the TCP port on which the CA listens. If not specified, 8000 is used.
- `--path path` specifies the path in the URL to connect to. If not specified, an empty path is used. As the CA listens at the empty path, this parameter is useful only with a reverse proxy.
- `--auth-user` specifies the user id for basic HTTP authentication. If not specified or specified as the empty string authentication is not used.
- **`--auth-password` specifies the password for basic HTTP authentication.** If not specified but the user id is specified an empty password is used.
- `--timeout seconds` specifies the timeout (in seconds) for the connection. If not given, 10 seconds is used.
- `--service service-name` specifies the target of a command. If not given, the CA will be used as the target. May be used more than once to specify multiple targets.
- `command` specifies the command to be sent. If not specified, the `list-commands` command is used.

Other switches are:

- `-h` - prints a help message.

- `-v` - prints the software version.

Once started, the shell reads parameters for the command from standard input, which are expected to be in JSON format. When all have been read, the shell establishes a connection with the CA using HTTP, sends the command, and awaits a response. Once that is received, it is displayed on standard output.

For a list of available commands, see *Management API*; additional commands may be provided by hooks libraries. For a list of all supported commands from the CA, use the `list-commands` command.

The following shows a simple example of usage:

```
$ kea-shell --host 192.0.2.1 --port 8001 --service dhcp4 list-commands
^D
```

After the command line is entered, the program waits for command parameters to be entered. Since `list-commands` does not take any arguments, CTRL-D (represented in the above example by “^D”) is pressed to indicate end-of-file and terminate the parameter input. The shell then contacts the CA and prints out the list of available commands returned for the service named `dhcp4`.

It is envisaged that the Kea shell will be most frequently used in scripts; the next example shows a simple scripted execution. It sends the command “config-write” to the CA (the `--service` parameter has not been used), along with the parameters specified in param.json. The result will be stored in result.json.

```
$ cat param.json
"filename": "my-config-file.json"
$ cat param.json | kea-shell --host 192.0.2.1 config-write > result.json
```

When a reverse proxy is used to de-multiplex requests to different servers, the default empty path in the URL is not enough, so the `--path` parameter should be used. For instance, if requests to the “/kea” path are forwarded to the CA this can be used:

```
$ kea-shell --host 192.0.2.1 --port 8001 --path kea ...
```

The Kea shell requires Python to to be installed on the system. It has been tested with Python 2.7 and various versions of Python 3, up to 3.5. Since not every Kea deployment uses this feature and there are deployments that do not have Python, the Kea shell is not enabled by default. To use it, specify `--enable-shell` when running “configure” during the installation of Kea. When building on Debian systems, also `--with-site-packages=...` may be useful.

The Kea shell is intended to serve more as a demonstration of the RESTful interface’s capabilities (and, perhaps, an illustration for people interested in integrating their management environments with Kea) than as a serious management client. It is not likely to be significantly expanded in the future; it is, and will remain, a simple tool.

Note: When using this tool with basic HTTP authentication please keep in mind that command line arguments are not hidden to local users.

CHAPTER

TWENTYONE

YANG/NETCONF SUPPORT

21.1 Overview

Kea 1.5.0 introduced optional support for a YANG/NETCONF interface with the new `kea-netconf` NETCONF agent.

This bare-bones documentation is a work in progress. Its current purpose is to let engineers joining the project or perhaps advanced early adopters to get up to speed quickly.

21.2 Installing NETCONF

Note that to get its NETCONF capabilities, Kea uses Sysrepo, which has many dependencies. Unfortunately, some of them are not available as packages and need to be compiled manually.

Please note that building libyang requires a minimum gcc version of at least 4.9, so on some environments - like CentOS 7.5 - the system compiler cannot be used.

The following sections provide installation instructions for Ubuntu 18.04 and CentOS 7.5. Due to a more modern compiler and many available packages, the installation procedure is much simpler on Ubuntu.

21.2.1 Installing NETCONF on Ubuntu 18.04

For detailed installation instructions, see the Ubuntu installation notes page.

21.2.2 Installing NETCONF on CentOS 7.5

For detailed installation instructions, see the CentOS installation notes page.

CentOS 7.5's gcc compiler (version 4.8.5) is very old. Some Sysrepo dependencies require at least version 4.9, which unfortunately means that a new compiler has to be installed. Also, many of the Sysrepo dependencies are not avalable in CentOS as packages, so for the time being they must be installed from sources.

21.3 Quick Sysrepo Overview

This section offers a rather brief overview of a subset of available functions in Sysrepo. For more complete information, see the Sysrepo homepage.

In YANG, configurations and state data are described in the YANG syntax in module files named: `"module-name"[@"revision"]`.yang

The revision part is optional and has YYYY-MM-DD format. An alternate XML syntax YIN is defined but less user-friendly. Top-level modules are named in Kea models (a short version of schema models).

To list the currently installed YANG modules:

```
$ sysrepoctl -l
```

After installation the result should be similar to this:

```
Sysrepo schema directory: /home/thomson/devel/sysrepo-0.7.6/build/repository/yang/
Sysrepo data directory:   /home/thomson/devel/sysrepo-0.7.6/build/repository/data/
(Do not alter contents of these directories manually)

Module Name       | Revision   | Conformance | Data Owner | Permissions
-----------------------------------------------------------------------
ietf-netconf-     | 2012-02-06 | Installed   | root:root  | 666
  -notifications  |            |             |            |
ietf-netconf      | 2011-06-01 | Imported    |            |
ietf-netconf-acm  | 2012-02-22 | Imported    |            |
nc-notifications  | 2008-07-14 | Installed   | root:root  | 666
notifications     | 2008-07-14 | Installed   | root:root  | 666
turing-machine    | 2013-12-27 | Installed   | root:root  | 666
iana-if-type      | 2014-05-08 | Installed   |            |
ietf-interfaces   | 2014-05-08 | Installed   | root:root  | 666
ietf-ip           | 2014-06-16 | Installed   |            |
```

There are two major modules that Kea is able to support: kea-dhcp4-server and kea-dhcp6-server. Note that while there is an active effort in the DHC working group at IETF to develop a DHCPv6 YANG model, a similar initiative in the past for DHCPv4 failed. Therefore, Kea uses its own dedicated models for DHCPv4 and DHCPv6 but partially supports the IETF model for DHCPv6. Those three models have extra modules as dependencies. The dependency modules are also provided in `src/share/yang/modules` in sources and in `share/kea/yang/modules` after installation.

To install modules from sources, do the following:

```
$ cd src/share/yang/modules
$ sudo sysrepoctl -i -s /home/thomson/devel/sysrepo-0.7.6/build/repository/yang -s . -
↪g ietf-dhcpv6-server*.yang
$ sudo sysrepoctl -i -s /home/thomson/devel/sysrepo-0.7.6/build/repository/yang -s . -
↪g kea-dhcp4-server*.yang
$ sudo sysrepoctl -i -s /home/thomson/devel/sysrepo-0.7.6/build/repository/yang -s . -
↪g kea-dhcp6-server*.yang
...
```

Note that the first -s parameter specifies the location of the YANG schema repository; it can be verified with sysrepoctl -l. This is a parameter that is configured during Sysrepo compilation and is detected by the Kea configuration under the SYSREPO_REPO name.

The installation should look similar to the following:

```
$ sudo sysrepoctl -i -s /home/thomson/devel/sysrepo-0.7.6/build/repository/yang -s . -
↪g ietf-dhcpv6-server*.yang
Installing a new module from file 'ietf-dhcpv6-server@2018-11-20.yang'...
Installing the YANG file to '/home/thomson/devel/sysrepo-0.7.6/build/repository/yang/
↪ietf-dhcpv6-server@2018-07-14.yang'...
Resolving dependency: 'ietf-dhcpv6-server' imports 'ietf-dhcpv6-options'...
Installing the YANG file to '/home/thomson/devel/sysrepo-0.7.6/build/repository/yang/
↪ietf-dhcpv6-options@2018-07-14.yang'...
Resolving dependency: 'ietf-dhcpv6-options' imports 'ietf-dhcpv6-types'...
Installing the YANG file to '/home/thomson/devel/sysrepo-0.7.6/build/repository/yang/
↪ietf-dhcpv6-types@2018-07-14.yang'...
```

```
Resolving dependency: 'ietf-dhcpv6-server' imports 'ietf-dhcpv6-types'...
Installing the YANG file to '/home/thomson/devel/sysrepo-0.7.6/build/repository/yang/
↪ietf-dhcpv6-types@2018-07-14.yang'...
Resolving dependency: 'ietf-dhcpv6-server' imports 'ietf-interfaces'...
Schema of the module ietf-interfaces is already installed, skipping...
Installing data files for module 'ietf-dhcpv6-server'...
Resolving dependency: 'ietf-dhcpv6-server' imports 'ietf-dhcpv6-options'...
Skipping installation of data files for module 'ietf-dhcpv6-options'...
Resolving dependency: 'ietf-dhcpv6-options' imports 'ietf-dhcpv6-types'...
Skipping installation of data files for module 'ietf-dhcpv6-types'...
Resolving dependency: 'ietf-dhcpv6-server' imports 'ietf-dhcpv6-types'...
Skipping installation of data files for module 'ietf-dhcpv6-types'...
Resolving dependency: 'ietf-dhcpv6-server' imports 'ietf-interfaces'...
Installing data files for module 'ietf-interfaces'...
Notifying sysrepo about the change...
Install operation completed successfully.
```

It is possible to confirm whether the models are imported correctly by using sysrepoctl -l:

```
$ sysrepoctl -l
Sysrepo schema directory: /home/thomson/devel/sysrepo-0.7.6/build/repository/yang/
Sysrepo data directory:   /home/thomson/devel/sysrepo-0.7.6/build/repository/data/
(Do not alter contents of these directories manually)

Module Name                 | Revision   | Conformance | Data Owner          |␣
↪Permissions
-------------------------------------------------------------------------------------
↪---
ietf-netconf-notifications | 2012-02-06 | Installed   | root:root           | 666
ietf-netconf               | 2011-06-01 | Imported    |                     |
ietf-netconf-acm           | 2012-02-22 | Imported    |                     |
nc-notifications           | 2008-07-14 | Installed   | root:root           | 666
notifications              | 2008-07-14 | Installed   | root:root           | 666
turing-machine             | 2013-12-27 | Installed   | root:root           | 666
iana-if-type               | 2014-05-08 | Installed   |                     |
ietf-interfaces            | 2014-05-08 | Installed   | root:root           | 666
ietf-ip                    | 2014-06-16 | Installed   |                     |
kea-dhcp4-server           | 2018-11-20 | Installed   | root:root           | 666
kea-dhcp6-server           | 2018-11-20 | Installed   | root:root           | 666
ietf-dhcpv6-server         | 2018-09-04 | Installed   | root:root           | 666
ietf-dhcpv6-options        | 2018-09-04 | Imported    |                     |
ietf-dhcpv6-types          | 2018-01-30 | Imported    |                     |
```

To install a new revision of a module it must first be uninstalled, e.g. by:

```
sudo sysrepoctl -u -m kea-dhcp4-server
```

If the module is used (i.e. imported) by other modules, it can be uninstalled only after those modules have finished using it. Installation and uninstallation must be done in dependency order and reverse-dependency order accordingly.

21.4 Supported YANG Models

The only currently supported models are `kea-dhcp4-server` and `kea-dhcp6-server`. There is partial support for `ietf-dhcpv6-server`, but the primary focus of testing has been on Kea DHCP servers. Several other models (`kea-dhcp-ddns` and `kea-ctrl-agent`) are currently not supported.

21.5 Using the NETCONF Agent

The NETCONF agent follows this algorithm:

- For each managed server, get the initial configuration from the server through the control socket.
- Open a connection with the Sysrepo environment and establish two sessions with the startup and running datastores.
- Check that used (not essential) and required (essential) modules are installed in the Sysrepo repository at the right revision. If an essential module - that is, a module where the configuration schema for a managed server is defined - is not installed, raise a fatal error.
- For each managed server, get the YANG configuration from the startup datastore, translate it to JSON, and load it onto the server being configured.
- For each managed server, subscribe a module change callback using its model name.
- When a running configuration is changed, try to validate or load the updated configuration via the callback to the managed server.

21.6 Configuration

The behavior described in *Using the NETCONF Agent* is controlled by a few configuration flags, which can be set in the global scope or in a specific managed-server scope. In the second case, the value defined in the managed-server scope takes precedence. These flags are:

- `boot-update` - controls the initial configuration phase; when true (the default), the initial configuration retrieved from the classic Kea server JSON configuration file is loaded first, and then the startup YANG model is loaded. This setting lets administrators define a control socket in the local JSON file and then download the configuration from YANG. When set to false, this phase is skipped.
- `subscribe-changes` - controls the module change subscription; when true (the default), a module change callback is subscribed, but when false the phase is skipped and running configuration updates are disabled. When set to true, the running datastore is used to subscribe for changes.
- `validate-changes` - controls how Kea monitors changes in the Sysrepo configuration. Sysrepo offers two stages where Kea can interact: validation and application. At the validation (or SR_EV_VERIFY event, in the Sysrepo naming convention) stage, Kea retrieves the newly committed configuration and verifies it. If the configuration is incorrect for any reason, the Kea servers reject it and the error is propagated back to the Sysrepo, which then returns an error. This step only takes place if validate-changes is set to true. In the application (or SR_EV_APPLY event in the Sysrepo naming convention) stage, the actual configuration is applied. At this stage Kea can receive the configuration, but it is too late to signal back any errors as the configuration has already been committed.

The idea behind the initial configuration phase is to boot Kea servers with a minimal configuration which includes only a control socket, making them manageable. For instance, for the DHCPv4 server:

```
{
    "Dhcp4": {
        "control-socket": {
            "socket-type": "unix",
            "socket-name": "/tmp/kea4-sock"
        }
    }
}
```

Note the alternative to boot with full configurations does not allow easy tracking of changes or synchronization between the JSON and YANG configuration sources; therefore, that setup is not really compatible with the YANG/NETCONF configuration management paradigm, where everything should be performed in YANG.

With module change subscriptions enabled, the kea-netconf daemon will monitor any configuration changes as they appear in the Sysrepo. Such changes can be done using the `sysrepocfg` tool or remotely using any NETCONF client. For details, please see the Sysrepo documentation or *A Step-by-Step NETCONF Agent Operation Example*. Those tools can be used to modify YANG configurations in the running datastore. Note that committed configurations are only updated in the running datastore; to keep them between server reboots they must be copied to the startup datastore.

When module changes are tracked (using `subscribe-changes` set to true) and the running configuration has changed (e.g. using `sysrepocfg` or any NETCONF client), the callback validates the modified configuration (if `validate-changes` was not set to false) and runs a second time to apply the new configuration. If the validation fails, the callback is still called again but with an ABORT (vs. APPLY) event with rollback changes.

The returned code of the callback on an APPLY event is ignored, as it is too late to refuse a bad configuration.

There are four ways in which a modified YANG configuration could possibly be incorrect:

1. It can be non-compliant with the schema, e.g. an unknown entry, missing a mandatory entry, a value with a bad type, or not matching a constraint.
2. It can fail to be translated from YANG to JSON, e.g. an invalid user context.
3. It can fail Kea server sanity checks, e.g. an out-of-subnet-pool range or an unsupported database type.
4. The syntax may be correct and pass server sanity checks but the configuration fails to run, e.g. the configuration specifies database credentials but the database refuses the connection.

The first case is handled by Sysrepo. The second and third cases are handled by kea-netconf in the validation phase (if not disabled by setting `validate-changes` to true). The last case causes the application phase to fail without a sensible report to Sysrepo.

The managed Kea servers or agents are described in the `managed-servers` section. Each sub-section begins by the service name: `dhcp4`, `dhcp6`, `d2` (the DHCP-DDNS server does not support the control channel feature yet), and `ca` (the control agent).

Each managed server entry contains optionally:

- `boot-update`, `subscribe-changes`, and `validate-changes` - control flags.
- `model` - specifies the YANG model / module name. For each service, the default is the corresponding Kea YANG model, e.g. for `"dhcp4"` it is `"kea-dhcp4-server"`.
- `control-socket` - specifies the control socket for managing the service configuration.

A control socket is specified by:

- `socket-type` - the socket type is either `stdout`, `unix`, or `http`. `stdout` is the default; it is not really a socket, but it allows `kea-netconf` to run in debugging mode where everything is printed on stdout, and it can also be used to redirect commands easily. `unix` is the standard direct server control channel, which uses UNIX sockets, and `http` uses a control agent, which accepts HTTP connections.
- `socket-name` - the local socket name for the `unix` socket type (default empty string).
- `socket-url` - the HTTP URL for the `http` socket type (default `http://127.0.0.1:8000/`).

User contexts can store arbitrary data as long as they are in valid JSON syntax and their top-level element is a map (i.e. the data must be enclosed in curly brackets). They are accepted at the NETCONF entry, i.e. below the top-level, managed-service entry, and control-socket entry scopes.

Hooks libraries can be loaded by the NETCONF agent just as with other servers or agents; however, currently no hook points are defined. The `hooks-libraries` list contains the list of hooks libraries that should be loaded by kea-netconf, along with their configuration information specified with `parameters`.

Please consult *Logging* for details on how to configure logging. The NETCONF agent's root logger's name is `kea-netconf`, as given in the example above.

21.7 A kea-netconf Configuration Example

The following example demonstrates the basic NETCONF configuration. More examples are available in the `doc/examples/netconf` directory in the Kea sources.

```
# This is a simple example of a configuration for the NETCONF agent.
# This server provides a YANG interface for all Kea servers and the agent.
{
    "Netconf":
    {
        # Control flags can be defined in the global scope or
        # in a managed server scope. Precedences are:
        # - use the default value (true)
        # - use the global value
        # - use the local value.
        # So this overwrites the default value:
        "boot-update": false,

        # This map specifies how each server is managed. For each server there
        # is a name of the YANG model to be used and the control channel.
        //
        # Currently three control channel types are supported:
        # "stdout" which outputs the configuration on the standard output,
        # "unix" which uses the local control channel supported by the
        # "dhcp4" and "dhcp6" servers ("d2" support is not yet available),
        # and "http" which uses the Control Agent "ca" to manage itself or
        # to forward commands to "dhcp4" or "dhcp6".
        "managed-servers":
        {
            # This is how kea-netconf can communicate with the DHCPv4 server.
            "dhcp4":
            {
                "comment": "DHCP4 server",
                "model": "kea-dhcp4-server",
                "control-socket":
                {
                    "socket-type": "unix",
                    "socket-name": "/tmp/kea4-ctrl-socket"
                }
            },

            # DHCPv6 parameters.
            "dhcp6":
            {
                "model": "kea-dhcp6-server",
                "control-socket":
                {
                    "socket-type": "unix",
                    "socket-name": "/tmp/kea6-ctrl-socket"
            }
        },

        # Currently the DHCP-DDNS (nicknamed D2) server does not support
        # a command channel.
        "d2":
        {
            "model": "kea-dhcp-ddns",
            "control-socket":
            {
                "socket-type": "stdout",
                "user-context": { "in-use": false }
            }
        },

        # Of course the Control Agent (CA) supports HTTP.
        "ca":
        {
            "model": "kea-ctrl-agent",
            "control-socket":
            {
                "socket-type": "http",
                "socket-url": "http://127.0.0.1:8000/"
            }
        }
    },

    # kea-netconf is able to load hooks libraries that augment its operation.
    # Currently there are no hook points defined in kea-netconf
    # processing.
    "hooks-libraries": [
        # The hooks libraries list may contain more than one library.
        {
            # The only necessary parameter is the library filename.
            "library": "/opt/local/netconf-commands.so",

            # Some libraries may support parameters. Make sure you
            # type this section carefully, as kea-netconf does not
            # validate it (because the format is library-specific).
            "parameters": {
                "param1": "foo"
            }
        }
    ],

    # Similar to other Kea components, NETCONF also uses logging.
    "loggers": [
        {
            "name": "kea-netconf",
            "output_options": [
                {
                    "output": "/var/log/kea-netconf.log",
                    # Several additional parameters are possible in
                    # addition to the typical output.
                    # Flush determines whether logger flushes output
                    #  to a file.
                    # Maxsize determines maximum filesize before
                    # the file is being rotated.
```

```
                        # Maxver specifies the maximum number of
                        #  rotated files being kept.
                        "flush": true,
                        "maxsize": 204800,
                        "maxver": 4
                    }
                ],
                "severity": "INFO",
                "debuglevel": 0
            }
        ]
    }
}
```

21.8 Starting and Stopping the NETCONF Agent

kea-netconf accepts the following command-line switches:

- `-c file` - specifies the configuration file.
- `-d` - specifies whether the agent logging should be switched to debug/verbose mode. In verbose mode, the logging severity and debuglevel specified in the configuration file are ignored and "debug" severity and the maximum debuglevel (99) are assumed. The flag is convenient for temporarily switching the server into maximum verbosity, e.g. when debugging.
- `-t file` - specifies the configuration file to be tested. Kea-netconf attempts to load it and conducts sanity checks; note that certain checks are possible only while running the actual server. The actual status is reported with exit code (0 = configuration looks ok, 1 = error encountered). Kea will print out log messages to standard output and error to standard error when testing configuration.
- `-v` - displays the version of kea-netconf and exits.
- `-V` - displays the extended version information for kea-netconf and exits. The listing includes the versions of the libraries dynamically linked to Kea.
- `-W` - displays the Kea configuration report and exits. The report is a copy of the `config.report` file produced by `./configure`; it is embedded in the executable binary.

21.9 A Step-by-Step NETCONF Agent Operation Example

Note: Copies of example configurations presented within this section can be found in the Kea source code, under `doc/examples/netconf/kea-dhcp6-operations`.

21.9.1 Setup of NETCONF Agent Operation Example

The test box has an Ethernet interface named eth1. On some systems it is possible to rename interfaces, for instance on a Linux with an ens38 interface:

```
# ip link set down dev ens38
# ip link set name eth1 dev ens38
# ip link set up dev eth1
```

The interface must have an address in the test prefix:

```
# ip -6 addr add 2001:db8::1/64 dev eth1
```

The Kea DHCPv6 server must be launched with the configuration specifying a control socket used to receive control commands. The `kea-netconf` process uses this socket to communicate with the DHCPv6 server, i.e. it pushes translated configurations to that server using control commands. The following is the example control socket specification for the Kea DHCPv6 server:

```
{
    "Dhcp6": {
        "control-socket": {
            "socket-type": "unix",
            "socket-name": "/tmp/kea6-sock"
        }
    }
}
```

In order to launch the Kea DHCPv6 server using the configuration contained within the `boot.json` file, run:

```
# kea-dhcp6 -d -c boot.json
```

The current configuration of the server can be fetched via control socket by running:

```
# echo '{ "command": "config-get" }' | socat UNIX:/tmp/kea6-sock '-,ignoreeof'
```

The following is the example `netconf.json` configuration for `kea-netconf`, to manage the Kea DHCPv6 server:

```
{
    "Netconf":
    {
        "managed-servers":
        {
            "dhcp6":
            {
                "control-socket":
                {
                    "socket-type": "unix",
                    "socket-name": "/tmp/kea6-sock"
                }
            }
        },

        "loggers":
        [
            {
                "name": "kea-netconf",
                "output_options":
                [
                    {
                        "output": "stderr"
                    }
                ],
                "severity": "DEBUG",
                "debuglevel": 99
            }
        ]
    }
}
```

Note that in production there should not be a need to log at the DEBUG level.

The Kea NETCONF agent is launched by:

```
# kea-netconf -d -c netconf.json
```

Now that both `kea-netconf` and `kea-dhcp6` are running, it is possible to populate updates to the configuration to the DHCPv6 server. The following is the configuration extracted from `startup.xml`:

```
<config xmlns="urn:ietf:params:xml:ns:yang:kea-dhcp6-server">
  <subnet6>
    <id>1</id>
    <pool>
      <start-address>2001:db8::1:0</start-address>
      <end-address>2001:db8::1:ffff</end-address>
      <prefix>2001:db8::1:0/112</prefix>
    </pool>
    <subnet>2001:db8::/64</subnet>
  </subnet6>
  <interfaces-config>
    <interfaces>eth1</interfaces>
  </interfaces-config>
  <control-socket>
    <socket-name>/tmp/kea6-sock</socket-name>
    <socket-type>unix</socket-type>
  </control-socket>
</config>
```

To populate this new configuration:

```
# sysrepocfg -d startup -f xml -i startup.xml kea-dhcp6-server
```

`kea-netconf` pushes the configuration found in the Sysrepo startup datastore to all Kea servers during its initialization phase, after it subscribes to module changes in the Sysrepo running datastore. This action copies the configuration from the startup datastore to the running datastore and enables the running datastore, making it available.

Changes to the running datastore are applied after validation to the Kea servers. Note that they are not by default copied back to the startup datastore, i.e. changes are not permanent.

21.9.2 Error Handling in NETCONF Operation Example

There are four classes of issues with the configurations applied via NETCONF:

1. The configuration does not comply with the YANG schema.
2. The configuration cannot be translated from YANG to the Kea JSON.
3. The configuration is rejected by the Kea server.
4. The configuration was validated by the Kea server but cannot be applied.

In the first case, consider the following `BAD-schema.xml` configuration file:

```
<config xmlns="urn:ietf:params:xml:ns:yang:kea-dhcp6-server">
  <subnet4>
    <id>1</id>
    <pool>
```

```
      <start-address>2001:db8::1:0</start-address>
      <end-address>2001:db8::1:ffff</end-address>
      <prefix>2001:db8::1:0/112</prefix>
    </pool>
    <subnet>2001:db8::/64</subnet>
  </subnet6>
  <interfaces-config>
    <interfaces>eth1</interfaces>
  </interfaces-config>
  <control-socket>
    <socket-name>/tmp/kea6-sock</socket-name>
    <socket-type>unix</socket-type>
  </control-socket>
</config>
```

It is directly rejected by `sysrepocfg`:

```
# sysrepocfg -d running -f xml -i BAD-schema.xml kea-dhcp6-server
```

In the second case, the configuration is rejected by `kea-netconf`. For example, consider this `BAD-translator.xml` file:

```
<config xmlns="urn:ietf:params:xml:ns:yang:kea-dhcp6-server">
  <subnet6>
    <id>1</id>
    <pool>
      <start-address>2001:db8::1:0</start-address>
      <end-address>2001:db8::1:ffff</end-address>
      <prefix>2001:db8::1:0/112</prefix>
    </pool>
    <subnet>2001:db8::/64</subnet>
  </subnet6>
  <interfaces-config>
    <interfaces>eth1</interfaces>
  </interfaces-config>
  <control-socket>
    <socket-name>/tmp/kea6-sock</socket-name>
    <socket-type>unix</socket-type>
  </control-socket>
  <user-context>bad</user-context>
</config>
```

In the third case, the configuration is presented to the Kea DHCPv6 server and fails to validate as in this `BAD-config.xml` file:

```
<config xmlns="urn:ietf:params:xml:ns:yang:kea-dhcp6-server">
  <subnet6>
    <id>1</id>
    <pool>
      <start-address>2001:db8:1::0</start-address>
      <end-address>2001:db8:1::ffff</end-address>
      <prefix>2001:db8:1::0/112</prefix>
    </pool>
    <subnet>2001:db8::/64</subnet>
  </subnet6>
  <interfaces-config>
    <interfaces>eth1</interfaces>
  </interfaces-config>
  <control-socket>
    <socket-name>/tmp/kea6-sock</socket-name>
    <socket-type>unix</socket-type>
  </control-socket>
</config>
```

In the last case, the misconfiguration is detected too late and the change must be reverted in Sysrepo, e.g. using the startup datastore as a backup. For this reason, please use the `sysrepocfg --permanent` / `-p` option (or any similar feature of NETCONF clients) with care.

21.9.3 NETCONF Operation Example with Two Pools

This example adds a second pool to the initial (i.e. startup) configuration in the `twopools.xml` file:

```
<config xmlns="urn:ietf:params:xml:ns:yang:kea-dhcp6-server">
  <subnet6>
    <id>1</id>
    <pool>
      <start-address>2001:db8::1:0</start-address>
      <end-address>2001:db8::1:ffff</end-address>
      <prefix>2001:db8::1:0/112</prefix>
    </pool>
    <pool>
      <start-address>2001:db8::2:0</start-address>
      <end-address>2001:db8::2:ffff</end-address>
      <prefix>2001:db8::2:0/112</prefix>
    </pool>
    <subnet>2001:db8::/64</subnet>
  </subnet6>
  <interfaces-config>
    <interfaces>eth1</interfaces>
  </interfaces-config>
  <control-socket>
    <socket-name>/tmp/kea6-sock</socket-name>
    <socket-type>unix</socket-type>
  </control-socket>
</config>
```

This configuration is installed by:

```
# sysrepocfg -d running -f xml -i twopools.xml kea-dhcp6-server
```

21.9.4 NETCONF Operation Example with Two Subnets

This example specifies two subnets in the `twosubnets.xml` file:

```
<config xmlns="urn:ietf:params:xml:ns:yang:kea-dhcp6-server">
  <subnet6>
    <id>1</id>
    <pool>
      <start-address>2001:db8:1::</start-address>
      <end-address>2001:db8:1::ffff</end-address>
      <prefix>2001:db8:1::/112</prefix>
    </pool>
    <subnet>2001:db8:1::/64</subnet>
  </subnet6>
  <subnet6>
    <id>2</id>
    <pool>
      <start-address>2001:db8:2::</start-address>
      <end-address>2001:db8:2::ffff</end-address>
      <prefix>2001:db8:2::/112</prefix>
    </pool>
    <subnet>2001:db8:2::/64</subnet>
  </subnet6>
  <interfaces-config>
    <interfaces>eth1</interfaces>
  </interfaces-config>
  <control-socket>
    <socket-name>/tmp/kea6-sock</socket-name>
    <socket-type>unix</socket-type>
  </control-socket>
</config>
```

This configuration is installed by:

```
# sysrepocfg -d running -f xml -i twosubnets.xml kea-dhcp6-server
```

21.9.5 NETCONF Operation Example with Logging

This example adds a logger entry to the initial (i.e. startup) configuration in the `logging.xml` file:

```
<config xmlns="urn:ietf:params:xml:ns:yang:kea-dhcp6-server">
  <interfaces-config>
    <interfaces>eth1</interfaces>
  </interfaces-config>
  <subnet6>
    <id>1</id>
    <pool>
      <start-address>2001:db8::1:0</start-address>
      <end-address>2001:db8::1:ffff</end-address>
      <prefix>2001:db8::1:0/112</prefix>
    </pool>
    <subnet>2001:db8::/64</subnet>
  </subnet6>
  <control-socket>
    <socket-name>/tmp/kea6-sock</socket-name>
    <socket-type>unix</socket-type>
  </control-socket>
  <logger>
    <name>kea-dhcp6</name>
    <output-option>
      <output>stderr</output>
    </output-option>
    <debuglevel>99</debuglevel>
    <severity>DEBUG</severity>
  </logger>
</config>
```

The corresponding Kea configuration in JSON is:

```
{
  "Dhcp6": {
    "control-socket": {
      "socket-name": "/tmp/kea6-sock",
      "socket-type": "unix"
    },
    "interfaces-config": {
      "interfaces": [ "eth1" ]
    },
    "subnet6": [
      {
        "id": 1,
        "pools": [
          {
            "pool": "2001:db8::1:0/112"
          }
        ],
        "subnet": "2001:db8::/64"
      }
    ],
    "loggers": [
      {
        "name": "kea-dhcp6",
        "output_options": [
          {
            "output": "stderr"
          }
        ],
        "severity": "DEBUG",
        "debuglevel": 99
      }
   ]
 }
}
```

Finally, any of the previous examples can be replayed using `sysrepocfg` in edit mode as follows:

```
# sysrepocfg -d running -f xml -e vi kea-dhcp6-server
```

or, of course, using a NETCONF client like `netopeer2-cli` from the Netopeer2 NETCONF Toolset.

CHAPTER

TWENTYTWO

MONITORING KEA WITH STORK

It is usually desired to monitor running Kea services. Kea exposes many sources of informations: configuration files, API, statistics, logs, open database content and more. It may be sometimes overwhelming to keep up. ISC started the Stork project to address this problem for both Kea and BIND 9. Stork can address a variety of scenarios:

- you can use Stork as a dashboard. It provides you with an insight into what exactly is going on with your servers. In particular, it can provide up to date details regarding pool utilization in your subnets and shared networks, can monitor state of your HA pair (and provide extra insight in case of failover and recovery events), lets you list, filter and search for specific host reservations and more. For this capability, you need to deploy a single Stork server and one Stork agent on each machine you want to monitor.
- you can use Stork agent to integrate Kea with Prometheus and Grafana. Once you deploy a Stork agent, it starts acting as Prometheus exporter. If you deploy Prometheus in your network, you will be able to visualize statistics as time-series using Grafana.
- you can do both of the above. If you have Grafana deployed in your network, you can let Stork know its location. In this configuration, Stork will let you inspect the current status and will provide a customized link to Grafana to see how a given property behaved over time.

Stork is available in source code, but also as native Deb and RPM packages, which makes it easy to install on most popular systems. For more details, please see Stork ARM or Stork project page. The former has a nice collection of screenshots that is frequently updated. It will let you quickly formulate a first impression of what is currently available. Stork is being rapidly developed with monthly release. Please check back frequently.

22.1 Kea statistics in Grafana

The ISC Stork project provides an agent, that can be deployed along side Kea and BIND 9. It will expose Kea statistics in a format that's acceptable by Prometheus. One of the major benefits of Prometheus is that it turns repeated one time observation into time series, which lets you monitor how certain behaviors change over time. It is easy to use other tools to visualize data available in Prometheus. The most common approach is to use Grafana to provide visual dashboards. The Stork project provides dashboard definitions for Kea and BIND 9 that can be imported into Grafana very easily.

You can learn more about Prometheus and Grafana on their websites: Prometheus homepage and Grafana homepage.

CHAPTER TWENTYTHREE

API REFERENCE

Kea currently supports 166 commands in *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6* daemons and *cb_cmds*, *class_cmds*, *high_availability*, *host_cache*, *host_cmds*, *lease_cmds*, *stat_cmds*, *subnet_cmds* hook libraries. Commands supported by *kea-ctrl-agent* daemon: *build-report*, *config-get*, *config-reload*, *config-set*, *config-test*, *config-write*, *list-commands*, *shutdown*, *status-get*, *version-get*. Commands supported by *kea-dhcp-ddns* daemon: *build-report*, *config-get*, *config-reload*, *config-set*, *config-test*, *config-write*, *list-commands*, *shutdown*, *status-get*, *version-get*. Commands supported by *kea-dhcp4* daemon: *build-report*, *cache-clear*, *cache-flush*, *cache-get*, *cache-get-by-id*, *cache-insert*, *cache-load*, *cache-remove*, *cache-size*, *cache-write*, *class-add*, *class-del*, *class-get*, *class-list*, *class-update*, *config-backend-pull*, *config-get*, *config-reload*, *config-set*, *config-test*, *config-write*, *dhcp-disable*, *dhcp-enable*, *ha-continue*, *ha-heartbeat*, *ha-maintenance-cancel*, *ha-maintenance-notify*, *ha-maintenance-start*, *ha-scopes*, *ha-sync*, *lease4-add*, *lease4-del*, *lease4-get*, *lease4-get-all*, *lease4-get-by-client-id*, *lease4-get-by-hostname*, *lease4-get-by-hw-address*, *lease4-get-page*, *lease4-resend-ddns*, *lease4-update*, *lease4-wipe*, *leases-reclaim*, *libreload*, *list-commands*, *network4-add*, *network4-del*, *network4-get*, *network4-list*, *network4-subnet-add*, *network4-subnet-del*, *remote-global-parameter4-del*, *remote-global-parameter4-get*, *remote-global-parameter4-get-all*, *remote-global-parameter4-set*, *remote-network4-del*, *remote-network4-get*, *remote-network4-list*, *remote-network4-set*, *remote-option-def4-del*, *remote-option-def4-get*, *remote-option-def4-get-all*, *remote-option-def4-set*, *remote-option4-global-del*, *remote-option4-global-get*, *remote-option4-global-get-all*, *remote-option4-global-set*, *remote-option4-network-del*, *remote-option4-network-set*, *remote-option4-pool-del*, *remote-option4-pool-set*, *remote-option4-subnet-del*, *remote-option4-subnet-set*, *remote-server4-del*, *remote-server4-get*, *remote-server4-get-all*, *remote-server4-set*, *remote-subnet4-del-by-id*, *remote-subnet4-del-by-prefix*, *remote-subnet4-get-by-id*, *remote-subnet4-get-by-prefix*, *remote-subnet4-list*, *remote-subnet4-set*, *reservation-add*, *reservation-del*, *reservation-get*, *reservation-get-all*, *reservation-get-by-hostname*, *reservation-get-by-id*, *reservation-get-page*, *server-tag-get*, *shutdown*, *stat-lease4-get*, *statistic-get*, *statistic-get-all*, *statistic-remove*, *statistic-remove-all*, *statistic-reset*, *statistic-reset-all*, *statistic-sample-age-set*, *statistic-sample-age-set-all*, *statistic-sample-count-set*, *statistic-sample-count-set-all*, *status-get*, *subnet4-add*, *subnet4-del*, *subnet4-get*, *subnet4-list*, *subnet4-update*, *version-get*. Commands supported by *kea-dhcp6* daemon: *build-report*, *cache-clear*, *cache-flush*, *cache-get*, *cache-get-by-id*, *cache-insert*, *cache-load*, *cache-remove*, *cache-size*, *cache-write*, *class-add*, *class-del*, *class-get*, *class-list*, *class-update*, *config-backend-pull*, *config-get*, *config-reload*, *config-set*, *config-test*, *config-write*, *dhcp-disable*, *dhcp-enable*, *ha-continue*, *ha-heartbeat*, *ha-maintenance-cancel*, *ha-maintenance-notify*, *ha-maintenance-start*, *ha-scopes*, *ha-sync*, *lease6-add*, *lease6-bulk-apply*, *lease6-del*, *lease6-get*, *lease6-get-all*, *lease6-get-by-duid*, *lease6-get-by-hostname*, *lease6-get-page*, *lease6-resend-ddns*, *lease6-update*, *lease6-wipe*, *leases-reclaim*, *libreload*, *list-commands*, *network6-add*, *network6-del*, *network6-get*, *network6-list*, *network6-subnet-add*, *network6-subnet-del*, *remote-global-parameter6-del*, *remote-global-parameter6-get*, *remote-global-parameter6-get-all*, *remote-global-parameter6-set*, *remote-network6-del*, *remote-network6-get*, *remote-network6-list*, *remote-network6-set*, *remote-option-def6-del*, *remote-option-def6-get*, *remote-option-def6-get-all*, *remote-option-def6-set*, *remote-option6-global-del*, *remote-option6-global-get*, *remote-option6-global-get-all*, *remote-option6-global-set*, *remote-option6-network-del*, *remote-option6-network-set*, *remote-option6-pd-pool-del*, *remote-option6-pd-pool-set*, *remote-option6-pool-del*, *remote-option6-pool-set*, *remote-option6-subnet-del*, *remote-option6-subnet-set*, *remote-server6-del*, *remote-server6-get*, *remote-server6-get-all*, *remote-server6-set*, *remote-subnet6-del-by-id*, *remote-subnet6-del-by-prefix*, *remote-subnet6-get-by-id*, *remote-subnet6-get-by-prefix*, *remote-subnet6-list*, *remote-subnet6-set*, *reservation-add*, *reservation-del*, *reservation-get*, *reservation-get-all*, *reservation-get-by-hostname*, *reservation-get-by-id*, *reservation-get-page*, *server-tag-get*, *shutdown*, *stat-lease6-get*, *statistic-get*, *statistic-get-all*, *statistic-remove*, *statistic-remove-all*, *statistic-reset*, *statistic-reset-all*, *statistic-sample-age-set*, *statistic-sample-age-set-all*, *statistic-sample-count-set*, *statistic-sample-count-set-all*, *status-get*, *subnet6-add*, *subnet6-del*, *subnet6-get*, *subnet6-list*, *subnet6-update*, *version-get*. Commands supported by *cb_cmds* hook library: *remote-global-parameter4-del*, *remote-global-parameter4-get*, *remote-global-parameter4-get-all*, *remote-global-parameter4-set*, *remote-global-parameter6-del*, *remote-global-parameter6-get*, *remote-global-parameter6-get-all*, *remote-global-parameter6-set*, *remote-network4-del*, *remote-network4-get*, *remote-network4-list*, *remote-network4-set*, *remote-network6-del*, *remote-network6-get*, *remote-network6-list*, *remote-network6-set*, *remote-option-def4-del*, *remote-option-def4-get*, *remote-option-def4-get-all*, *remote-option-def4-set*, *remote-option-def6-del*, *remote-option-def6-get*, *remote-option-def6-get-all*, *remote-option-def6-set*, *remote-option4-global-del*, *remote-option4-global-get*, *remote-option4-global-get-all*, *remote-option4-global-set*, *remote-option4-network-del*, *remote-option4-network-set*, *remote-option4-pool-del*, *remote-option4-pool-set*, *remote-option4-subnet-del*, *remote-option4-subnet-set*, *remote-option6-global-del*, *remote-option6-global-get*, *remote-option6-global-get-all*, *remote-option6-global-set*, *remote-option6-network-del*, *remote-option6-network-set*, *remote-option6-pd-pool-del*, *remote-option6-pd-pool-set*, *remote-option6-pool-del*, *remote-option6-pool-set*, *remote-option6-subnet-del*, *remote-option6-subnet-set*, *remote-server4-del*, *remote-server4-get*, *remote-server4-get-all*, *remote-server4-set*, *remote-server6-del*, *remote-server6-get*, *remote-server6-get-all*, *remote-server6-set*, *remote-subnet4-del-by-id*, *remote-subnet4-del-by-prefix*, *remote-subnet4-get-by-id*, *remote-subnet4-get-by-prefix*, *remote-subnet4-list*, *remote-subnet4-set*, *remote-subnet6-del-by-id*, *remote-subnet6-del-by-prefix*, *remote-subnet6-get-by-id*, *remote-subnet6-get-by-prefix*, *remote-subnet6-list*, *remote-subnet6-set*. Commands supported by *class_cmds* hook library: *class-add*, *class-del*, *class-get*, *class-list*, *class-update*. Commands supported by *high_availability* hook library: *ha-continue*, *ha-heartbeat*, *ha-maintenance-cancel*, *ha-maintenance-notify*, *ha-maintenance-start*, *ha-scopes*, *ha-sync*. Commands supported by *host_cache* hook library: *cache-clear*, *cache-flush*, *cache-get*, *cache-get-by-id*, *cache-insert*, *cache-load*, *cache-remove*, *cache-size*, *cache-write*. Commands supported by *host_cmds* hook library: *reservation-add*, *reservation-del*, *reservation-get*, *reservation-get-all*, *reservation-get-by-hostname*, *reservation-get-by-id*, *reservation-get-page*. Commands supported by *lease_cmds* hook library: *lease4-add*, *lease4-del*, *lease4-get*, *lease4-get-all*, *lease4-get-by-client-id*, *lease4-get-by-hostname*, *lease4-get-by-hw-address*, *lease4-get-page*, *lease4-resend-ddns*, *lease4-update*, *lease4-wipe*, *lease6-add*, *lease6-bulk-apply*, *lease6-del*, *lease6-get*, *lease6-get-all*, *lease6-get-by-duid*, *lease6-get-by-hostname*, *lease6-get-page*, *lease6-resend-ddns*, *lease6-update*, *lease6-wipe*. Commands supported by *stat_cmds* hook library: *stat-lease4-get*, *stat-lease6-get*. Commands supported by *subnet_cmds* hook library: *network4-add*, *network4-del*, *network4-get*, *network4-list*, *network4-subnet-add*, *network4-subnet-del*, *network6-add*, *network6-del*, *network6-get*, *network6-list*, *network6-subnet-add*, *network6-subnet-del*, *subnet4-add*, *subnet4-del*, *subnet4-get*, *subnet4-list*, *subnet4-update*, *subnet6-add*, *subnet6-del*, *subnet6-get*, *subnet6-list*, *subnet6-update*.

23.1 build-report

This command returns the list of compilation options that this particular binary was built with.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (built-in)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *build-report command*

Command syntax:

```
{
    "command": "build-report"
}
```

Response syntax:

```
{
    "result": 0,
    "text": <string with build details>
}
```

23.2 cache-clear

This command removes all cached host reservations.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*host_cache* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *cache-clear command*

Command syntax:

```
{
    "command": "cache-clear"
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.3 cache-flush

This command removes up to the given number or all cached host reservations.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*host_cache* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *cache-flush command*

Command syntax:

```
{
    "command": "cache-flush",
    "arguments": 5
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.4 cache-get

This command returns the full content of the host cache.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*host_cache* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *cache-get command*

Command syntax:

```
{
    "command": "cache-get"
}
```

Response syntax:

```
{
    "result": 0,
    "text": "123 entries returned.",
    "arguments": <list of host reservations>
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.5 cache-get-by-id

This command returns entries matching the given identifier from the host cache.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (*host_cache* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *cache-get-by-id command*

Command syntax:

```
{
    "command": "cache-get-by-id",
    "arguments": {
        "hw-address": "01:02:03:04:05:06"
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "2 entries returned.",
    "arguments": <list of host reservations>
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.6 cache-insert

This command inserts a host into the cache.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*host_cache* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *cache-insert command*

Command syntax:

```
{
    "command": "cache-insert",
    "arguments": {
        "hw-address": "01:02:03:04:05:06",
        "subnet-id4": 4,
        "subnet-id6": 0,
        "ip-address": "192.0.2.100",
        "hostname": "somehost.example.org",
        "client-classes4": [ ],
        "client-classes6": [ ],
        "option-data4": [ ],
        "option-data6": [ ],
        "next-server": "192.0.0.2",
        "server-hostname": "server-hostname.example.org",
        "boot-file-name": "bootfile.efi",
        "host-id": 0
    }
},
{
    "command": "cache-insert",
    "arguments": {
        "hw-address": "01:02:03:04:05:06",
        "subnet-id4": 0,
        "subnet-id6": 6,
        "ip-addresses": [ "2001:db8::cafe:babe" ],
        "prefixes": [ "2001:db8:dead:beef::/64" ],
        "hostname": "",
        "client-classes4": [ ],
        "client-classes6": [ ],
        "option-data4": [ ],
        "option-data6": [ ],
        "next-server": "0.0.0.0",
        "server-hostname": "",
        "boot-file-name": "",
        "host-id": 0
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.7 cache-load

This command allows the contents of a file on disk to be loaded into an in-memory cache.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*host_cache* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *cache-load command*

Command syntax:

```
{
    "command": "cache-load",
    "arguments": "/tmp/kea-host-cache.json"
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.8 cache-remove

This command removes entries from the host cache. It takes parameters similar to the `reservation-get` command.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*host_cache* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *cache-remove command*

Command syntax:

```
{
    "command": "cache-remove",
    "arguments": {
        "ip-address": "192.0.2.1",
        "subnet-id": 123
    }
}

Another example that removes the IPv6 host identifier by DUID and specific subnet-id␣
↪is:
{
    "command": "cache-remove",
    "arguments": {
        "duid": "00:01:ab:cd:f0:a1:c2:d3:e4",
        "subnet-id": 123
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success

- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.9 cache-size

This command returns the number of entries in the host cache.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (*host_cache* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *cache-size command*

Command syntax:

```
{
    "command": "cache-size"
}
```

Response syntax:

```
{
    "result": 0,
    "text": "123 entries.",
    "arguments": { "size": 123 }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.10 cache-write

This command instructs Kea to write its host cache content to disk.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*host_cache* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *cache-write command*

Command syntax:

```
{
    "command": "cache-write",
    "arguments": "/path/to/the/file.json"
}
```

The command takes one mandatory argument that specifies the filename path of a file to be written.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.11 class-add

This command adds a new class to the existing server configuration.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.5.0 (*class_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *class-add command*

Command syntax:

```
{
    "command": "class-add",
    "arguments": {
        "client-classes": [ {
           "name": <name of the class>,
           "test": <test expression to be evaluated on incoming packets>,
           "option-data": [ <option values here> ],
           "option-def": [ <option definitions here> ],
           "next-server": <ipv4 address>,
           "server-hostname": <string>,
           "boot-file-name": <name of the boot file>
        } ]
    }
}
```

The `next-server`, `server-hostname`, and `boot-file-name` are DHCPv4-specific. Only one client class can be added with a single command.

Response syntax:

```
{
    "result": 0,
    "text": "Class '<class-name>' added."
}
```

The command is successful (result 0), unless the class name is a duplicate or another error occurs (result 1).

23.12 class-del

This command removes a client class from the server configuration.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.5.0 (*class_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *class-del command*

Command syntax:

```
{
    "command": "class-del",
    "arguments": {
        "name": <name of the class>
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "Class '<class-name>' deleted."
}
```

The command returns a result of 3 (empty) if the client class does not exist. If the client class exists, the returned result is 0 if the deletion was successful; the result is 1 if the deletion is unsuccessful.

23.13 class-get

This command returns detailed information about an existing client class.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.5.0 (*class_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *class-get command*

Command syntax:

```
{
    "command": "class-get",
    "arguments": {
        "name": <name of the class>
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "Class '<class-name>' definition returned",
    "arguments": {
        "client-classes": [
            {
                "name": <name of the class>,
                "only-if-required": <only if required boolean value>,
                "test": <test expression to be evaluated on incoming packets>,
                "option-data": [ <option values here> ],
                "option-def": [ <option definitions here> ],
                "next-server": <ipv4 address>,
                "server-hostname": <string>,
                "boot-file-name": <name of the boot file>
            }
        ]
    }
}
```

The returned information depends on the DHCP server type, i.e. some parameters are specific to the DHCPv4 server. Also, some parameters may not be returned if they are not set for the client class. If a class with the specified name does not exist, a result of 3 (empty) is returned. If the client class is found, the result of 0 is returned. If there is an error while processing the command, the result of 1 is returned.

23.14 class-list

This command retrieves a list of all client classes from the server configuration.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.5.0 (*class_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *class-list command*

Command syntax:

```
{
    "command": "class-list"
}
```

This command includes no arguments.

Response syntax:

```
{
    "result": 0,
    "text": "'<number of>' classes found",
    "arguments": {
        "client-classes": [
            {
                "name": <first class name>
            },
            {
                "name": <second class name>
            }
        ]
    }
}
```

The returned list of classes merely contains their names. In order to retrieve full information about one of these classes, use *The class-get Command*. The returned result is 3 (empty) if no classes are found. If the command is processed successfully and the list of client classes is not empty, the result of 0 is returned. If there is an error while processing the command, the result of 1 is returned.

23.15 class-update

This command updates an existing client class in the server configuration.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.5.0 (*class_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *class-update command*

Command syntax:

```
{
    "command": "class-update",
    "arguments": {
        "client-classes": [ {
           "name": <name of the class>,
           "test": <test expression to be evaluated on incoming packets>,
           "option-data": [ <option values here> ],
           "option-def": [ <option definitions here> ],
           "next-server": <ipv4 address>,
           "server-hostname": <string>,
           "boot-file-name": <name of the boot file>
        } ]
    }
}
```

The `next-server`, `server-hostname`, and `boot-file-name` are DHCPv4-specific. Only one client class can be updated with a single command.

Response syntax:

```
{
    "result": 0,
    "text": "Class '<class-name>' updated."
}
```

The command returns the result of 3 (empty) if the client class does not exist. If the client class exists, the returned result is 0 if the update was successful, or 1 if the update is unsuccessful.

23.16 config-backend-pull

This command forces an immediate update of the server using Config Backends. This command does not take any parameters.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.7.1 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *config-backend-pull command*

Command syntax:

```
{
    "command": "config-backend-pull"
}
```

Response syntax:

```
{
    "result": 0,
    "text": "On demand configuration update successful."
}
```

When no Config Backends are configured this command returns empty (3); If an error occurs error (1) is returned with the error details; otherwise success (0) is returned.

23.17 config-get

This command retrieves the current configuration used by the server. The configuration is essentially the same as the contents of the configuration file, but includes additional changes made by other commands and due to parameters' inheritance.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (built-in)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *config-get command*

Command syntax:

```
{
    "command": "config-get"
}
```

This command takes no parameters.

Response syntax:

```
{
    "result": <integer>,
    "arguments": {
        <Dhcp4, Dhcp6, or Control-agent object>: <JSON configuration here>
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.18 config-reload

This command instructs Kea to reload the configuration file that was used previously.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *config-reload command*

Command syntax:

```
{
    "command": "config-reload"
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.19 config-set

This command instructs the server to replace its current configuration with the new configuration supplied in the command's arguments.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *config-set command*

Command syntax:

```
{
    "command": "config-set",
    "arguments":  {
        "'<server>'": {
        }
     }
}
```

In the example below, '<server>' is the configuration element name for a given server such as "Dhcp4" or "Dhcp6".

Response syntax:

```
{"result": 0, "text": "Configuration successful." }

or

{"result": 1, "text": "unsupported parameter: BOGUS (<string>:16:26)" }
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.20 config-test

This command instructs the server to check whether the new configuration supplied in the command's arguments can be loaded.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *config-test command*

Command syntax:

```
{
    "command": "config-test",
    "arguments":  {
        "'<server>'": {
        }
     }
}
```

In the example below, <server> is the configuration element name for a given server such as "Dhcp4" or "Dhcp6".

Response syntax:

```
{ "result": 0, "text": "Configuration seems sane..." }

or

{ "result": 1, "text": "unsupported parameter: BOGUS (<string>:16:26)" }
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.21 config-write

This command instructs the Kea server to write its current configuration to a file on disk.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *config-write command*

Command syntax:

```
{
    "command": "config-write",
    "arguments": {
        "filename": "config-modified-2017-03-15.json"
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.22 dhcp-disable

This command globally disables the DHCP service.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *dhcp-disable command*

Command syntax:

```
{
    "command": "dhcp-disable",
    "arguments": {
        "max-period": 20
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.23 dhcp-enable

This command globally enables the DHCP service.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *dhcp-enable command*

Command syntax:

```
{
    "command": "dhcp-enable"
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.24 ha-continue

This command resumes the operation of a paused HA state machine.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*high_availability* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *ha-continue command*

Command syntax:

```
{
    "command": "ha-continue"
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.25 ha-heartbeat

This command is sent internally by a Kea partner when operating in High Availability (HA) mode or by the system administrator to verify the state of the servers with regards to the High Availability. It retrieves the server's HA state and clock value.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*high_availability* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *ha-heartbeat command*

Command syntax:

```
{
    "command": "ha-heartbeat"
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

The response to this command is different from the typical command response. The response includes the server state (see *Server States*) plus the current clock value.

23.26 ha-maintenance-cancel

This command is sent to instruct a server in the partner-in-maintenance state to transition back to the previous state, effectively canceling the maintenance.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.7.4 (*high_availability* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *ha-maintenance-cancel command*

Command syntax:

```
{
    "command": "ha-maintenance-cancel"
}
```

This command takes no arguments.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.27 ha-maintenance-notify

This command is sent by the server receiving the ha-maintenance-start to its partner to cause the partner to transition to the in-maintenance state or to revert it from the in-maintenance state as a result of receiving the ha-maintenance-cancel command.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.7.4 (*high_availability* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *ha-maintenance-notify command*

Command syntax:

```
{
    "command": "ha-maintenance-notify",
    "arguments": {
        "cancel": <boolean>
    }
}
```

This command includes a boolean argument which, if false, indicates that the server should transition to the in-maintenance state. If the argument is set to true it instructs the server to revert from the in-maintenance state to its previous state. This command is not meant to be used by the administrator. It is merely used for internal communication between the HA partners.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

The response may include a special error code of 1001 to indicate that the partner refused to enter the maintenance state.

23.28 ha-maintenance-start

This command is sent to instruct one of the servers to transition to the partner-in-maintenance state in which it will be responding to all DHCP queries. The server receiving this command sends the ha-maintenance-notify to its partner to cause the partner to transition to the in-maintenance state.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.7.4 (*high_availability* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *ha-maintenance-start command*

Command syntax:

```
{
    "command": "ha-maintenance-start"
}
```

This command takes no arguments.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.29 ha-scopes

This command modifies the scope that the server is responsible for serving when operating in High Availability (HA) mode.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*high_availability* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *ha-scopes command*

Command syntax:

```
{
    "command": "ha-scopes",
    "service": [ <service, typically 'dhcp4' or 'dhcp6'> ],
    "arguments": {
        "scopes": [ "HA_server1", "HA_server2" ]
    }
}
```

In the example below, the arguments configure the server to handle traffic from both the HA_server1 and HA_server2 scopes.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.30 ha-sync

This command instructs the server running in HA mode to synchronize its local lease database with the selected peer.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.4.0 (*high_availability* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *ha-sync command*

Command syntax:

```
{
    "command": "ha-sync",
    "service": [ <service affected: 'dhcp4' or 'dhcp6'> ],
    "arguments": {
        "server-name": <name of the partner server>,
        "max-period": <integer, in seconds>
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.31 lease4-add

This command administratively adds a new IPv4 lease.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease4-add command*

Command syntax:

```
{
    "command": "lease4-add",
    "arguments": {
        "ip-address": "192.0.2.202",
        "hw-address": "1a:1b:1c:1d:1e:1f"
    }
}
```

Note that Kea 1.4 requires an additional argument, subnet-ID, which is optional as of Kea 1.5. A number of other, more-detailed, optional arguments are also supported.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.32 lease4-del

This command deletes a lease from the lease database.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease4-del command*

Command syntax:

```
{
    "command": "lease4-del",
    "arguments": {
        "ip-address": "192.0.2.202"
    }
}
```

The lease to be deleted can be specified either by IP address or by identifier-type and identifier value. The currently supported identifiers are “hw-address” and “client-id”.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.33 lease4-get

This command queries the lease database and retrieves existing leases.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease4-get command*

Command syntax:

```
{
    "command": "lease4-get",
    "arguments": {
        "ip-address": "192.0.2.1"
    }
}
```

Response syntax:

```
{
  "arguments": {
    "client-id": "42:42:42:42:42:42:42:42",
    "cltt": 12345678,
    "fqdn-fwd": false,
    "fqdn-rev": true,
    "hostname": "myhost.example.com.",
    "hw-address": "08:08:08:08:08:08",
    "ip-address": "192.0.2.1",
    "state": 0,
    "subnet-id": 44,
    "valid-lft": 3600
  },
  "result": 0,
  "text": "IPv4 lease found."
}
```

lease4-get returns a result that indicates the outcome of the operation and lease details, if found. It has one of the following values: 0 (success), 1 (error), or 3 (empty). Result 3 is returned if no leases are found with specified IP address.

23.34 lease4-get-all

This command retrieves all IPv4 leases or all leases for the specified set of subnets.

Supported by: *kea-dhcp4*

Availability: 1.4.0 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease4-get-all command*

Command syntax:

```
{
    "command": "lease4-get-all",
    "arguments": {
        "subnets": [ 1, 2, 3, ... ]
    }
}
```

The `lease4-get-all` command may result in very large responses. Please consider using `lease4-get-page` to get them in chunks. The `subnets` parameter is optional. If not specified, leases from all subnets are returned.

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "client-id": "01:00:0c:01:02:03:04",
          "cltt": 1600432232,
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "",
          "hw-address": "00:0c:01:02:03:04",
          "ip-address": "192.168.1.150",
          "state": 0,
          "subnet-id": 1,
          "valid-lft": 4000
        },
        {
          "client-id": "01:00:0c:01:02:03:05",
          "cltt": 1600432234,
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "",
          "hw-address": "00:0c:01:02:03:05",
          "ip-address": "192.168.1.151",
          "state": 0,
          "subnet-id": 1,
          "valid-lft": 4000
        }
      ]
    },
    "result": 0,
    "text": "2 IPv4 lease(s) found."
  }
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found with specified parameter.

23.35 lease4-get-by-client-id

This command retrieves all IPv4 leases with the specified client id.

Supported by: *kea-dhcp4*

Availability: 1.7.1 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease4-get-by-client-id command*

Command syntax:

```
{
    "command": "lease4-get-by-client-id",
    "arguments": {
        "client-id": "01:00:0c:01:02:03:04"
    }
}
```

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "client-id": "01:00:0c:01:02:03:04",
          "cltt": 1600432232,
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "",
          "hw-address": "00:0c:01:02:03:04",
          "ip-address": "192.168.1.150",
          "state": 0,
          "subnet-id": 1,
          "valid-lft": 4000
        },
      ]
    },
    "result": 0,
    "text": "1 IPv4 lease(s) found."
  }
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found.

23.36 lease4-get-by-hostname

This command retrieves all IPv4 leases with the specified hostname.

Supported by: *kea-dhcp4*

Availability: 1.7.1 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease4-get-by-hostname command*

Command syntax:

```
{
    "command": "lease4-get-by-hostname",
    "arguments": {
        "hostname": "myhost.example.com."
    }
}
```

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "client-id": "01:00:0c:01:02:03:04",
          "cltt": 1600432232,
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "myhost.example.com.",
          "hw-address": "00:0c:01:02:03:04",
          "ip-address": "192.168.1.150",
          "state": 0,
            "subnet-id": 1,
            "valid-lft": 4000
          },
        ]
      },
      "result": 0,
      "text": "1 IPv4 lease(s) found."
  }
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found.

23.37 lease4-get-by-hw-address

This command retrieves all IPv4 leases with the specified hardware address.

Supported by: *kea-dhcp4*

Availability: 1.7.1 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease4-get-by-hw-address command*

Command syntax:

```
{
    "command": "lease4-get-by-hw-address",
    "arguments": {
        "hw-address": "00:0c:01:02:03:04"
    }
}
```

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "client-id": "01:00:0c:01:02:03:04",
          "cltt": 1600432232,
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "myhost.example.com.",
          "hw-address": "00:0c:01:02:03:04",
          "ip-address": "192.168.1.150",
          "state": 0,
          "subnet-id": 1,
          "valid-lft": 4000
        },
      ]
    },
    "result": 0,
    "text": "1 IPv4 lease(s) found."
  }
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found.

23.38 lease4-get-page

This command retrieves all IPv4 leases by page.

Supported by: *kea-dhcp4*

Availability: 1.5.0 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease4-get-page command*

Command syntax:

```
{
    "command": "lease4-get-page",
    "arguments": {
        "limit": <integer>,
        "from": <IPv4 address or "start">
    }
}
```

The from address and the page size limit are mandatory.

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "client-id": "01:00:0c:01:02:03:04",
          "cltt": 1600432232,
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "",
          "hw-address": "00:0c:01:02:03:04",
          "ip-address": "192.168.1.150",
          "state": 0,
          "subnet-id": 1,
          "valid-lft": 4000
        },
        {
          "client-id": "01:00:0c:01:02:03:05",
          "cltt": 1600432234,
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "",
          "hw-address": "00:0c:01:02:03:05",
          "ip-address": "192.168.1.151",
          "state": 0,
          "subnet-id": 1,
          "valid-lft": 4000
        }
        ]
      },
      "result": 0,
      "text": "2 IPv4 lease(s) found."
  }
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found with specified parameters.

23.39 lease4-resend-ddns

This command resends a request to kea-dhcp-ddns to update DNS for an existing lease.

Supported by: *kea-dhcp4*

Availability: 1.7.6 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease4-resend-ddns command*

Command syntax:

```
{
    "command": "lease4-resend-ddns",
    "arguments": {
        "ip-address": "192.0.2.1"
    }
}
```

Response syntax:

```
{
  "arguments": {
  },
  "result": 0,
  "text": "NCR generated for: 192.0.2.1, hostname: example.com."
}
```

lease4-resend-ddns returns a result that indicates the outcome of the operation and lease details, if found. It has one of the following values: 0 (success), 1 (error), or 3 (empty).

23.40 lease4-update

This command updates existing leases.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease4-update command*

Command syntax:

```
{
  "command": "lease4-update",
  "arguments": {
    "ip-address": "192.0.2.1",
    "hostname": "newhostname.example.org",
    "hw-address": "1a:1b:1c:1d:1e:1f",
    "subnet-id": 44,
    "force-create": true
  }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.41 lease4-wipe

This command removes all leases associated with a given subnet.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease4-wipe command*

Command syntax:

```
{
    "command": "lease4-wipe",
    "arguments": {
        "subnet-id": 44
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success

- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.42 lease6-add

This command administratively creates a new lease.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease6-add command*

Command syntax:

```
{
    "command": "lease6-add",
    "arguments": {
        "subnet-id": 66,
        "ip-address": "2001:db8::3",
        "duid": "1a:1b:1c:1d:1e:1f:20:21:22:23:24",
        "iaid": 1234
    }
}
```

lease6-add can be also used to add leases for IPv6 prefixes.

Response syntax:

```
{ "result": 0, "text": "Lease added." }
or
{ "result": 1, "text": "missing parameter 'ip-address' (<string>:3:19)" }
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.43 lease6-bulk-apply

This command creates, updates, or deletes multiple IPv6 leases in a single transaction. It communicates lease changes between HA peers, but may be used in all cases where it is desirable to apply multiple lease updates in a single transaction.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease6-bulk-apply command*

Command syntax:

```
{
    "command": "lease6-bulk-apply",
    "arguments": {
        "deleted-leases": [
            {
                "ip-address": "2001:db8:abcd::",
                "type": "IA_PD",
                ...
            },
            {
                "ip-address": "2001:db8:abcd::234",
                "type": "IA_NA",
                ...
            }
        ],
        "leases": [
            {
                "subnet-id": 66,
                "ip-address": "2001:db8:cafe::",
                "type": "IA_PD",
                ...
            },
            {
                "subnet-id": 66,
                "ip-address": "2001:db8:abcd::333",
                "type": "IA_NA",
                ...
            }
        ]
    }
}
```

If any of the leases is malformed, all changes are rolled back. If the leases are well-formed but the operation fails for one or more leases, these leases are listed in the response; however, the changes are preserved for all leases for which the operation was successful. The “deleted-leases” and “leases” are optional parameters, but one of them must be specified.

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 leases bulk apply completed.",
    "arguments": {
        "failed-deleted-leases": [
            {
                "ip-address": "2001:db8:abcd::",
                "type": "IA_PD",
                "result": <control result>,
                "error-message": <error message>
            }
        ],
        "failed-leases": [
            {
                "ip-address": "2001:db8:cafe::",
                "type": "IA_PD",
                "result": <control result>,
                "error-message": <error message>
            }
        ]
    }
}
```

The "failed-deleted-leases" holds the list of leases which failed to delete; this includes leases which were not found in the database. The "failed-leases" includes the list of leases which failed to create or update. For each lease for which there was an error during processing, insertion into the database, etc., the result is set to 1. For each lease which was not deleted because the server did not find it in the database, the result of 3 is returned.

23.44 lease6-del

This command deletes a lease from the lease database.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease6-del command*

Command syntax:

```
{
    "command": "lease6-del",
    "arguments": {
        "ip-address": "192.0.2.202"
    }
}
```

lease6-del returns a result that indicates the outcome of the operation. It has one of the following values: 0 (success), 1 (error), or 3 (empty).

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.45 lease6-get

This command queries the lease database and retrieves existing leases.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease6-get command*

Command syntax:

```
{
  "command": "lease6-get",
  "arguments": {
    "ip-address": "2001:db8:1234:ab::",
    "type": "IA_PD"
  }
}
```

lease6-get returns a result that indicates the outcome of the operation and lease details, if found.

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "cltt": 1600439560,
          "duid": "00:01:00:01:26:f7:81:88:00:0c:01:02:03:04",
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "foobar.example.org",
          "hw-address": "00:0c:01:02:03:04",
          "iaid": 1,
          "ip-address": "2001:db8:1::",
          "preferred-lft": 3000,
          "state": 0,
          "subnet-id": 1,
          "type": "IA_NA",
          "valid-lft": 4000
        }
      ]
    },
    "result": 0,
    "text": "1 IPv6 lease(s) found."
  }
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found with specified parameter.

23.46 lease6-get-all

This command retrieves all IPv6 leases or all leases for the specified set of subnets.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease6-get-all command*

Command syntax:

```
{
    "command": "lease6-get-all",
    "arguments": {
        "subnets": [ 1, 2, 3, 4 ]
    }
}
```

The `lease6-get-all` command may result in very large responses. Please consider using `lease6-get-page` to get them in chunks. the `subnets` parameter is optional. If not specified, leases from all subnets are returned.

Response syntax:

```
{
    "arguments": {
        "leases": [
            {
                "cltt": 12345678,
                "duid": "42:42:42:42:42:42:42:42",
                "fqdn-fwd": false,
                "fqdn-rev": true,
                "hostname": "myhost.example.com.",
                "hw-address": "08:08:08:08:08:08",
                "iaid": 1,
                "ip-address": "2001:db8:2::1",
                "preferred-lft": 500,
                "state": 0,
                "subnet-id": 44,
                "type": "IA_NA",
                "valid-lft": 3600
            },
            {
                "cltt": 12345678,
                "duid": "21:21:21:21:21:21:21:21",
                "fqdn-fwd": false,
                "fqdn-rev": true,
                "hostname": "",
                "iaid": 1,
                "ip-address": "2001:db8:0:0:2::",
                "preferred-lft": 500,
                "prefix-len": 80,
                "state": 0,
                "subnet-id": 44,
                "type": "IA_PD",
                "valid-lft": 3600
            }
        ]
    },
    "result": 0,
    "text": "2 IPv6 lease(s) found."
}
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found with specified parameter.

23.47 lease6-get-by-duid

This command retrieves all IPv6 leases with the specified hardware address.

Supported by: *kea-dhcp6*

Availability: 1.7.1 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease6-get-by-duid command*

Command syntax:

```
{
    "command": "lease6-get-by-duid",
    "arguments": {
        "duid": "1a:1b:1c:1d:1e:1f:20:21:22:23:24"
    }
}
```

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "cltt": 1600439560,
          "duid": "00:01:00:01:26:f7:81:88:00:0c:01:02:03:04",
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "foobar.example.org",
          "hw-address": "00:0c:01:02:03:04",
          "iaid": 1,
          "ip-address": "2001:db8:1::",
          "preferred-lft": 3000,
          "state": 0,
          "subnet-id": 1,
          "type": "IA_NA",
          "valid-lft": 4000
        }
      ]
    },
    "result": 0,
    "text": "1 IPv6 lease(s) found."
  }
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found with specified parameter.

23.48 lease6-get-by-hostname

This command retrieves all IPv6 leases with the specified hostname.

Supported by: *kea-dhcp6*

Availability: 1.7.1 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease6-get-by-hostname command*

Command syntax:

```
{
    "command": "lease6-get-by-hostname",
    "arguments": {
        "hostname": "myhost.example.com."
    }
}
```

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "cltt": 1600439560,
          "duid": "00:01:00:01:26:f7:81:88:00:0c:01:02:03:04",
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "foobar.example.org",
          "hw-address": "00:0c:01:02:03:04",
          "iaid": 1,
          "ip-address": "2001:db8:1::",
          "preferred-lft": 3000,
          "state": 0,
          "subnet-id": 1,
          "type": "IA_NA",
          "valid-lft": 4000
        }
      ]
    },
    "result": 0,
    "text": "1 IPv6 lease(s) found."
  }
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found with specified parameter.

23.49 lease6-get-page

This command retrieves all IPv6 leases by page.

Supported by: *kea-dhcp6*

Availability: 1.5.0 (*lease_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *lease6-get-page command*

Command syntax:

```
{
    "command": "lease6-get-page",
    "arguments": {
        "limit": <integer>,
        "from": <IPv6 address or "start">
    }
}
```

The from address and the page size limit are mandatory.

Response syntax:

```
[
  {
    "arguments": {
      "leases": [
        {
          "cltt": 1600439560,
          "duid": "00:01:00:01:26:f7:81:88:00:0c:01:02:03:04",
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "foo.example.org",
          "hw-address": "00:0c:01:02:03:04",
          "iaid": 1,
          "ip-address": "2001:db8:1::1",
          "preferred-lft": 3000,
          "state": 0,
          "subnet-id": 1,
          "type": "IA_NA",
          "valid-lft": 4000
        }
        {
          "cltt": 1600439570,
          "duid": "00:01:00:01:26:f7:81:88:00:0c:01:02:03:05",
          "fqdn-fwd": false,
          "fqdn-rev": false,
          "hostname": "bar.example.org",
          "hw-address": "00:0c:01:02:03:05",
          "iaid": 1,
          "ip-address": "2001:db8:1::2",
          "preferred-lft": 3000,
          "state": 0,
          "subnet-id": 1,
          "type": "IA_NA",
          "valid-lft": 4000
        }
      ]
    },
    "result": 0,
    "text": "1 IPv6 lease(s) found."
}
]
```

Result 0 is returned when at least one lease is found, 1 when parameters are malformed or missing, 3 is returned if no leases are found.

23.50 lease6-resend-ddns

This command resends a request to kea-dhcp-ddns to update DNS for an existing lease.

Supported by: *kea-dhcp6*

Availability: 1.7.6 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease6-resend-ddns command*

Command syntax:

```
{
    "command": "lease6-resend-ddns",
    "arguments": {
        "ip-address": "2001:db8::1"
    }
}
```

Response syntax:

```
{
  "arguments": {
  },
  "result": 0,
  "text": "NCR generated for: 2001:db8::1, hostname: example.com."
}
```

lease6-resend-ddns returns a result that indicates the outcome of the operation and lease details, if found. It has one of the following values: 0 (success), 1 (error), or 3 (empty).

23.51 lease6-update

This command updates existing leases.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease6-update command*

Command syntax:

```
{
  "command": "lease6-update",
  "arguments": {
    "ip-address": "2001:db8::1",
    "duid": "88:88:88:88:88:88:88:88",
    "iaid": 7654321,
    "hostname": "newhostname.example.org",
    "subnet-id": 66,
    "force-create": false
  }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.52 lease6-wipe

This command removes all leases associated with a given subnet.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*lease_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *lease6-wipe command*

Command syntax:

```
{
  "command": "lease6-wipe",
  "arguments": {
    "subnet-id": 66
  }
}
```

Note: not all backends support this command.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.53 leases-reclaim

This command instructs the server to reclaim all expired leases immediately.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *leases-reclaim command*

Command syntax:

```
{
    "command": "leases-reclaim",
    "arguments": {
        "remove": true
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.54 libreload

This command first unloads and then reloads all currently loaded hooks libraries.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *libreload command*

Command syntax:

```
{
    "command": "libreload",
    "arguments": { }
}
```

The server responds with 0, indicating success, or 1, indicating a failure.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.55 list-commands

This command retrieves a list of all commands supported by the server.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *list-commands command*

Command syntax:

```
{
    "command": "list-commands",
    "arguments": { }
}
```

The server responds with a list of all supported commands.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.56 network4-add

This command adds a new shared network.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *network4-add command*

Command syntax:

```
{
    "command": "network4-add",
    "arguments": {
        "shared-networks": [ {
            "name": "floor13",
            "subnet4": [
            {
                "id": 100,
                "pools": [ { "pool": "192.0.2.2-192.0.2.99" } ],
                "subnet": "192.0.2.0/24",
                "option-data": [
                    {
                        "name": "routers",
                        "data": "192.0.2.1"
                    }
                ]
            },
            {
                "id": 101,
                "pools": [ { "pool": "192.0.3.2-192.0.3.99" } ],
                "subnet": "192.0.3.0/24",
                "option-data": [
                    {
                        "name": "routers",
                        "data": "192.0.3.1"
                    }
                ]
            } ]
        } ]
    }
}
```

Response syntax:

```
{
    "arguments": {
        "shared-networks": [ { "name": "floor13" } ]
    },
    "result": 0,
    "text": "A new IPv4 shared network 'floor13' added"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.57 network4-del

This command deletes existing shared networks.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *network4-del command*

Command syntax:

```
{
    "command": "network4-del",
    "arguments": {
        "name": "floor13"
    }
}
```

Response syntax:

```
{
    "arguments": {
        "shared-networks": [
            {
                "name": "floor13"
            }
        ]
    },
    "result": 0,
    "text": "IPv4 shared network 'floor13' deleted"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.58 network4-get

This command retrieves detailed information about shared networks, including subnets that are currently part of a given network.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *network4-get command*

Command syntax:

```
{
    "command": "network4-get",
    "arguments": {
        "name": "floor13"
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "Info about IPv4 shared network 'floor13' returned",
    "arguments": {
        "shared-networks": [
        {
            "match-client-id": true,
            "name": "floor13",
            "option-data": [ ],
            "rebind-timer": 90,
            "relay": {
                "ip-address": "0.0.0.0"
            },
            "renew-timer": 60,
            "reservation-mode": "all",
            "subnet4": [
                {
                    "subnet": "192.0.2.0/24",
                    "id": 5,
                    // many other subnet-specific details here
                },
                {
                    "subnet": "192.0.3.0/31",
                    "id": 6,
                    // many other subnet-specific details here
                }
            ],
            "valid-lifetime": 120
        }
        ]
    }
}
```

Note that the actual response contains many additional fields that are omitted here for clarity.

23.59 network4-list

This command retrieves the full list of currently configured shared networks.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *network4-list command*

Command syntax:

```
{
    "command": "network4-list"
}
```

Response syntax:

```
{
    "arguments": {
        "shared-networks": [
            { "name": "floor1" },
            { "name": "office" }
        ]
    },
    "result": 0,
    "text": "2 IPv4 network(s) found"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.60 network4-subnet-add

This command adds existing subnets to existing shared networks.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *network4-subnet-add command*

Command syntax:

```
{
    "command": "network4-subnet-add",
    "arguments": {
        "name": "floor13",
        "id": 5
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet 10.0.0.0/8 (id 5) is now part of shared network 'floor1'"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success

- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.61 network4-subnet-del

This command removes a subnet that is part of an existing shared network and demotes it to a plain, stand-alone subnet.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *network4-subnet-del command*

Command syntax:

```
{
    "command": "network4-subnet-del",
    "arguments": {
        "name": "floor13",
        "id": 5
    }
 }
```

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet 10.0.0.0/8 (id 5) is now removed from shared network 'floor13
↪'"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.62 network6-add

This command adds a new shared network.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *network6-add command*

Command syntax:

```
{
    "command": "network4-add",
    "arguments": {
        "shared-networks": [ {
            "name": "floor13",
            "subnet4": [
            {
                "id": 100,
                "pools": [ { "pool": "192.0.2.2-192.0.2.99" } ],
                "subnet": "192.0.2.0/24",
                "option-data": [
                    {
                        "name": "routers",
                        "data": "192.0.2.1"
                    }
                ]
            },
            {
                "id": 101,
                "pools": [ { "pool": "192.0.3.2-192.0.3.99" } ],
                "subnet": "192.0.3.0/24",
                "option-data": [
                    {
                        "name": "routers",
                        "data": "192.0.3.1"
                    }
                ]
            } ]
        } ]
    }
}
```

The `network6-add` command uses the same syntax as `network4-add` for both the query and the response. However, there are some parameters that are IPv4-only (e.g. match-client-id) and some that are IPv6-only (e.g. interface-id).

Response syntax:

```
{
    "arguments": {
        "shared-networks": [ { "name": "floor13" } ]
    },
    "result": 0,
    "text": "A new IPv4 shared network 'floor13' added"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.63 network6-del

This command deletes existing shared networks.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *network6-del command*

Command syntax:

```
{
    "command": "network4-del",
    "arguments": {
        "name": "floor13"
    }
}
```

The `network6-del` command uses exactly the same syntax as `network4-del` for both the query and the response.

Response syntax:

```
{
    "command": "network4-del",
    "arguments": {
        "name": "floor13",
        "subnets-action": "delete"
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.64 network6-get

The network6-get command retrieves detailed information about shared networks, including subnets that are currently part of a given network.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *network6-get command*

Command syntax:

```
{
    "command": "network4-get",
    "arguments": {
        "name": "floor13"
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "Info about IPv4 shared network 'floor13' returned",
    "arguments": {
        "shared-networks": [
        {
            "match-client-id": true,
            "name": "floor13",
            "option-data": [ ],
            "rebind-timer": 90,
            "relay": {
                "ip-address": "0.0.0.0"
            },
            "renew-timer": 60,
            "reservation-mode": "all",
            "subnet4": [
                {
                    "subnet": "192.0.2.0/24",
                    "id": 5,
                    // many other subnet specific details here
                },
                {
                    "subnet": "192.0.3.0/31",
                    "id": 6,
                    // many other subnet specific details here
                }
            ],
            "valid-lifetime": 120
        }
        ]
    }
}
```

Note that the actual response contains many additional fields that are omitted here for clarity.

23.65 network6-list

This command retrieves the full list of currently configured shared networks.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *network6-list command*

Command syntax:

```
{
    "command": "network4-list"
}
```

The `network6-list` command uses exactly the same syntax as `network4-list` for both the query and the response.

Response syntax:

```
{
    "arguments": {
        "shared-networks": [
            { "name": "floor1" },
            { "name": "office" }
        ]
    },
    "result": 0,
    "text": "2 IPv4 network(s) found"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.66 network6-subnet-add

This command adds existing subnets to existing shared networks.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *network6-subnet-add command*

Command syntax:

```
{
    "command": "network4-subnet-add",
    "arguments": {
        "name": "floor13",
        "id": 5
    }
}
```

The `network6-subnet-add` command uses exactly the same syntax as `network4-subnet-add` for both the query and the response.

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet 10.0.0.0/8 (id 5) is now part of shared network 'floor1'"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.67 network6-subnet-del

This command removes a subnet that is part of an existing shared network and demotes it to a plain, stand-alone subnet.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *network6-subnet-del command*

Command syntax:

```
{
    "command": "network4-subnet-del",
    "arguments": {
        "name": "floor13",
        "id": 5
    }
 }
```

The `network6-subnet-del` command uses exactly the same syntax as `network4-subnet-del` for both the query and the response.

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet 10.0.0.0/8 (id 5) is now removed from shared network 'floor13
↪'"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.68 remote-global-parameter4-del

This command deletes a global DHCPv4 parameter from the configuration database. The server uses the value specified in the configuration file, or a default value if the parameter is not specified, after deleting the parameter from the database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-global-parameter4-del command*

Command syntax:

```
{
    "command": "remote-global-parameter4-del",
    "arguments": {
        "parameters": [ <parameter name as string> ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command carries the list including exactly one name of the parameter to be deleted. The `server-tags` list is mandatory and it must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 global parameter(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.69 remote-global-parameter4-get

This command fetches the selected global parameter for the server from the specified database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-global-parameter4-get command*

Command syntax:

```
{
    "command": "remote-global-parameter4-get",
    "arguments": {
        "parameters": [ <parameter name as string> ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command carries a list including exactly one name of the parameter to be fetched. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed; it fetches the global parameter value shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 global parameter found.",
    "arguments": {
        "parameters": {
            <parameter name>: <parameter value>,
            "metadata": {
                "server-tags": [ <server tag> ]
            }
        },
        "count": 1
    }
}
```

The returned response contains a map with a global parameter name/value pair. The value may be a JSON string, integer, real, or boolean. The metadata is included and provides database-specific information associated with the returned object. If the "all" server tag is specified, the command attempts to fetch the global parameter value associated with all servers. If the explicit server tag is specified, the command fetches the value associated with the given server. If the server-specific value does not exist, the `remote-global-parameter4-get` command fetches the value associated with all servers.

23.70 remote-global-parameter4-get-all

This command fetches all global parameters for the server from the specified database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-global-parameter4-get-all command*

Command syntax:

```
{
    "command": "remote-global-parameter4-get-all",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The special server tag "all" is allowed; it fetches the global parameters shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 global parameters found.",
    "arguments": {
        "parameters": [
            {
                <first parameter name>: <first parameter value>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            },
            {
                <second parameter name>: <second parameter value>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains a list of maps. Each map contains a global parameter name/value pair. The value may be a JSON string, integer, real, or boolean. The metadata is appended to each parameter and provides database-specific information associated with the returned objects. If the server tag "all" is included in the command, the response contains the global parameters shared among all servers. It excludes server-specific global parameters. If an explicit server tag is included in the command, the response contains all global parameters directly associated with the given server, and the global parameters associated with all servers when server-specific values are not present.

23.71 remote-global-parameter4-set

This command creates or updates one or more global parameters in the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-global-parameter4-set command*

Command syntax:

```
{
    "command": "remote-global-parameter4-set",
    "arguments": {
        "parameters": {
            <first parameter name>: <first parameter value>,
            <second parameter name>: <second parameter value>
        },
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command carries multiple global parameters with their values. Care should be taken when specifying more than one parameter; in some cases, only a subset of the parameters may be successfully stored in the database and other parameters may fail to be stored. In such cases the `remote-global-parameter4-get-all` command may be useful to verify the contents of the database after the update. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed; it associates the specified parameters with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 global parameter(s) successfully set.",
    "arguments": {
        "parameters": {
            <first parameter name>: <first parameter value>,
            <second parameter name>: <second parameter value>
        },
        "count": 2
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.72 remote-global-parameter6-del

This command deletes a global DHCPv6 parameter from the configuration database. The server uses the value specified in the configuration file, or a default value if the parameter is not specified in the configuration file, after deleting the parameter from the database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-global-parameter6-del command*

Command syntax:

```
{
    "command": "remote-global-parameter6-del",
    "arguments": {
        "parameters": [ <parameter name as string> ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command carries the list including exactly one name of the parameter to be deleted. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 global parameter(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.73 remote-global-parameter6-get

This command fetches the selected global parameter for the server from the specified database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-global-parameter6-get command*

Command syntax:

```
{
    "command": "remote-global-parameter6-get",
    "arguments": {
        "parameters": [ <parameter name as string> ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command carries a list including exactly one name of the parameter to be fetched. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed; it fetches the global parameter value shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 global parameter found.",
    "arguments": {
        "parameters": {
            <parameter name>: <parameter value>,
            "metadata": {
                "server-tags": [ <server tag> ]
            }
        },
        "count": 1
    }
}
```

The returned response contains a map with a global parameter name/value pair. The value may be a JSON string, integer, real, or boolean. The metadata is included and provides database-specific information associated with the returned object. If the "all" server tag is specified, the command attempts to fetch the global parameter value associated with all servers. If the explicit server tag is specified, the command fetches the value associated with the given server. If the server-specific value does not exist, the `remote-global-parameter6-get` fetches the value associated with all servers.

23.74 remote-global-parameter6-get-all

This command fetches all global parameters for the server from the specified database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-global-parameter6-get-all command*

Command syntax:

```
{
    "command": "remote-global-parameter6-get-all",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The special server tag "all" is allowed; it fetches the global parameters shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 global parameters found.",
    "arguments": {
        "parameters": [
            {
                <first parameter name>: <first parameter value>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            },
            {
                <second parameter name>: <second parameter value>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains a list of maps. Each map contains a global parameter name/value pair. The value may be a JSON string, integer, real, or boolean. The metadata is appended to each parameter and provides database-specific information associated with the returned objects. If the server tag “all” is included in the command, the response contains the global parameters shared among all servers. It excludes server-specific global parameters. If an explicit server tag is included in the command, the response contains all global parameters directly associated with the given server, and the global parameters associated with all servers when server-specific values are not present.

23.75 remote-global-parameter6-set

This command creates or updates one or more global parameters in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-global-parameter6-set command*

Command syntax:

```
{
    "command": "remote-global-parameter6-set",
    "arguments": {
        "parameters": {
            <first parameter name>: <first parameter value>,
            <second parameter name>: <second parameter value>
        },
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command carries multiple global parameters with their values. Care should be taken when specifying more than one parameter; in some cases, only a subset of the parameters may be successfully stored in the database and other parameters may fail to be stored. In such cases the `remote-global-parameter6-get-all` command may be useful to verify the contents of the database after the update. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed; it associates the specified parameters with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 global parameter(s) successfully set.",
    "arguments": {
        "parameters": {
            <first parameter name>: <first parameter value>,
            <second parameter name>: <second parameter value>
        },
        "count": 2
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.76 remote-network4-del

This command deletes an IPv4 shared network from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-network4-del command*

Command syntax:

```
{
    "command": "remote-network4-del",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>
            }
        ],
        "subnets-action": <'keep' | 'delete'>,
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one name of the shared network to be deleted. The `subnets-action` parameter denotes whether the subnets in this shared network should be deleted. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "1 IPv4 shared network(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.77 remote-network4-get

This command fetches the selected IPv4 shared network for the server from the specified database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-network4-get command*

Command syntax:

```
{
    "command": "remote-network4-get",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>
            }
        ],
        "subnets-include": <'full' | 'no'>,
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one name of the shared network to be returned. The `subnets-include` optional parameter allows for specifying whether the subnets belonging to the shared network should also be returned. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 shared network found.",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>,
                "metadata": {
                    "server-tags": [ <first server tag>, <second server tag>, ... ]
                },
                <the rest of the shared network information, potentially including␣
↪subnets>
            }
        ],
        "count": 1
    }
}
```

If the subnets are returned with the shared network, they are carried in the `subnet4` list within the shared network definition. The metadata is included in the returned shared network definition and provides the database-specific information associated with the returned object.

23.78 remote-network4-list

This command fetches a list of all IPv4 shared networks from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-network4-list command*

Command syntax:

```
{
    "command": "remote-network4-list",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <first server tag>, <second server tag>, ... ]
    }
}
```

The `server-tags` list is required for this command, and must not be empty. It may either contain one or multiple server tags as strings, or a single `null` value.

Response syntax:

```
{
    "result": 0,
    "text": "2 IPv4 shared network(s) found.",
    "arguments": {
        "shared-networks": [
            {
                "name": <first shared network name>,
                "metadata": {
                    "server-tags": [ <first server tag>, <second server tag>, ... ]
                }
            },
            {
                "name": <second shared network name>,
                "metadata": {
                    "server-tags": [ <first server tag>, ... ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains the list of maps. Each map contains the shared network name and the metadata, which provides database-specific information associated with the shared network. The returned list does not contain full definitions of the shared networks; use `remote-network4-get` to fetch the full information about the selected shared networks. If the command includes explicit server tags as strings (including the special server tag "all"), the list contains all shared networks which are associated with any of the specified tags. A network is returned even if it is associated with multiple servers and only one of the specified tags matches. If the command includes the `null` value in the `server-tags` list, the response contains all shared networks which are assigned to no servers (unassigned).

23.79 remote-network4-set

This command creates or replaces an IPv4 shared network in the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-network4-set command*

Command syntax:

```
{
    "command": "remote-network4-set",
    "arguments": {
        "shared-networks": [
            {
                <shared network specification excluding subnets list>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <first server tag>, <second server tag>, ... ]
    }
}
```

The provided list must contain exactly one shared network specification, and must not contain subnets (the "subnet4" parameter). The subnets are added to the shared network using the `remote-subnet4-set` command. The `server-tags` list is mandatory and must contain one or more server tags as strings to explicitly associate the shared network with one or more user-defined servers. It may include the special server tag "all" to associate the network with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 shared network successfully set."
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.80 remote-network6-del

This command deletes an IPv6 shared network from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-network6-del command*

Command syntax:

```
{
    "command": "remote-network6-del",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>
            }
        ],
        "subnets-action": <'keep' | 'delete'>,
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one name of the shared network to be deleted. The `subnets-action` parameter indicates whether the subnets in this shared network should be deleted. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "1 IPv6 shared network(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.81 remote-network6-get

This command fetches the selected IPv6 shared network for the server from the specified database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-network6-get command*

Command syntax:

```
{
    "command": "remote-network6-get",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>
            }
        ],
        "subnets-include": <'full' | 'no'>,
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one name of the shared network to be returned. The `subnets-include` optional parameter allows for specifying whether the subnets belonging to the shared network should also be returned. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 shared network found.",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>,
                "metadata": {
                    "server-tags": [ <first server tag>, <second server tag>, ... ]
                },
                <the rest of the shared network information, potentially including␣
↪subnets>
            }
        ],
        "count": 1
    }
}
```

If the subnets are returned with the shared network, they are carried in the `subnet6` list within the shared network definition. The metadata is included in the returned shared network definition and provides the database-specific information associated with the returned object.

23.82 remote-network6-list

This command fetches a list of all IPv6 shared networks from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-network6-list command*

Command syntax:

```
{
    "command": "remote-network6-list",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <first server tag>, <second server tag>, ... ]
    }
}
```

The `server-tags` list is required for this command, and must not be empty. It may either contain one or multiple server tags as strings, or a single `null` value.

Response syntax:

```
{
    "result": 0,
    "text": "2 IPv6 shared network(s) found.",
    "arguments": {
        "shared-networks": [
            {
                "name": <first shared network name>,
                "metadata": {
                    "server-tags": [ <first server tag>, <second server tag>, ... ]
                }
            },
            {
                "name": <second shared network name>,
                "metadata": {
                    "server-tags": [ <first server tag>, ... ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains the list of maps. Each map contains the shared network name and the metadata, which provides database-specific information associated with the shared network. The returned list does not contain full definitions of the shared networks; use `remote-network6-get` to fetch the full information about the selected shared networks. If the command includes explicit server tags as strings (including the special server tag "all"), the list contains all shared networks which are associated with any of the specified tags. A network is returned even if it is associated with multiple servers and only one of the specified tags matches. If the command includes the `null` value in the `server-tags` list, the response contains all shared networks which are assigned to no servers (unassigned).

23.83 remote-network6-set

This command creates or replaces an IPv6 shared network in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-network6-set command*

Command syntax:

```
{
    "command": "remote-network6-set",
    "arguments": {
        "shared-networks": [
            {
                <shared network specification excluding subnets list>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <first server tag>, <second server tag>, ... ]
    }
}
```

The provided list must contain exactly one shared network specification, and must not contain subnets (the "subnet6" parameter). The subnets are added to the shared network using the `remote-subnet6-set` command. The `server-tags` list is mandatory and must contain one or more server tags as strings to explicitly associate the shared network with one or more user-defined servers. It may include the special server tag "all" to associate the network with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 shared network successfully set."
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.84 remote-option-def4-del

This command deletes a DHCPv4 option definition from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option-def4-del command*

Command syntax:

```
{
    "command": "remote-option-def4-del",
    "arguments": {
        "option-defs": [ {
            "code": <option code>,
            "space": <option space>
        } ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command includes a list with exactly one option definition specification, comprising an option name and code. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv4 option definition(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.85 remote-option-def4-get

This command fetches a DHCPv4 option definition from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-option-def4-get command*

Command syntax:

```
{
    "command": "remote-option-def4-get",
    "arguments": {
        "option-defs": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The desired option definition is identified by the pair of option code/space values. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed, to fetch the option definition instance shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 option definition found.",
    "arguments": {
        "option-defs": [
            {
                <option definition>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ],
        "count": 1
    }
}
```

The metadata is included and provides database-specific information associated with the returned object. If the "all" server tag is specified, the command attempts to fetch the option definition associated with all servers. If the explicit server tag is specified, the command fetches the option definition associated with the given server. If the server-specific option definition does not exist, the `remote-option-def4-get` command fetches the option definition associated with all servers.

23.86 remote-option-def4-get-all

This command fetches all DHCPv4 option definitions from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-option-def4-get-all command*

Command syntax:

```
{
    "command": "remote-option-def4-get-all",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The special server tag "all" is allowed, to fetch the option definitions shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "2 DHCPv4 option definition(s) found.",
    "arguments": {
        "option-defs": [
            {
                <first option definition>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            },
            {
                <second option definition>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains a list of maps. Each map contains an option definition specification and the metadata, including database-specific information associated with the returned objects. If the server tag "all" is included in the command, the response contains the option definitions shared among all servers. It excludes server-specific option definitions. If an explicit server tag is included in the command, the response contains all option definitions directly associated with the given server, and the option definitions associated with all servers when server-specific option definitions are not present.

23.87 remote-option-def4-set

This command creates or replaces a DHCPv4 option definition in the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option-def4-set command*

Command syntax:

```
{
    "command": "remote-option-def4-set",
    "arguments": {
        "option-defs": [
            {
                <option definition specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The provided list must contain exactly one option definition specification. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed; it associates the specified option definition with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 option definition set."
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.88 remote-option-def6-del

This command deletes a DHCPv6 option definition from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option-def6-del command*

Command syntax:

```
{
    "command": "remote-option-def6-del",
    "arguments": {
        "option-defs": [ {
            "code": <option code>,
            "space": <option space>
        } ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command includes a list with exactly one option definition specification, comprising an option name and code. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv6 option definition(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.89 remote-option-def6-get

This command fetches a DHCPv6 option definition from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-option-def6-get command*

Command syntax:

```
{
    "command": "remote-option-def6-get",
    "arguments": {
        "option-defs": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The desired option definition is identified by the pair of option code/space values. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed, to fetch the option definition instance shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 option definition found.",
    "arguments": {
        "option-defs": [
            {
                <option definition>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ],
        "count": 1
    }
}
```

The metadata is included and provides database-specific information associated with the returned object. If the "all" server tag is specified, the command fetches the option definition associated with all servers. If the explicit server tag is specified, the command fetches the option definition associated with the given server. If the server-specific option definition does not exist, the `remote-option-def6-get` command fetches the option definition associated with all servers.

23.90 remote-option-def6-get-all

This command fetches all DHCPv6 option definitions from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-option-def6-get-all command*

Command syntax:

```
{
    "command": "remote-option-def6-get-all",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The special server tag "all" is allowed, to fetch the option definitions shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "2 DHCPv6 option definition(s) found.",
    "arguments": {
        "option-defs": [
            {
                <first option definition>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            },
            {
                <second option definition>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains a list of maps. Each map contains an option definition specification and the metadata, including database-specific information associated with the returned objects. If the server tag "all" is included in the command, the response contains the option definitions shared among all servers. It excludes server-specific option definitions. If an explicit server tag is included in the command, the response contains all option definitions directly associated with the given server, and the option definitions associated with all servers when server-specific option definitions are not present.

23.91 remote-option-def6-set

This command creates or replaces a DHCPv6 option definition in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option-def6-set command*

Command syntax:

```
{
    "command": "remote-option-def6-set",
    "arguments": {
        "option-defs": [
            {
                <option definition specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The provided list must contain exactly one option definition specification. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed; it associates the specified option definition with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 option definition set."
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.92 remote-option4-global-del

This command deletes a DHCPv4 global option from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-global-del command*

Command syntax:

```
{
    "command": "remote-option4-global-del",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command includes a list with exactly one option specification, comprising an option name and code. Specifying an empty list, a value of `null`, or multiple server tags will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv4 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.93 remote-option4-global-get

This command fetches a global DHCPv4 option for the server from the specified database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-global-get command*

Command syntax:

```
{
    "command": "remote-option4-global-get",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The option is identified by the pair of option code/space values. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag “all” is allowed, to fetch the global option instance shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 option is found.",
    "arguments": {
        "options": [
            {
                <option information>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ]
    }
}
```

The metadata is included and provides database specific information associated with the returned object. If the "all" server tag is specified, the command fetches the global option associated with all servers. If the explicit server tag is specified, the command fetches the global option associated with the given server. If the server specific option does not exist, it fetches the option associated with all servers.

23.94 remote-option4-global-get-all

This command fetches all DHCPv4 global options for the server from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-global-get-all command*

Command syntax:

```
{
    "command": "remote-option4-global-get-all",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The special server tag "all" is allowed, to fetch the global options shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "2 DHCPv4 option(s) found.",
    "arguments": {
        "options": [
            {
                <first option specification>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            },
            {
                <second option specification>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains a list of maps. Each map contains a global option specification and the metadata, including database-specific information associated with the returned object. If the server tag "all" is included in the command, the response contains the global options shared among all servers. It excludes server-specific global options. If an explicit server tag is included in the command, the response contains all global options directly associated with the given server, and the options associated with all servers when server-specific options are not present.

23.95 remote-option4-global-set

This command creates or replaces a DHCPv4 global option in the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-global-set command*

Command syntax:

```
{
    "command": "remote-option4-global-set",
    "arguments": {
        "options": [
            {
                <global option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The provided list must contain exactly one option specification. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed; it associates the specified option with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 option set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.96 remote-option4-network-del

This command deletes a DHCPv4 option from a shared network from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-network-del command*

Command syntax:

```
{
    "command": "remote-option4-network-del",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>
            }
        ],
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly one name of the shared network and exactly one option specification, comprising an option name and code. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv4 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.97 remote-option4-network-set

This command creates or replaces a DHCPv4 option in a shared network in the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-network-set command*

Command syntax:

```
{
    "command": "remote-option4-network-set",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>
            }
        ],
        "options": [
            {
                <shared network option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

The provided lists must contain exactly one name of the shared network and one option specification. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 option successfully set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.98 remote-option4-pool-del

This command deletes a DHCPv4 option from an address pool from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-pool-del command*

Command syntax:

```
{
    "command": "remote-option4-pool-del",
    "arguments": {
        "pools": [
            {
                "pool": <pool range or prefix>
            }
        ],
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly one address pool specification and exactly one option specification comprising an option space name and code. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv4 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.99 remote-option4-pool-set

This command creates or replaces a DHCPv4 option in an address pool in the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-pool-set command*

Command syntax:

```
{
    "command": "remote-option4-pool-set",
    "arguments": {
        "pools": [
            {
                "pool": <pool range or prefix>
            }
        ],
        "options": [
            {
                <address pool option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly address pool specification and exactly one option specification. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 option successfully set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.100 remote-option4-subnet-del

This command deletes a DHCPv4 option from a subnet from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-subnet-del command*

Command syntax:

```
{
    "command": "remote-option4-subnet-del",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>
            }
        ],
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly one ID of the subnet and exactly one option specification, comprising an option name and code. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv4 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.101 remote-option4-subnet-set

This command creates or replaces a DHCPv4 option in a subnet in the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option4-subnet-set command*

Command syntax:

```
{
    "command": "remote-option4-subnet-set",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>
            }
        ],
        "options": [
            {
                <subnet option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

The provided lists must contain exactly one ID of the subnet and one option specification. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 option successfully set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.102 remote-option6-global-del

This command deletes a DHCPv6 global option from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-global-del command*

Command syntax:

```
{
    "command": "remote-option6-global-del",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

This command includes a list with exactly one option specification, comprising an option name and code. Specifying an empty list, a value of `null`, or multiple server tags will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv6 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.103 remote-option6-global-get

This command fetches a global DHCPv6 option for the server from the specified database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-global-get command*

Command syntax:

```
{
    "command": "remote-option6-global-get",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The option is identified by the pair of option code/space values. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed, to fetch the global option instance shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 option is found.",
    "arguments": {
        "options": [
            {
                <option information>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ]
    }
}
```

The metadata is included and provides database-specific information associated with the returned object. If the "all" server tag is specified, the command attempts to fetch the global option associated with all servers. If the explicit server tag is specified, the command will fetch the global option associated with the given server. If the server-specific option does not exist, it fetches the option associated with all servers.

23.104 remote-option6-global-get-all

This command fetches all DHCPv6 global options for the server from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-global-get-all command*

Command syntax:

```
{
    "command": "remote-option6-global-get-all",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The special server tag "all" is allowed, to fetch the global options shared by all servers.

Response syntax:

```
{
    "result": 0,
    "text": "2 DHCPv6 option(s) found.",
    "arguments": {
        "options": [
            {
                <first option specification>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            },
            {
                <second option specification>,
                "metadata": {
                    "server-tags": [ <server tag> ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains a list of maps. Each map contains a global option specification and the metadata, including database-specific information associated with the returned object. If the server tag "all" is included in the command, the response contains the global options shared between all servers. It excludes server-specific global options. If an explicit server tag is included in the command, the response contains all global options directly associated with the given server, and the options associated with all servers when server-specific options are not present.

23.105 remote-option6-global-set

This command creates or replaces a DHCPv6 global option in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-global-set command*

Command syntax:

```
{
    "command": "remote-option6-global-set",
    "arguments": {
        "options": [
            {
                <global option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <single server tag as string> ]
    }
}
```

The provided list must contain exactly one option specification. The `server-tags` list is mandatory and must contain exactly one server tag. Specifying an empty list, a value of `null`, or multiple server tags will result in an error. The server tag "all" is allowed; it associates the specified option with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 option set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.106 remote-option6-network-del

This command deletes a DHCPv6 option from a shared network from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-network-del command*

Command syntax:

```
{
    "command": "remote-option6-network-del",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>
            }
        ],
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly one name of the shared network and exactly one option specification, comprising an option name and code. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv6 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.107 remote-option6-network-set

This command creates or replaces a DHCPv6 option in a shared network in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-network-set command*

Command syntax:

```
{
    "command": "remote-option6-network-set",
    "arguments": {
        "shared-networks": [
            {
                "name": <shared network name>
            }
        ],
        "options": [
            {
                <shared network option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

The provided lists must contain exactly one name of the shared network and one option specification. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 option successfully set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.108 remote-option6-pd-pool-del

This command deletes a DHCPv6 option from a prefix delegation pool from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-pd-pool-del command*

Command syntax:

```
{
    "command": "remote-option6-pd-pool-del",
    "arguments": {
        "pd-pools": [
            {
                "prefix": <pool prefix (address part)>
                "prefix-len": <pool prefix (length part)>
            }
        ],
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly one prefix delegation pool specification and exactly one option specification, comprising an option name and code. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv6 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.109 remote-option6-pd-pool-set

This command creates or replaces a DHCPv6 option in a prefix delegation pool in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-pd-pool-set command*

Command syntax:

```
{
    "command": "remote-option6-pd-pool-set",
    "arguments": {
        "pd-pools": [
            {
                "prefix": <pool prefix (address part)>
                "prefix-len": <pool prefix (length part)>
            }
        ],
        "options": [
            {
                <prefix delegation pool option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly one prefix delegation pool specification and exactly one option specification. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 option successfully set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.110 remote-option6-pool-del

This command deletes a DHCPv6 option from an address pool from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-pool-del command*

Command syntax:

```
{
    "command": "remote-option6-pool-del",
    "arguments": {
        "pools": [
            {
                "pool": <pool range or prefix>
            }
        ],
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly one address pool specification and exactly one option specification, comprising an option name and code. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv6 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success

- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.111 remote-option6-pool-set

This command creates or replaces a DHCPv6 option in an address pool in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-pool-set command*

Command syntax:

```
{
    "command": "remote-option6-pool-set",
    "arguments": {
        "pools": [
            {
                "pool": <pool range or prefix>
            }
        ],
        "options": [
            {
                <address pool option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly address pool specification and exactly one option specification. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 option successfully set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success

- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.112 remote-option6-subnet-del

This command deletes a DHCPv6 option from a subnet from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-subnet-del command*

Command syntax:

```
{
    "command": "remote-option6-subnet-del",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>
            }
        ],
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes two lists with exactly one ID of the subnet and exactly one option specification, comprising an option name and code. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv6 option(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported

- 3 - empty (command was completed successfully, but no data was affected or returned)

23.113 remote-option6-subnet-set

This command creates or replaces a DHCPv6 option in a subnet in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-option6-subnet-set command*

Command syntax:

```
{
    "command": "remote-option6-subnet-set",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>
            }
        ],
        "options": [
            {
                <subnet option specification>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

The provided lists must contain exactly one ID of the subnet and one option specification. Specifying an empty list, a value of `null`, or a server tag will result in an error.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 option successfully set.",
    "arguments": {
        "options": [
            {
                "code": <option code>,
                "space": <option space>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported

- 3 - empty (command was completed successfully, but no data was affected or returned)

23.114 remote-server4-del

This command deletes information about a DHCPv4 server from the configuration database. Any configuration explicitly associated with the deleted server is automatically disassociated. In addition, configuration elements not shareable with other servers (e.g. global DHCP parameters) are deleted. Shareable configuration elements (e.g. subnets, shared networks) are not deleted as they may be used by other servers.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-server4-del command*

Command syntax:

```
{
    "command": "remote-server4-del",
    "arguments": {
        "servers": [
            {
                "server-tag": <server name>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command carries the list including exactly one map with the tag of the server to be deleted.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv4 server(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.115 remote-server4-get

This command fetches information about the DHCPv4 server, such as the server tag and description.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-server4-get command*

Command syntax:

```
{
    "command": "remote-server4-get",
    "arguments": {
        "servers": [
            {
                "server-tag": <server tag>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command carries the list including exactly one map with the tag of the server to be fetched.

Response syntax:

```
{
    "result": 0,
    "text": "DHCP server 'server tag' found.",
    "arguments": {
        "servers": [
            {
                "server-tag": <server tag>,
                "description": <server description>
            }
        ],
        "count": 1
    }
}
```

The server tag is the unique identifier of the server, used to associate the configuration elements in the database with the particular server instance. The returned server description is specified by the user when setting the server information.

23.116 remote-server4-get-all

This command fetches information about all DHCPv4 servers specified by the user.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-server4-get-all command*

Command syntax:

```
{
    "command": "remote-server4-get-all",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command contains no arguments besides the optional `remote`.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 servers found.",
    "arguments": {
        "servers": [
            {
                "server-tag": <first server tag>,
                "description": <first server description>
            },
            {
                "server-tag": <second server tag>,
                "description": <second server description>
            }
        ],
        "count": 2
    }
}
```

The returned response contain a list of maps. Each map contains a server tag uniquely identifying a server, and the user-defined description of the server. The Kea Configuration Backend uses the keyword `all` to associate parts of the configuration with all servers. Internally, it creates the logical server `all` for this purpose. However, this logical server is not returned as a result of the `remote-server4-get-all` command; only the user-defined servers are returned.

23.117 remote-server4-set

This command creates or replaces information about the DHCPv4 server in the database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-server4-set command*

Command syntax:

```
{
    "command": "remote-server4-set",
    "arguments": {
        "servers": [
            {
                "server-tag": <server tag>,
                "description": <server description>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

The provided list must contain exactly one server specification. The `server-tag` must be unique across all servers within the configuration database. The `description` is the arbitrary text describing the server, its location within the network, etc.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv4 server successfully set.",
    "arguments": {
        "servers": [
            {
                "server-tag": <server tag>,
                "description": <server description>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.118 remote-server6-del

This command deletes information about a DHCPv6 server from the configuration database. Any configuration explicitly associated with the deleted server is automatically disassociated. In addition, configuration elements not shareable with other servers (e.g. global DHCP parameters) are deleted. Shareable configuration elements (e.g. subnets, shared networks) are not deleted as they may be used by other servers.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-server6-del command*

Command syntax:

```
{
    "command": "remote-server6-del",
    "arguments": {
        "servers": [
            {
                "server-tag": <server name>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command carries the list including exactly one map with the tag of the server to be deleted.

Response syntax:

```
{
    "result": 0,
    "text": "1 DHCPv6 server(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.119 remote-server6-get

This command fetches information about the DHCPv6 server, such as the server tag and description.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-server6-get command*

Command syntax:

```
{
    "command": "remote-server6-get",
    "arguments": {
        "servers": [
            {
                "server-tag": <server tag>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command carries the list including exactly one map with the tag of the server to be fetched.

Response syntax:

```
{
    "result": 0,
    "text": "DHCP server 'server tag' found.",
    "arguments": {
        "servers": [
            {
                "server-tag": <server tag>,
                "description": <server description>
            }
        ],
        "count": 1
    }
}
```

The server tag is the unique identifier of the server, used to associate the configuration elements in the database with the particular server instance. The returned server description is specified by the user when setting the server information.

23.120 remote-server6-get-all

This command fetches information about all DHCPv6 servers specified by the user.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-server6-get-all command*

Command syntax:

```
{
    "command": "remote-server6-get-all",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command contains no arguments besides the optional `remote`.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 servers found.",
    "arguments": {
        "servers": [
            {
                "server-tag": <first server tag>,
                "description": <first server description>
            },
            {
                "server-tag": <second server tag>,
                "description": <second server description>
            }
        ],
        "count": 2
    }
}
```

The returned response contain a list of maps. Each map contains a server tag uniquely identifying a server, and the user-defined description of the server. The Kea Configuration Backend uses the keyword `all` to associate parts of the configuration with all servers. Internally, it creates the logical server `all` for this purpose. However, this logical server is not returned as a result of the `remote-server6-get-all` command; only the user-defined servers are returned.

23.121 remote-server6-set

This command creates or replaces information about the DHCPv6 server in the database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-server6-set command*

Command syntax:

```
{
    "command": "remote-server6-set",
    "arguments": {
        "servers": [
            {
                "server-tag": <server tag>,
                "description": <server description>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

The provided list must contain exactly one server specification. The `server-tag` must be unique across all servers within the configuration database. The `description` is the arbitrary text describing the server, its location within the network, etc.

Response syntax:

```
{
    "result": 0,
    "text": "DHCPv6 server successfully set.",
    "arguments": {
        "servers": [
            {
                "server-tag": <server tag>,
                "description": <server description>
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.122 remote-subnet4-del-by-id

This command deletes an IPv4 subnet by ID from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet4-del-by-id command*

Command syntax:

```
{
    "command": "remote-subnet4-del-by-id",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one ID of the subnet to be deleted. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "1 IPv4 subnet(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error

- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.123 remote-subnet4-del-by-prefix

This command deletes an IPv4 subnet by prefix from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet4-del-by-prefix command*

Command syntax:

```
{
    "command": "remote-subnet4-del-by-prefix",
    "arguments": {
        "subnets": [
            {
                "subnet": <subnet prefix>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one prefix of the subnet to be deleted. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "1 IPv4 subnet(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.124 remote-subnet4-get-by-id

This command fetches the selected IPv4 subnet by ID from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet4-get-by-id command*

Command syntax:

```
{
    "command": "remote-subnet4-get-by-id",
    "arguments": {
        "subnets": [ {
            "id": <subnet identifier>
        } ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one ID of the subnet to be returned. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet found.",
    "arguments": {
        "subnets": [ {
            "id": <subnet identifier>,
            "subnet": <subnet prefix>,
            "shared-network-name": <shared network name> | null,
            "metadata": {
                "server-tags": [ <first server tag>, <second server tag>, ... ]
            },
            <the rest of the subnet specification here>
        } ],
        "count": 1
    }
}
```

If the shared network name is null, it means that the returned subnet does not belong to any shared network (a global subnet). The metadata is included in the returned subnet definition and provides database-specific information associated with the returned object.

23.125 remote-subnet4-get-by-prefix

This command fetches the selected IPv4 subnet by prefix from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet4-get-by-prefix command*

Command syntax:

```
{
    "command": "remote-subnet4-get-by-prefix",
    "arguments": {
        "subnets": [ {
            "subnet": <subnet prefix>
        } ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one prefix of the subnet to be returned. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet found.",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>,
                "subnet": <subnet prefix>,
                "shared-network-name": <shared network name> | null,
                "metadata": {
                    "server-tags": [ <first server tag>, <second server tag>, ... ]
                },
                <the rest of the subnet specification here>
            }
        ],
        "count": 1
    }
}
```

If the shared network name is null, it means that the returned subnet does not belong to any shared network (global subnet). The metadata is included in the returned subnet definition and provides database-specific information associated with the returned object.

23.126 remote-subnet4-list

This command fetches a list of all IPv4 subnets from the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet4-list command*

Command syntax:

```
{
    "command": "remote-subnet4-list",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <first server tag>, <second server tag>, ... ]
    }
}
```

The `server-tags` list is required for this command, and must not be empty. It may either contain one or multiple server tags as strings, or a single `null` value.

Response syntax:

```
{
    "result": 0,
    "text": "2 IPv4 subnets found.",
    "arguments": {
        "subnets": [
            {
                "id": <first subnet identifier>,
                "subnet": <first subnet prefix>,
                "shared-network-name": <shared network name> | null,
                "metadata": {
                    "server-tags": [ <first server tag>, <second server tag>, ... ]
                }
            },
            {
                "id": <second subnet identifier>,
                "subnet": <second subnet prefix>,
                "shared-network-name": <shared network name> | null,
                "metadata": {
                    "server-tags": [ <first server tag>, ... ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains a list of maps. Each map contains a subnet identifier, prefix, and shared network name to which the subnet belongs. If the subnet does not belong to a shared network, the name is null. The metadata includes database-specific information associated with the subnets. The returned list does not contain full subnet definitions; use `remote-subnet4-get` to fetch the full information about the selected subnets. If the command includes explicit server tags as strings (including the special server tag "all"), the list contains all subnets which are associated with any of the specified tags. A subnet is returned even if it is associated with multiple servers and only one of the specified tags matches. If the command includes the `null` value in the `server-tags` list, the response contains all subnets which are assigned to no servers (unassigned).

23.127 remote-subnet4-set

This command creates or replaces an IPv4 subnet in the configuration database.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet4-set command*

Command syntax:

```
{
    "command": "remote-subnet4-set",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>,
                "subnet": <subnet prefix>,
                "shared-network-name": <shared network name> | null,
                <the rest of the subnet specification here>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <first server tag>, <second server tag>, ... ]
    }
}
```

The provided list must contain exactly one subnet specification. The `shared-network-name` parameter is required for these commands; it associates the subnet with the shared network by its name. If the subnet must not belong to any shared network (a global subnet), the `null` value must be specified for the shared network name. The `server-tags` list is mandatory and must contain one or more server tags as strings to explicitly associate the subnet with one or more user-defined servers. The `remote-subnet4-set` command may include the special server tag "all" to associate the subnet with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet successfully set.",
    "arguments": {
        "id": <subnet identifier>,
        "subnet": <subnet prefix>
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.128 remote-subnet6-del-by-id

This command deletes an IPv6 subnet by ID from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet6-del-by-id command*

Command syntax:

```
{
    "command": "remote-subnet6-del-by-id",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one ID of the subnet to be deleted. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "1 IPv6 subnet(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.129 remote-subnet6-del-by-prefix

This command deletes an IPv6 subnet by prefix from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet6-del-by-prefix command*

Command syntax:

```
{
    "command": "remote-subnet6-del-by-prefix",
    "arguments": {
        "subnets": [
            {
                "subnet": <subnet prefix>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one prefix of the subnet to be deleted. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "1 IPv6 subnet(s) deleted.",
    "arguments": {
        "count": 1
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.130 remote-subnet6-get-by-id

This command fetches the selected IPv6 subnet by ID from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet6-get-by-id command*

Command syntax:

```
{
    "command": "remote-subnet6-get-by-id",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one ID of the subnet to be returned. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 subnet found.",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>,
                "subnet": <subnet prefix>,
                "shared-network-name": <shared network name> | null,
                "metadata": {
                    "server-tags": [ <first server tag>, <second server tag>, ... ]
                },
                <the rest of the subnet specification here>
            }
        ],
        "count": 1
    }
}
```

If the shared network name is null, it means that the returned subnet does not belong to any shared network (a global subnet). The metadata is included in the returned subnet definition and provides database-specific information associated with the returned object.

23.131 remote-subnet6-get-by-prefix

This command fetches the selected IPv6 subnet by prefix from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet6-get-by-prefix command*

Command syntax:

```
{
    "command": "remote-subnet6-get-by-prefix",
    "arguments": {
        "subnets": [
            {
                "subnet": <subnet prefix>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        }
    }
}
```

This command includes a list with exactly one prefix of the subnet to be returned. The `server-tags` parameter must not be specified for this command.

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 subnet found.",
    "arguments": {
        "subnets": [ {
            "id": <subnet identifier>,
            "subnet": <subnet prefix>,
            "shared-network-name": <shared network name> | null,
            "metadata": {
                "server-tags": [ <first server tag>, <second server tag>, ... ]
            },
            <the rest of the subnet specification here>
        } ],
        "count": 1
    }
}
```

If the shared network name is null, it means that the returned subnet does not belong to any shared network (global subnet). The metadata is included in the returned subnet definition and provides database-specific information associated with the returned object.

23.132 remote-subnet6-list

This command fetches a list of all IPv6 subnets from the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet6-list command*

Command syntax:

```
{
    "command": "remote-subnet6-list",
    "arguments": {
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <first server tag>, <second server tag>, ... ]
    }
}
```

The `server-tags` list is required for this command, and must not be empty. It may either contain one or multiple server tags as strings, or a single `null` value.

Response syntax:

```
{
    "result": 0,
    "text": "2 IPv6 subnets found.",
    "arguments": {
        "subnets": [
            {
                "id": <first subnet identifier>,
                "subnet": <first subnet prefix>,
                "shared-network-name": <shared network name> | null,
                "metadata": {
                    "server-tags": [ <first server tag>, <second server tag>, ... ]
                }
            },
            {
                "id": <second subnet identifier>,
                "subnet": <second subnet prefix>,
                "shared-network-name": <shared network name> | null,
                "metadata": {
                    "server-tags": [ <first server tag>, ... ]
                }
            }
        ],
        "count": 2
    }
}
```

The returned response contains a list of maps. Each map contains a subnet identifier, prefix, and shared network name to which the subnet belongs. If the subnet does not belong to a shared network, the name is null. The metadata includes database-specific information associated with the subnets. The returned list does not contain full subnet definitions; use `remote-subnet6-get` to fetch the full information about the selected subnets. If the command includes explicit server tags as strings (including the special server tag "all"), the list contains all subnets which are associated with any of the specified tags. A subnet is returned even if it is associated with multiple servers and only one of the specified tags matches. If the command includes the `null` value in the `server-tags` list, the response contains all subnets which are assigned to no servers (unassigned).

23.133 remote-subnet6-set

This command creates or replaces an IPv6 subnet in the configuration database.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*cb_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *remote-subnet6-set command*

Command syntax:

```
{
    "command": "remote-subnet6-set",
    "arguments": {
        "subnets": [
            {
                "id": <subnet identifier>,
                "subnet": <subnet prefix>,
                "shared-network-name": <shared network name> | null,
                <the rest of the subnet specification here>
            }
        ],
        "remote": {
            <specification of the database to connect to>
        },
        "server-tags": [ <first server tag>, <second server tag>, ... ]
    }
}
```

The provided list must contain exactly one subnet specification. The `shared-network-name` parameter is required for these commands; it associates the subnet with the shared network by its name. If the subnet must not belong to any shared network (a global subnet), the `null` value must be specified for the shared network name. The `server-tags` list is mandatory and must contain one or more server tags as strings to explicitly associate the subnet with one or more user-defined servers. The `remote-subnet6-set` command may include the special server tag "all" to associate the subnet with all servers.

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 subnet successfully set.",
    "arguments": {
        "id": <subnet identifier>,
        "subnet": <subnet prefix>
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.134 reservation-add

This command adds a new host reservation. The reservation may include IPv4 addresses, IPv6 addresses, IPv6 prefixes, various identifiers, a class the client will be assigned to, DHCPv4 and DHCPv6 options, and more.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (*host_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *reservation-add command*

Command syntax:

```
{
    "command": "reservation-add",
    "arguments": {
        "reservation": {
            "boot-file-name": <string>,
            "comment": <string>,
            "client-id": <string>,
            "circuit-id": <string>,
            "duid": <string>,
            "flex-id": <string>,
            "ip-address": <string (IPv4 address)>,
            "ip-addresses": [ <comma-separated strings> ],
            "hw-address": <string>,
            "hostname": <string>,
            "next-server": <string (IPv4 address)>,
            "option-data-list": [ <comma-separated structures defining options> ],
            "prefixes": [ <comma-separated IPv6 prefixes> ],
            "client-classes": [ <comma-separated strings> ],
            "server-hostname": <string>,
            "subnet-id": <integer>,
            "user-context": <any valid JSON>
        }
    }
}
```

Note that ip-address, client-id, next-server, server-hostname, and boot-file-name are IPv4-specific. duid, ip-addresses, and prefixes are IPv6-specific.

Response syntax:

```
{
    "result": <integer>,
    "text": <string>
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.135 reservation-del

This command deletes an existing host reservation.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (*host_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *reservation-del command*

Command syntax:

```
{
    "command": "reservation-del",
    "arguments": {
        "subnet-id": <integer>,
        "ip-address": <string>,
        "identifier-type": <one of 'hw-address', 'duid', 'circuit-id', 'client-id'␣
↪and 'flex-id'>,
        "identifier": <string>
    }
}
```

The host reservation can be identified by either the (subnet-id, ip-address) pair or a triplet of (subnet-id, identifier-type, identifier).

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.136 reservation-get

This command retrieves an existing host reservation.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (*host_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *reservation-get command*

Command syntax:

```
{
    "command": "reservation-get",
    "arguments": {
        "subnet-id": <integer>,
        "identifier-type": <one of 'hw-address', 'duid', 'circuit-id', 'client-id'␣
↪and 'flex-id'>,
        "identifier": <string>
    }
}
```

The host reservation can be identified by either the (subnet-id, ip-address) pair or a triplet of (subnet-id, identifier-type, identifier).

Response syntax:

```
{
    "result": <integer>,
    "text": <string>,
    "arguments": {
        "boot-file-name": <string>,
        "comment": <string>,
        "client-id": <string>,
        "circuit-id": <string>,
        "duid": <string>,
        "flex-id": <string>,
        "ip-address": <string (IPv4 address)>,
        "ip-addresses": [ <comma-separated strings> ],
        "hw-address": <string>,
        "hostname": <string>,
        "next-server": <string (IPv4 address)>,
        "option-data-list": [ <comma-separated structures defining options> ],
        "prefixes": [ <comma-separated IPv6 prefixes> ],
        "reservation-client-classes": [ <comma-separated strings> ],
        "server-hostname": <string>,
        "subnet-id": <integer>,
        "user-context": <any valid JSON>
    }
}
```

The arguments object appears only if a host is found. Many fields in the arguments object appear only if a specific field is set.

23.137 reservation-get-all

This command retrieves all host reservations for a specified subnet.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (*host_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *reservation-get-all command*

Command syntax:

```
{
    "command": "reservation-get-all",
    "arguments": {
        "subnet-id": <integer>
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

The reservation-get-all command may result in very large responses.

23.138 reservation-get-by-hostname

This command retrieves all host reservations for a specified hostname and optionally a specified subnet.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.7.1 (*host_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *reservation-get-by-hostname command*

Command syntax:

```
{
    "command": "reservation-get-by-hostname",
    "arguments": {
        "hostname": <hostname>,
        "subnet-id": <integer>
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

The reservation-get-by-hostname command may result in large responses.

23.139 reservation-get-by-id

This command retrieves all host reservations for a specified identifier (type and value).

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.9.0 (*host_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *reservation-get-by-id command*

Command syntax:

```
{
    "command": "reservation-get-by-id",
    "arguments": {
        "identifier-type": <one of 'hw-address', 'duid', 'circuit-id', 'client-id'␣
↪and 'flex-id'>,
        "identifier": <string>
    }
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

The reservation-get-by-id command may result in large responses.

23.140 reservation-get-page

This command retrieves all host reservations or host reservations for a specified subnet by page.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (*host_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *reservation-get-page command*

Command syntax:

```
{
    "command": "reservation-get-page",
    "arguments": {
        "subnet-id": <integer>,
        "limit": <integer>,
        "source-index": <integer>,
        "from": <integer>
    }
}
```

The page size limit is mandatory. The subnet-id is optional since version 1.9.0. The source-index and from host-id are optional and default to 0. Values to use to load the next page are returned in responses in a next map.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.141 server-tag-get

This command returns the server tag used by the server. Server tag is essential configuration parameter in the Config Backend configuration. This parameter is configured in the local config file. This command does not take any parameters.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (built-in)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *server-tag-get command*

Command syntax:

```
{
    "command": "server-tag-get"
}
```

Response syntax:

```
{
    "result": 0,
    "arguments": {
        "server-tag": "office1"
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.142 shutdown

This command instructs the server to initiate its shutdown procedure.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *shutdown command*

Command syntax:

```
{
    "command": "shutdown"
    "arguments": {
        "exit-value": 123
    }
}
```

The server responds with a confirmation that the shutdown procedure has been initiated.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.143 stat-lease4-get

This command fetches lease statistics for a range of known IPv4 subnets.

Supported by: *kea-dhcp4*

Availability: 1.4.0 (*stat_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *stat-lease4-get command*

Command syntax:

```
{
  "command": "stat-lease4-get"
}
```

Response syntax:

```
{
    "result": 0,
    "text": "stat-lease4-get: 2 rows found",
    "arguments": {
      "result-set": {
        "columns": [ "subnet-id",
                     "total-addresses",
                     "cumulative-assigned-addresses",
                     "assigned-addresses",
                     "declined-addresses" ],
        "rows": [
          [ 10, 256, 200, 111, 0 ],
          [ 20, 4098, 5000, 2034, 4 ]
        ],
      "timestamp": "2018-05-04 15:03:37.000000"
      }
    }
  }
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.144 stat-lease6-get

This command fetches lease statistics for a range of known IPv6 subnets.

Supported by: *kea-dhcp6*

Availability: 1.4.0 (*stat_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *stat-lease6-get command*

Command syntax:

```
{
  "command": "stat-lease6-get",
  "arguments": {
    "subnet-id" : 10
  }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "stat-lease6-get: 2 rows found",
    "arguments": {
      "result-set": {
        "columns": [ "subnet-id", "total-nas", "cumulative-assigned-nas", "assigned-
↪nas", "declined-nas", "total-pds",  "cumulative-assigned-pds", "assigned-pds" ],
        "rows": [
          [ 10, 4096, 3000, 2400, 3, 0, 0],
          [ 20, 0, 0, 0, 1048, 500, 233 ],
          [ 30, 256, 300, 60, 0, 1048, 15, 15 ]
        ],
      "timestamp": "2018-05-04 15:03:37.000000"
      }
    }
  }
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.145 statistic-get

This command retrieves a single statistic. It takes a single string parameter called name that specifies the statistic name.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *statistic-get command*

Command syntax:

```
{
    "command": "statistic-get",
    "arguments": {
        "name": "pkt4-received"
    }
}
```

The server responds with the details of the requested statistic, with a result of 0 indicating success, and the specified statistic as the value of the "arguments" parameter.

Response syntax:

```
{
   "result": 0,
   "arguments": {
       "pkt4-received": [ [ "first_value", "2019-07-30 10:11:19.498739" ], [ "second_
↪value", "2019-07-30 10:11:19.498662" ] ]
       }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.146 statistic-get-all

This command retrieves all recorded statistics.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *statistic-get-all command*

Command syntax:

```
{
    "command": "statistic-get-all",
    "arguments": { }
}
```

The server responds with the details of all recorded statistics, with a result of 0 indicating that it iterated over all statistics (even when the total number of statistics is zero).

Response syntax:

```
{
   "result": 0,
   "arguments": {
       "cumulative-assigned-addresses": [ [ 0, "2019-07-30 10:04:28.386740" ] ],
       "declined-addresses": [ [ 0, "2019-07-30 10:04:28.386733" ] ],
       "reclaimed-declined-addresses": [ [ 0, "2019-07-30 10:04:28.386735" ] ],
       "reclaimed-leases": [ [ 0, "2019-07-30 10:04:28.386736" ] ],
       "subnet[1].assigned-addresses": [ [ 0, "2019-07-30 10:04:28.386740" ] ],
       "subnet[1].cumulative-assigned-addresses": [ [ 0, "2019-07-30 10:04:28.386740"␣
↪] ],
       "subnet[1].declined-addresses": [ [ 0, "2019-07-30 10:04:28.386743" ] ],
       "subnet[1].reclaimed-declined-addresses": [ [ 0, "2019-07-30 10:04:28.386745"␣
↪] ],
        "subnet[1].reclaimed-leases": [ [ 0, "2019-07-30 10:04:28.386747" ] ],
        "subnet[1].total-addresses": [ [ 200, "2019-07-30 10:04:28.386719" ] ]
        }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.147 statistic-remove

This command deletes a single statistic. It takes a single string parameter called name that specifies the statistic name.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *statistic-remove command*

Command syntax:

```
{
    "command": "statistic-remove",
    "arguments": {
        "name": "pkt4-received"
    }
}
```

If the specific statistic is found and its removal is successful, the server responds with a status of 0, indicating success, and an empty parameters field. If an error is encountered (e.g. the requested statistic was not found), the server returns a status code of 1 (error) and the text field contains the error description.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.148 statistic-remove-all

(Deprecated) This command deletes all statistics.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *statistic-remove-all command*

Command syntax:

```
{
    "command": "statistic-remove-all",
    "arguments": { }
}
```

If the removal of all statistics is successful, the server responds with a status of 0, indicating success, and an empty parameters field. If an error is encountered, the server returns a status code of 1 (error) and the text field contains the error description.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.149 statistic-reset

This command sets the specified statistic to its neutral value: 0 for integer, 0.0 for float, 0h0m0s0us for time duration, and “” for string type. It takes a single string parameter called name that specifies the statistic name.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *statistic-reset command*

Command syntax:

```
{
    "command": "statistic-reset",
    "arguments": {
        "name": "pkt4-received"
    }
}
```

If the specific statistic is found and the reset is successful, the server responds with a status of 0, indicating success, and an empty parameters field. If an error is encountered (e.g. the requested statistic was not found), the server returns a status code of 1 (error) and the text field contains the error description.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.150 statistic-reset-all

This command sets all statistics to their neutral values: 0 for integer, 0.0 for float, 0h0m0s0us for time duration, and “” for string type.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.0.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *statistic-reset-all command*

Command syntax:

```
{
    "command": "statistic-reset-all",
    "arguments": { }
}
```

If the operation is successful, the server responds with a status of 0, indicating success, and an empty parameters field. If an error is encountered, the server returns a status code of 1 (error) and the text field contains the error description.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.151 statistic-sample-age-set

This command sets a time-based limit for a single statistic. It takes two parameters: a string called name and an integer value called duration.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *statistic-sample-age-set command*

Command syntax:

```
{
    "command": "statistic-sample-age-set",
    "arguments": {
        "name": "pkt4-received",
        "duration": 1245
    }
}
```

The server responds with a message about a successfully set limit for the given statistic, with a result of 0 indicating success, and an empty parameters field. If an error is encountered (e.g. the requested statistic was not found), the server returns a status code of 1 (error) and the text field contains the error description.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.152 statistic-sample-age-set-all

This command sets a time-based limit for all statistics. It takes a single integer parameter called duration.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *statistic-sample-age-set-all command*

Command syntax:

```
{
    "command": "statistic-sample-age-set-all",
    "arguments": {
        "duration": 1245
    }
}
```

The server responds with a message about successfully set limits for all statistics, with a result of 0 indicating success, and an empty parameters field. If an error is encountered, the server returns a status code of 1 (error) and the text field contains the error description.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.153 statistic-sample-count-set

This command sets a size-based limit for a single statistic. It takes two parameters: a string called name and an integer value called max-samples.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *statistic-sample-count-set command*

Command syntax:

```
{
    "command": "statistic-sample-count-set",
    "arguments": {
        "name": "pkt4-received",
        "max-samples": 100
    }
}
```

The server responds with a message about a successfully set limit for the given statistic, with a result of 0 indicating success, and an empty parameters field. If an error is encountered (e.g. the requested statistic was not found), the server returns a status code of 1 (error) and the text field contains the error description.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.154 statistic-sample-count-set-all

This command sets a size-based limit for all statistics. It takes a single integer parameter called max-samples.

Supported by: *kea-dhcp4*, *kea-dhcp6*

Availability: 1.6.0 (built-in)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *statistic-sample-count-set-all command*

Command syntax:

```
{
    "command": "statistic-sample-count-set-all",
    "arguments": {
        "max-samples": 100
    }
}
```

The server responds with a message about successfully set limits for all statistics, with a result of 0 indicating success, and an empty parameters field. If an error is encountered, the server returns a status code of 1 (error) and the text field contains the error description.

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.155 status-get

This command returns server's runtime information. It takes no arguments.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.7.3 (built-in)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *status-get command*

Command syntax:

```
{
    "command": "status-get"
}
```

Response syntax:

```
{
    "result": <integer>,
    "arguments": {
        "pid": <integer>,
        "uptime": <uptime in seconds>,
        "reload": <time since reload in seconds>,
        "high-availability": [
            {
                "ha-mode": <HA mode configured for this relationship>
                "ha-servers": {
                    "local": {
                        "role": <role of this server as in the configuration file>,
                        "scopes": <list of scope names served by this server>,
                        "state": <HA state name of the server receiving the command>,
                    },
                    "remote": {
                        "age": <the age of the remote status in seconds>,
                        "in-touch": <indicates if this server communicated with␣
↪remote>,
                        "last-scopes": <list of scopes served by partner>,
                        "last-state": <HA state name of the partner>,
                        "role": <partner role>
                    }
                }
            }
        ],
        "multi-threading-enabled": true,
        "thread-pool-size": 4,
        "packet-queue-size": 64
    }
}
```

If the libdhcp_ha (High Availability) hooks library is loaded by the DHCP server receiving this command the response also includes the HA specific status information of the server receiving the command and its partner's status.

23.156 subnet4-add

This command creates and adds a new subnet to the existing server configuration. This operation has no impact on other subnets.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *subnet4-add command*

Command syntax:

```
{
    "command": "subnet4-add",
    "arguments": {
        "subnets": [ {
            "id": 123,
            "subnet": "10.20.30.0/24",
            ...
        } ]
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet added",
    "arguments": {
        "subnets": [
            {
                "id": 123,
                "subnet": "10.20.30.0/24"
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.157 subnet4-del

This command removes a subnet from the server's configuration. This command has no effect on other configured subnets, but removing a subnet has certain implications which the server's administrator should be aware of.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *subnet4-del command*

Command syntax:

```
{
    "command": "subnet4-del",
    "arguments": {
        "id": 123
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet 192.0.2.0/24 (id 123) deleted",
    "arguments": {
        "subnets": [
            {
                "id": 123,
                "subnet": "192.0.2.0/24"
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.158 subnet4-get

This command retrieves detailed information about the specified subnet. This command usually follows `subnet4-list`, which discovers available subnets with their respective subnet identifiers and prefixes.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *subnet4-get command*

Command syntax:

```
{
    "command": "subnet4-get",
    "arguments": {
        "id": 10
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "Info about IPv4 subnet 10.0.0.0/8 (id 10) returned",
    "arguments": {
        "subnets": [
            {
                "subnet": "10.0.0.0/8",
                "id": 1,
                "option-data": [
                    ...
                ],
                ...
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.159 subnet4-list

This command lists all currently configured subnets. The subnets are returned in a brief format, i.e. a subnet identifier and subnet prefix are included for each subnet.

Supported by: *kea-dhcp4*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *subnet4-list command*

Command syntax:

```
{
    "command": "subnet4-list"
}
```

Response syntax:

```
{
    "result": 0,
    "text": "2 IPv4 subnets found",
    "arguments": {
        "subnets": [
            {
                "id": 10,
                "subnet": "10.0.0.0/8"
            },
            {
                "id": 100,
                "subnet": "192.0.2.0/24"
            }
        ]
    }
}
```

If no IPv4 subnets are found, an error code is returned along with the error description.

23.160 subnet4-update

This command updates a subnet in the existing server configuration. This operation has no impact on other subnets.

Supported by: *kea-dhcp4*

Availability: 1.6.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *subnet4-update command*

Command syntax:

```
{
    "command": "subnet4-update",
    "arguments": {
        "subnets": [ {
            "id": 123,
            "subnet": "10.20.30.0/24",
            ...
        } ]
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "IPv4 subnet updated",
    "arguments": {
        "subnets": [
            {
                "id": 123,
                "subnet": "10.20.30.0/24"
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.161 subnet6-add

This command creates and adds a new subnet to the existing server configuration. This operation has no impact on other subnets.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *subnet6-add command*

Command syntax:

```
{
    "command": "subnet6-add",
    "arguments": {
        "subnet6": [ {
            "id": 234,
            "subnet": "2001:db8:1::/64",
            ...
        } ]
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 subnet added",
    "arguments": {
        "subnet6": [
            {
                "id": 234,
                "subnet": "2001:db8:1::/64"
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.162 subnet6-del

This command removes a subnet from the server's configuration. This command has no effect on other configured subnets, but removing a subnet has certain implications which the server's administrator should be aware of.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *subnet6-del command*

Command syntax:

```
{
    "command": "subnet6-del",
    "arguments": {
        "id": 234
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 subnet 2001:db8:1::/64 (id 234) deleted",
    "subnets": [
        {
            "id": 234,
            "subnet": "2001:db8:1::/64"
        }
    ]
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.163 subnet6-get

This command retrieves detailed information about the specified subnet. This command usually follows `subnet6-list`, which discovers available subnets with their respective subnet identifiers and prefixes.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *subnet6-get command*

Command syntax:

```
{
    "command": "subnet6-get",
    "arguments": {
        "id": 11
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "Info about IPv6 subnet 2001:db8:1::/64 (id 11) returned",
    "arguments": {
        "subnets": [
            {
                "subnet": "2001:db8:1::/64",
                "id": 1,
                "option-data": [
                    ...
                ],
                ...
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.164 subnet6-list

This command lists all currently configured subnets. The subnets are returned in a brief format, i.e. a subnet identifier and subnet prefix are included for each subnet.

Supported by: *kea-dhcp6*

Availability: 1.3.0 (*subnet_cmds* hook library)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *subnet6-list command*

Command syntax:

```
{
    "command": "subnet6-list"
}
```

Response syntax:

```
{
    "result": 0,
    "text": "2 IPv6 subnets found",
    "arguments": {
        "subnets": [
            {
                "id": 11,
                "subnet": "2001:db8:1::/64"
            },
            {
                "id": 233,
                "subnet": "3000::/16"
            }
        ]
    }
}
```

If no IPv6 subnets are found, an error code is returned along with the error description.

23.165 subnet6-update

This command updates a subnet in the existing server configuration. This operation has no impact on other subnets.

Supported by: *kea-dhcp6*

Availability: 1.6.0 (*subnet_cmds* hook library)

Access: write *(parameter ignored in this Kea version)*

Description and examples: see *subnet6-update command*

Command syntax:

```
{
    "command": "subnet6-update",
    "arguments": {
        "subnet6": [ {
            "id": 234,
            "subnet": "2001:db8:1::/64",
            ...
        } ]
    }
}
```

Response syntax:

```
{
    "result": 0,
    "text": "IPv6 subnet updated",
    "arguments": {
        "subnet6": [
            {
                "id": 234,
                "subnet": "2001:db8:1::/64"
            }
        ]
    }
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

23.166 version-get

This command returns extended information about the Kea version that is running. The returned string is the same as if Kea were run with the `-V` command-line option.

Supported by: *kea-ctrl-agent*, *kea-dhcp-ddns*, *kea-dhcp4*, *kea-dhcp6*

Availability: 1.2.0 (built-in)

Access: read *(parameter ignored in this Kea version)*

Description and examples: see *version-get command*

Command syntax:

```
{
    "command": "version-get"
}
```

Response syntax:

```
{
    "result": <integer>,
    "text": "<string>"
}
```

Result is an integer representation of the status. Currently supported statuses are:

- 0 - success
- 1 - error
- 2 - unsupported
- 3 - empty (command was completed successfully, but no data was affected or returned)

CHAPTER

TWENTYFOUR

MANUAL PAGES

24.1 kea-dhcp4 - DHCPv4 server in Kea

24.1.1 Synopsis

kea-dhcp4 [**-v**] [**-V**] [**-W**] [**-d**] [**-c** config-file] [**-t** config-file] [**-p** server-port-number] [**-P** client-port-number]

24.1.2 Description

The `kea-dhcp4` daemon provides the DHCPv4 server implementation.

24.1.3 Arguments

The arguments are as follows:

-v Displays the version.

-V Displays the extended version.

-W Displays the configuration report.

-d Enables the debug mode with extra verbosity.

-c config-file Specifies the configuration file with the configuration for the DHCPv4 server. It may also contain configuration entries for other Kea services.

-t config-file Checks the configuration file and reports the first error, if any. Note that not all parameters are completely checked; in particular, service and control channel sockets are not opened, and hook libraries are not loaded.

-p server-port-number Specifies the server port number (1-65535) on which the server listens. This is useful for testing purposes only.

-P client-port-number Specifies the client port number (1-65535) to which the server responds. This is useful for testing purposes only.

24.1.4 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer's Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.1.5 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.1.6 History

The `b10-dhcp4` daemon was first coded in November 2011 by Tomek Mrugalski.

In mid-2014, Kea was decoupled from the BIND 10 framework and became a standalone DHCP server. The DHCPv4 server binary was renamed to kea-dhcp4. Kea 1.0.0 was released in December 2015.

24.1.7 See Also

`kea-dhcp6(8)`, *`kea-dhcp-ddns(8)`*, *`kea-ctrl-agent(8)`*, *`kea-admin(8)`*, *`keactrl(8)`*, *`perfdhcp(8)`*, *`kea-netconf(8)`*, *`kea-lfc(8)`*, Kea Administrator Reference Manual.

24.2 kea-dhcp6 - DHCPv6 server in Kea

24.2.1 Synopsis

`kea-dhcp6` [**-v**] [**-V**] [**-W**] [**-d**] [**-c** config-file] [**-t** config-file] [**-p** server-port-number] [**-P** client-port-number]

24.2.2 Description

The `kea-dhcp6` daemon provides the DHCPv6 server implementation.

24.2.3 Arguments

The arguments are as follows:

`-v` Displays the version.

`-V` Displays the extended version.

`-W` Displays the configuration report.

`-d` Enables the debug mode with extra verbosity.

`-c config-file` Specifies the configuration file with the configuration for the DHCPv6 server. It may also contain configuration entries for other Kea services.

`-t config-file` Checks the configuration file and reports the first error, if any. Note that not all parameters are completely checked; in particular, service and control channel sockets are not opened, and hook libraries are not loaded.

`-p server-port-number` Specifies the server port number (1-65535) on which the server listens. This is useful for testing purposes only.

`-P client-port-number` Specifies the client port number (1-65535) to which the server responds. This is useful for testing purposes only.

24.2.4 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer's Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.2.5 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.2.6 History

The `b10-dhcp6` daemon was first coded in June 2011 by Tomek Mrugalski.

Kea became a standalone server and the BIND 10 framework was removed. The DHCPv6 server binary was renamed to kea-dhcp6 in July 2014.

24.2.7 See Also

`kea-dhcp4(8)`, *`kea-dhcp-ddns(8)`*, *`kea-ctrl-agent(8)`*, *`kea-admin(8)`*, *`keactrl(8)`*, *`perfdhcp(8)`*, *`kea-netconf(8)`*, *`kea-lfc(8)`*, Kea Administrator Reference Manual.

24.3 kea-ctrl-agent - Control Agent process in Kea

24.3.1 Synopsis

`kea-ctrl-agent` [**-v**] [**-V**] [**-W**] [**-d**] [**-c** config-file] [**-t** config-file]

24.3.2 Description

The `kea-ctrl-agent` provides a REST service for controlling Kea services. The received HTTP requests are decapsulated and forwarded to the respective Kea services in JSON format. Received JSON responses are encapsulated within HTTP responses and returned to the controlling entity. Some commands may be handled by the Control Agent directly, and not forwarded to any Kea service.

24.3.3 Arguments

The arguments are as follows:

`-v` Displays the version.

`-V` Displays the extended version.

`-W` Displays the configuration report.

`-d` Sets the logging level to debug with extra verbosity. This is primarily for development purposes in stand-alone mode.

`-c config-file` Specifies the file with the configuration for the Control Agent server. It may also contain configuration entries for other Kea services.

`-t config-file` Checks the syntax of the configuration file and reports the first error, if any. Note that not all parameters are completely checked; in particular, service and client sockets are not opened, and hook libraries are not loaded.

24.3.4 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer's Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.3.5 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.3.6 History

The `kea-ctrl-agent` was first coded in December 2016 by Marcin Siodelski.

24.3.7 See Also

kea-dhcp4(8), *kea-dhcp6(8)*, *kea-dhcp-ddns(8)*, *kea-admin(8)*, *keactrl(8)*, *perfdhcp(8)*, *kea-lfc(8)*, Kea Administrator Reference Manual.

24.4 keactrl - Shell script for managing Kea

24.4.1 Synopsis

keactrl [**command**] [**-c** keactrl-config-file] [**-s** server[,server,...]] [**-v**]

24.4.2 Description

`keactrl` is a shell script which controls the startup, shutdown, and reconfiguration of the Kea servers (`kea-dhcp4`, `kea-dhcp6`, `kea-dhcp-ddns`, `kea-ctrl-agent`, and `kea-netconf`). It also provides the means for checking the current status of the servers and determining the configuration files in use.

24.4.3 Configuration File

Depending on the user's requirements, not all of the available servers need be run. The `keactrl` configuration file specifies which servers are enabled and which are disabled. By default the configuration file is `[kea-install-dir]/etc/kea/keactrl.conf`.

See the Kea Administrator Reference Manual for documentation of the parameters in the `keactrl` configuration file.

24.4.4 Options

`command` Specifies the command to be issued to the servers. It can be one of the following:

- **start** Starts the servers.
- **stop** Stops the servers.
- **reload** Instructs the servers to re-read the Kea configuration file. This command is not supported by the Netconf agent.
- **status** Prints the status of the servers.

`-c|--ctrl-config keactrl-config-file` Specifies the `keactrl` configuration file. Without this switch, `keactrl` attempts to use the file `[kea-install-dir]/etc/kea/keactrl.conf`.

`-s|--server server[,server,...]` Specifies a subset of the enabled servers to which the command should be issued. The list of servers should be separated by commas with no intervening spaces. Acceptable values are:

- **dhcp4** DHCPv4 server (`kea-dhcp4`).
- **dhcp6** DHCPv6 server (`kea-dhcp6`).
- **dhcp_ddns** DHCP DDNS server (`kea-dhcp-ddns`).
- **ctrl_agent** Control Agent (`kea-ctrl-agent`).
- **netconf** Netconf agent (`kea-netconf`).
- **all** All servers, including Netconf if it was configured to be built. This is the default.

`-v|--version` Prints the `keactrl` version and quits.

24.4.5 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer's Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.4.6 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.4.7 See Also

`kea-dhcp4(8)`, *`kea-dhcp6(8)`*, *`kea-dhcp-ddns(8)`*, *`kea-ctrl-agent(8)`*, *`kea-admin(8)`*, *`kea-netconf(8)`*, *`perfdhcp(8)`*, *`kea-lfc(8)`*, Kea Administrator Reference Manual.

24.5 kea-admin - Shell script for managing Kea databases

24.5.1 Synopsis

`kea-admin` [command] [backend] [**-u** database username] [**-p** database password] [**-n** database name] [**-d** scripts directory] [**-4** | **-6**] [**-o** output file] [**-v**]

24.5.2 Description

`kea-admin` is a shell script which offers database maintenance. In particular, it features database initialization, database version checking, and database schema upgrade.

24.5.3 Arguments

`command` Specifies the command to be issued to the servers. It can be one of the following:

db-init Initializes a new database schema. This is useful during a new Kea installation. The database is initialized to the latest version supported by the version of the software being installed.

db-version Reports the database backend version number. This is not necessarily equal to the Kea version number as each backend has its own versioning scheme.

db-upgrade Conducts a database schema upgrade. This is useful when upgrading Kea.

lease-dump Dumps the contents of the lease database (for MySQL, PostgreSQL, or CQL backends) to a CSV (comma-separated values) text file. The first line of the file contains the column names. This is meant to be used as a diagnostic tool, so it provides a portable, human-readable form of the lease data.

stats-recount Recounts lease statistics for MySQL or PostgreSQL database.

`backend` Specifies the backend type. Currently allowed backends are: memfile, mysql, and pgsql.

`-u|--user username` Specifies the username when connecting to a database. If not specified, the default value of **keatest** is used.

`-p|--password password` Specifies the password when connecting to a database. If not specified, the default value of **keatest** is used.

`-n|--name database-name` Specifies the name of the database to connect to. If not specified, the default value of **keatest** is used.

`-d|--directory script-directory` Specifies the override scripts directory. That script is used during upgrades, database initialization, and possibly other operations. If not specified, the default value of `(prefix)/share/kea/scripts/` is used.

`-o|--output output_file` Specifies the file to which the lease data will be dumped. Required for lease-dump.

`-v|--version` Prints the `kea-admin` version and quits.

`-4` Directs `kea-admin` to lease-dump the DHCPv4 leases. Incompatible with the -6 option.

`-6` Directs `kea-admin` to lease-dump the DHCPv6 leases. Incompatible with the -4 option.

24.5.4 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer's Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.5.5 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.5.6 See Also

`kea-dhcp4(8)`, *`kea-dhcp6(8)`*, *`kea-dhcp-ddns(8)`*, *`kea-ctrl-agent(8)`*, *`keactrl(8)`*, *`perfdhcp(8)`*, *`kea-netconf(8)`*, Kea Administrator Reference Manual.

24.6 kea-dhcp-ddns - DHCP-DDNS process in Kea

24.6.1 Synopsis

`kea-dhcp-ddns` [**-v**] [**-V**] [**-W**] [**-d**] [**-c** config-file] [**-t** config-file]

24.6.2 Description

The `kea-dhcp-ddns` service process requests an update of DNS mapping based on DHCP lease change events. It runs as a separate process that expects to receive Name Change Requests from Kea DHCP servers.

24.6.3 Arguments

The arguments are as follows:

`-v` Displays the version.

`-V` Displays the extended version.

`-W` Displays the configuration report.

`-d` Sets the logging level to debug with extra verbosity. This is primarily for development purposes in stand-alone mode.

`-c config-file` Specifies the configuration file with the configuration for the DHCP-DDNS server. It may also contain configuration entries for other Kea services.

`-t config-file` Checks the syntax of the configuration file and reports the first error if any. Note that not all parameters are completely checked, in particular, service socket is not opened.

24.6.4 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer's Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.6.5 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.6.6 History

The `b10-dhcp-ddns` process was first coded in May 2013 by Thomas Markwalder.

Kea became a standalone server and the BIND 10 framework was removed. The DHCP-DDNS server binary was renamed to kea-dhcp-ddns in July 2014. Kea 1.0.0 was released in December 2015.

24.6.7 See Also

kea-dhcp4(8), *kea-dhcp6(8)*, *kea-ctrl-agent(8)*, *kea-admin(8)*, *keactrl(8)*, *perfdhcp(8)*, *kea-netconf(8)*, *kea-lfc(8)*, Kea Administrator Reference Manual.

24.7 kea-lfc - Lease File Cleanup process in Kea

24.7.1 Synopsis

kea-lfc [**-4****|-6****] [**-c** config-file] [**-p** pid-file] [**-x** previous-file] [**-i** copy-file] [**-o** output-file] [**-f** finish-file] [**-v**] [**-V**] [**-W**] [**-d**] [**-h**]

24.7.2 Description

The `kea-lfc` service process removes redundant information from the files used to provide persistent storage for the memfile database backend. The service is written to run as a stand-alone process. While it can be started externally, there is usually no need to do this. It is run periodically by the Kea DHCP servers.

24.7.3 Arguments

The arguments are as follows:

-4 | -6 Indicates the protocol version of the lease files; must be either 4 or 6.

-c config-file Specifies the file with the configuration for the `kea-lfc` process. It may also contain configuration entries for other Kea services. Currently `kea-lfc` gets all of its arguments from the command line; in the future it will be extended to obtain some arguments from the configuration file.

-p pid-file Specifies the PID file. When the `kea-lfc` process starts, it attempts to determine if another instance of the process is already running by examining the PID file. If one is already running, the new process is terminated. If one is not running, Kea writes its PID into the PID file.

-x previous-file Specifies the previous or ex-lease file. When `kea-lfc` starts, this is the result of any previous run of `kea-lfc`; when `kea-lfc` finishes, it is the result of this run. If `kea-lfc` is interrupted before completing, this file may not exist.

-i copy-file Specifies the input or copy of lease file. Before the DHCP server invokes `kea-lfc`, it will move the current lease file here and then call `kea-lfc` with this file.

-o output-file Specifies the output lease file, which is the temporary file `kea-lfc` should use to write the leases. Once this file is finished writing, it is moved to the finish file (see below).

-f finish-file Specifies the finish or completion file, another temporary file `kea-lfc` uses for bookkeeping. When `kea-lfc` finishes writing the output file, it moves it to this file name. After `kea-lfc` finishes deleting the other files (previous and input), it moves this file to the previous lease file. By moving the files in this fashion, the `kea-lfc` and the DHCP server processes can determine the correct file to use even if one of the processes was interrupted before completing its task.

-v Causes the version stamp to be printed.

-V Causes a longer form of the version stamp to be printed.

-W Displays the configuration report.

-d Sets the logging level to debug with extra verbosity. This is primarily for development purposes in stand-alone mode.

-h Causes the usage string to be printed.

24.7.4 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer's Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.7.5 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.7.6 History

The `kea-lfc` process was first coded in January 2015 by the ISC Kea/DHCP team.

24.7.7 See Also

kea-dhcp4(8), *kea-dhcp6(8)*, *kea-dhcp-ddns(8)*, *kea-ctrl-agent(8)*, *kea-admin(8)*, *keactrl(8)*, *perfdhcp(8)*, *kea-netconf(8)*, Kea Administrator Reference Manual.

24.8 kea-shell - Text client for Control Agent process

24.8.1 Synopsis

kea-shell [**-h**] [**-v**] [**–host**] [**–port**] [**–path**] [**–auth-user**] [**–auth-password**] [**–timeout**] [**–service**] [command]

24.8.2 Description

The `kea-shell` provides a REST client for the Kea Control Agent (CA). It takes command as a command-line parameter that is being sent to CA with proper JSON encapsulation. Optional arguments may be specified on the standard input. The request is sent via HTTP and a response is retrieved, displayed on the standard output.

24.8.3 Arguments

The arguments are as follows:

`-h` Displays help regarding command-line parameters.

`-v` Displays the version.

`--host` Specifies the host to connect to. Control Agent must be running at the specified host. If not specified, 127.0.0.1 is used.

`--port` Specifies the TCP port to connect to. Control Agent must be listening at the specified port. If not specified, 8000 is used.

`--path` Specifies the path in the URL to connect to. If not specified, an empty path is used. As Control Agent listens at the empty path, this parameter is useful only with a reverse proxy.

`--auth-user` Specifies the user id for basic HTTP authentication. If not specified or specified as the empty string authentication is not used.

`--auth-password` Specifies the password for basic HTTP authentication. If not specified but the user id is specified an empty password is used.

`--timeout` Specifies the connection timeout in seconds. If not specified, 10 (seconds) is used.

`--service` Specifies the service that is the target of a command. If not specified, Control Agent will be targeted. May be used more than once to specify multiple targets.

`command` Specifies the command to be sent to CA. If not specified, “list-commands” is used.

24.8.4 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer’s Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.8.5 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.8.6 History

The `kea-shell` was first coded in March 2017 by Tomek Mrugalski.

24.8.7 See Also

`kea-dhcp4(8)`, *`kea-dhcp6(8)`*, *`kea-dhcp-ddns(8)`*, *`kea-ctrl-agent(8)`*, *`kea-admin(8)`*, *`keactrl(8)`*, *`perfdhcp(8)`*, *`kea-lfc(8)`*, Kea Administrator Reference Manual.

24.9 kea-netconf - NETCONF agent for Kea environment

24.9.1 Synopsis

`kea-netconf` [**-v**] [**-V**] [**-W**] [**-d**] [**-c** config-file] [**-t** config-file]

24.9.2 Description

The `kea-netconf` agent provides a YANG/NETCONF interface for the Kea environment.

24.9.3 Arguments

The arguments are as follows:

`-v` Displays the version.

`-V` Displays the extended version.

`-W` Displays the configuration report.

`-d` Enables the debug mode with extra verbosity.

`-c config-file` Specifies the file with the configuration for the NETCONF agent.

`-t config-file` Checks the syntax of the configuration file and reports the first error, if any. Note that not all parameters are completely checked; in particular, service and client sockets are not opened, and hook libraries are not loaded.

24.9.4 Documentation

Kea comes with an extensive Kea Administrator Reference Manual that covers all aspects of running the Kea software - compilation, installation, configuration, configuration examples, and much more. Kea also features a Kea Messages Manual, which lists all possible messages Kea can print with a brief description for each of them. Both documents are available in various formats (.txt, .html, .pdf) with the Kea distribution. The Kea documentation is available at https://kb.isc.org/docs/kea-administrator-reference-manual .

Kea source code is documented in the Kea Developer's Guide. Its online version is available at https://jenkins.isc.org/job/Kea_doc/doxygen/.

The Kea project website is available at https://kea.isc.org.

24.9.5 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.9.6 History

Early prototypes of `kea-netconf` implementation were written during IETF Hackathons in Berlin, London, and Montreal. An actual production-ready implementation was started in August 2018 by Tomek Mrugalski and Francis Dupont.

24.9.7 See Also

kea-dhcp4(8), *kea-dhcp6(8)*, *kea-dhcp-ddns(8)*, *kea-ctrl-agent(8)*, *kea-admin(8)*, *keactrl(8)*, *perfdhcp(8)*, *kea-lfc(8)*, Kea Administrator Reference Manual.

24.10 perfdhcp - DHCP benchmarking tool

24.10.1 Synopsis

perfdhcp [**-1**] [**-4 | -6**] [**-A** encapsulation-level] [**-b** base] [**-B**] [**-c**] [**-C** separator] [**-d** drop-time] [**-D** max-drop] [-e lease-type] [**-E** time-offset] [**-f** renew-rate] [**-F** release-rate] [**-g** thread-mode] [**-h**] [**-i**] [**-I** ip-offset] [**-J** giaddr-list-file] [**-l** local-address|interface] [**-L** local-port] [**-M** mac-list-file] [**-n** num-request] [**-N** remote-port] [**-O** random-offset] [**-o** code,hexstring] [**-p** test-period] [**-P** preload] [**-r** rate] [**-R** num-clients] [**-s** seed] [**-S** srvid-offset] [**–scenario** name] [**-t** report] [**-T** template-file] [**-u**] [**-v**] [**-W** exit-wait-time] [**-w** script_name] [**-x** diagnostic-selector] [**-X** xid-offset] [server]

24.10.2 Description

`perfdhcp` is a DHCP benchmarking tool. It provides a way of measuring the performance of DHCP servers by generating large amounts of traffic from simulated multiple clients. It is able to test both IPv4 and IPv6 servers, and provides statistics concerning response times and the number of requests that are dropped.

The tool supports scenarios, which offer certain behaviours. By default (basic scenario) tests are run using the full four-packet exchange sequence (DORA for DHCPv4, SARR for DHCPv6). An option is provided to run tests using the initial two-packet exchange (DO and SA) instead. It is also possible to configure `perfdhcp` to send DHCPv6 RENEW and RELEASE messages at a specified rate in parallel with the DHCPv6 four-way exchanges. By default, if there is no response received with 1 second, a response is considered lost and perfdhcp continues with other transactions.

Second scenario is called avalanche, which is selected by `--scenario avalanche`. It first sends as many Discovery or Solicit messages as request in -R option then a retransmission (with exponential back off mechanism) is used for each simulated client until all requests are answered. It will generate report when all clients get their addresses or when it will be manually stopped. This scenario attempts to replicate a case where the server is not able to handle the traffic swiftly enough. Real clients will assume the packet or the response was lost and will retransmit, further increasing DHCP traffic. This is sometimes called avalanche effect, thus the scenario name. Option `-p` is ignored in avalanche scenario.

When running a performance test, `perfdhcp` will exchange packets with the server under test as fast as possible unless the `-r` parameter is used to limit the request rate. The length of the test can be limited by setting a threshold on any or all of the number of requests made by `perfdhcp`, the elapsed time, or the number of requests dropped by the server.

24.10.3 Templates

To allow the contents of packets sent to the server to be customized, `perfdhcp` allows the specification of template files that determine the contents of the packets. For example, the customized packet may contain a DHCPv6 ORO to request a set of options to be returned by the server, or it may contain the Client FQDN option to request that the server perform DNS updates. This may be used to discover performance bottlenecks for different server configurations (e.g. DDNS enabled or disabled).

Up to two template files can be specified on the command line, each file representing the contents of a particular type of packet, the type being determined by the test being carried out. For example, if testing DHCPv6:

- With no template files specified on the command line, `perfdhcp` will generate both SOLICIT and REQUEST packets.
- With one template file specified, that file will be used as the pattern for SOLICIT packets: `perfdhcp` will generate the REQUEST packets.
- With two template files given on the command line, the first will be used as the pattern for SOLICIT packets, the second as the pattern for REQUEST packets.

(Similar determination applies to DHCPv4's DISCOVER and REQUEST packets.)

The template file holds the DHCP packet represented as a stream of ASCII hexadecimal digits and it excludes any IP/UDP stack headers. The template file must not contain any characters other than hexadecimal digits and spaces. Spaces are discarded when the template file is parsed; in the file, '12B4' is the same as '12 B4' which is the same as '1 2 B 4'.

The template files should be used in conjunction with the command-line parameters which specify offsets of the data fields being modified in outbound packets. For example, the `-E time-offset` switch specifies the offset of the DHCPv6 Elapsed Time option in the packet template. If the offset is specified, `perfdhcp` will inject the current elapsed-time value into this field before sending the packet to the server.

In many scenarios, `perfdhcp` needs to simulate multiple clients, each having a unique client identifier. Since packets for each client are generated from the same template file, it is necessary to randomize the client identifier (or HW address in DHCPv4) in the packet created from it. The `-O random-offset` option allows specification of the offset in the template where randomization should be performed. It is important to note that this offset points to the end (not the beginning) of the client identifier (or HW address field). The number of bytes being randomized depends on the number of simulated clients. If the number of simulated clients is between 1 and 255, only one byte (to which the randomization offset points) will be randomized. If the number of simulated clients is between 256 and 65535, two bytes will be randomized. Note that the last two bytes of the client identifier will be randomized in this case: the byte which the randomization offset parameter points to, and the one which precedes it (random-offset - 1). If the number of simulated clients exceeds 65535, three bytes will be randomized, and so on.

Perfdhcp can now simulate traffic from multiple subnets by enabling option -J and passing path to file that contains v4 addresses that will be used as giaddr in generated messages. That enable testing of vast numbers of Kea shared networks. Kea should be started with KEA_TEST_SEND_RESPONSES_TO_SOURCE environment variable to force Kea to send generated messages to source address of incoming packet. Feature is not available in kea-dhcp6.

Templates may currently be used to generate packets being sent to the server in 4-way exchanges, i.e. SOLICIT, REQUEST (DHCPv6) and DISCOVER, REQUEST (DHCPv4). They cannot be used when RENEW or RELEASE packets are being sent.

24.10.4 Options

`-1` Takes the server-ID option from the first received message.

`-4` Establishes DHCPv4 operation; this is the default. It is incompatible with the `-6` option.

`-6` Establishes DHCPv6 operation. This is incompatible with the `-4` option.

`-b basetype=value` Indicates the base MAC or DUID used to simulate different clients. The basetype may be “mac” or “duid”. (The keyword “ether” may alternatively used for MAC.) The `-b` option can be specified multiple times. The MAC address must consist of six octets separated by single (:) or double (::) colons, for example: mac=00:0c:01:02:03:04. The DUID value is a hexadecimal string; it must be at least six octets long and not longer than 64 bytes, and the length must be less than 128 hexadecimal digits, for example: duid=0101010101010101010110111F14.

`-d drop-time` Specifies the time after which a request is treated as having been lost. The value is given in seconds and may contain a fractional component. The default is 1 second.

`-e lease-type` Specifies the type of lease being requested from the server. It may be one of the following:

address-only Only regular addresses (v4 or v6) will be requested.

prefix-only Only IPv6 prefixes will be requested.

address-and-prefix Both IPv6 addresses and prefixes will be requested.

The `-e prefix-only` and `-e address-and-prefix` forms may not be used with the `-4` option.

`-f renew-rate` Specifies the rate at which DHCPv4 or DHCPv6 renew requests are sent to a server. This value is only valid when used in conjunction with the exchange rate (given by `-r rate`). Furthermore, the sum of this value and the release-rate (given by `-F rate`) must be equal to or less than the exchange rate.

`-g thread-mode` Allows selection of thread-mode, which can be either ‘single’ or ‘multi’. In multi-thread mode packets are received in a separate thread, which allows better utilisation of CPUs. In a single-CPU system it is better to run in one thread to avoid threads blocking each other. If more than one CPU is present in the system, multi-thread mode is the default; otherwise single-thread is the default.

`-h` Prints help and exits.

`-i` Performs only the initial part of the exchange: DISCOVER-OFFER if `-4` is selected, SOLICIT-ADVERTISE if `-6` is chosen.

`-i` is incompatible with the following options: `-1`, `-d`, `-D`, `-E`, `-S`, `-I` and `-F`. In addition, it cannot be used with multiple instances of `-O`, `-T` and `-X`.

`-l local-addr|interface` For DHCPv4 operation, specifies the local hostname/address to use when communicating with the server. By default, the interface address through which traffic would normally be routed to the server is used. For DHCPv6 operation, specifies the name of the network interface through which exchanges are initiated.

`-L local-port` Specifies the local port to use. This must be zero or a positive integer up to 65535. A value of 0 (the default) allows `perfdhcp` to choose its own port.

`-M mac-list-file` Specifies a text file containing a list of MAC addresses, one per line. If provided, a MAC address will be chosen randomly from this list for every new exchange. In DHCPv6, MAC addresses are used to generate DUID-LLs. This parameter must not be used in conjunction with the -b parameter.

`-N remote-port` Specifies the remote port to use. This must be zero or a positive integer up to 65535. A value of 0 (the default) allows `perfdhcp` to choose the standard service port.

`-o code,hexstring` Forces `perfdhcp` to insert the specified extra option (or options if used several times) into packets being transmitted. The code specifies the option code and the hexstring is a hexadecimal string that defines the content of the option. Care should be taken as `perfdhcp` does not offer any kind of logic behind those options; they are simply inserted into packets and sent as is. Be careful not to duplicate options that are already inserted. For example, to insert client class identifier (option code 60) with a string 'docsis', use -o 60,646f63736973. The `-o` may be used multiple times. It is necessary to specify the protocol family (either `-4` or `-6`) before using `-o`.

`-P preload` Initiates preload exchanges back-to-back at startup. Must be 0 (the default) or a positive integer.

`-r rate` Initiates the rate of DORA/SARR (or if `-i` is given, DO/SA) exchanges per second. A periodic report is generated showing the number of exchanges which were not completed, as well as the average response latency. The program continues until interrupted, at which point a final report is generated.

`-R num-clients` Specifies how many different clients are used. With a value of 1 (the default), all requests seem to come from the same client. Must be a positive number.

`-s seed` Specifies the seed for randomization, making runs of `perfdhcp` repeatable. This must be 0 or a positive integer. The value 0 means that a seed is not used; this is the default.

`--scenario name` Specifies type of the scenario, can be **basic** (default) or **avalanche**.

`-T template-file` Specifies a file containing the template to use as a stream of hexadecimal digits. This may be specified up to two times and controls the contents of the packets sent (see the "Templates" section above).

`-u` Enable checking address uniqueness. Lease valid lifetime should not be shorter than test duration and clients should not request address more than once without releasing it first.

`-v` Prints the version of this program.

`-W exit-wait-time` Specifies the exit-wait-time parameter, which causes `perfdhcp` to wait for exit-wait-time after an exit condition has been met, to receive all packets without sending any new packets. Expressed in microseconds. If not specified, 0 is used (i.e. exit immediately after exit conditions are met).

`-w script_name` Specifies the name of the script to be run before/after `perfdhcp`. When called, the script is passed a single parameter, either "start" or "stop", indicating whether it is being called before or after `perfdhcp`.

`-x diagnostic-selector` Includes extended diagnostics in the output. This is a string of single keywords specifying the operations for which verbose output is desired. The selector key letters are:

a Prints the decoded command line arguments.

e Prints the exit reason.

i Prints the rate processing details.

s Prints the first server-ID.

t When finished, prints timers of all successful exchanges.

T When finished, prints templates.

24.10.5 DHCPv4-Only Options

The following options only apply for DHCPv4 (i.e. when `-4` is given).

`-B` Forces broadcast handling.

`-J giaddr-list-file` Text file that include multiple addresses. If provided perfdhcp will choose randomly one of addresses for each exchange. This is used to generate traffic from multiple subnets. Designed to test shared-networks in kea-dhcp4. Kea should be started with KEA_TEST_SEND_RESPONSES_TO_SOURCE=ENABLE env variable otherwise perfdhcp will not be able to receive responses.

24.10.6 DHCPv6-Only Options

The following options only apply for DHCPv6 (i.e. when `-6` is given).

-c Adds a rapid-commit option (exchanges will be SOLICIT-ADVERTISE).

-F release-rate Specifies the rate at which IPv6 RELEASE requests are sent to a server. This value is only valid when used in conjunction with the exchange rate (given by `-r rate`). Furthermore, the sum of this value and the renew-rate (given by `-f rate`) must be equal to or less than the exchange rate value.

-A encapsulation-level Specifies that relayed traffic must be generated. The argument specifies the level of encapsulation, i.e. how many relay agents are simulated. Currently the only supported encapsulation-level value is 1, which means that the generated traffic is equivalent to the amount of traffic passing through a single relay agent.

24.10.7 Template-Related Options

The following options may only be used in conjunction with `-T` and control how `perfdhcp` modifies the template. The options may be specified multiple times on the command line; each occurrence affects the corresponding template file (see "Templates" above).

-E time-offset Specifies the offset of the secs field (DHCPv4) or elapsed-time option (DHCPv6) in the second (i.e. REQUEST) template; must be 0 or a positive integer. A value of 0 disables this.

-I ip-offset Specifies the offset of the IP address (DHCPv4) in the requested-IP option or IA_NA option (DHCPv6) in the second (REQUEST) template.

-O random-offset Specifies the offset of the last octet to randomize in the template. This must be an integer greater than 3. The `-T` switch must be given to use this option.

-S srvid-offset Specifies the offset of the server-ID option in the second (REQUEST) template. This must be a positive integer, and the switch can only be used when the template option (`-T`) is also given.

-X xid-offset Specifies the offset of the transaction ID (xid) in the template. This must be a positive integer, and the switch can only be used when the template option (`-T`) is also given.

24.10.8 Options Controlling a Test

-D max-drop Aborts the test immediately if **max-drop** requests have been dropped. Use `-D 0` to abort if even a single request has been dropped. **max-drop** must be a positive integer. If **max-drop** includes the suffix '%', it specifies a maximum percentage of requests that may be dropped before abort. In this case, testing of the threshold begins after 10 requests have been expected to be received.

-n num-requests Initiates **num-request** transactions. No report is generated until all transactions have been initiated/waited-for, after which a report is generated and the program terminates.

-p test-period Sends requests for **test-period**, which is specified in the same manner as `-d`. This can be used as an alternative to `-n` or both options can be given, in which case the testing is completed when either limit is reached.

-t interval Sets the delay (in seconds) between two successive reports.

-C separator Output reduced, an argument is a separator for periodic (-t) reports generated in easy parsable mode. Data output won't be changed, remain identical as in -t option.

24.10.9 Arguments

server Indicates the server to test, specified as an IP address. In the DHCPv6 case, the special name 'all' can be used to refer to All_DHCP_Relay_Agents_and_Servers (the multicast address FF02::1:2), or the special name 'servers' to refer to All_DHCP_Servers (the multicast address FF05::1:3). The server is mandatory except where the `-l` option is given to specify an interface, in which case it defaults to 'all'.

24.10.10 Errors

`perfdhcp` can report the following errors in the packet exchange:

tooshort A message was received that was too short.

orphans A message was received which does not match one sent to the server (i.e. it is a duplicate message, a message that has arrived after an excessive delay, or one that is just not recognized).

locallimit Local system limits have been reached when sending a message.

24.10.11 Exit Status

`perfdhcp` can exit with one of the following status codes:

0 Success.

1 General error.

2 Error in command-line arguments.

3 No general failures in operation, but one or more exchanges were unsuccessful.

24.10.12 Mailing Lists and Support

There are two public mailing lists available for the Kea project. **kea-users** (kea-users at lists.isc.org) is intended for Kea users, while **kea-dev** (kea-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. Both lists are available at https://lists.isc.org. The community provides best-effort support on both of those lists.

ISC provides professional support for Kea services. See https://www.isc.org/kea/ for details.

24.10.13 History

The `perfdhcp` tool was initially coded in October 2011 by John DuBois, Francis Dupont, and Marcin Siodelski of ISC. Kea 1.0.0, which included `perfdhcp`, was released in December 2015.

24.10.14 See Also

`kea-dhcp4(8)`, *`kea-dhcp6(8)`*, *`kea-dhcp-ddns(8)`*, *`kea-ctrl-agent(8)`*, *`kea-admin(8)`*, *`kea-netconf(8)`*, *`keactrl(8)`*, *`kea-lfc(8)`*, Kea Administrator Reference Manual.

CHAPTER

TWENTYFIVE

KEA MESSAGES MANUAL

Kea is an open source implementation of the Dynamic Host Configuration Protocol (DHCP) servers, developed and maintained by Internet Systems Consortium (ISC).

This is the reference guide for Kea version 1.9.0. Links to the most up-to-date version of this document (in PDF, HTML, and plain text formats), along with other documents for Kea, can be found in ISC's Knowledgebase.

25.1 ALLOC

ALLOC_ENGINE_LEASE_RECLAIMED

successfully reclaimed lease %1

> This debug message is logged when the allocation engine successfully reclaims a lease. The lease is now available for assignment.

ALLOC_ENGINE_REMOVAL_NCR_FAILED

sending removal name change request failed for lease %1: %2

> This error message is logged when sending a removal name change request to DHCP DDNS failed. This name change request is usually generated when the lease reclamation routine acts upon expired leases. If a lease being reclaimed has a corresponding DNS entry it needs to be removed. This message indicates that removal of the DNS entry has failed. Nevertheless the lease will be reclaimed.

ALLOC_ENGINE_V4_ALLOC_ERROR

%1: error during attempt to allocate an IPv4 address: %2

> An error occurred during an attempt to allocate an IPv4 address, the reason for the failure being contained in the message. The server will return a message to the client refusing a lease. The first argument includes the client identification information.

ALLOC_ENGINE_V4_ALLOC_FAIL

%1: failed to allocate an IPv4 address after %2 attempt(s)

> The DHCP allocation engine gave up trying to allocate an IPv4 address after the specified number of attempts. This probably means that the address pool from which the allocation is being attempted is either empty, or very nearly empty. As a result, the client will have been refused a lease. The first argument includes the client identification information. This message may indicate that your address pool is too small for the number of clients you are trying to service and should be expanded. Alternatively, if the you know that the number of concurrently active clients is less than the addresses you have available, you may want to consider reducing the lease lifetime. In this way, addresses allocated to clients that are no longer active on the network will become available sooner.

ALLOC_ENGINE_V4_ALLOC_FAIL_CLASSES

%1: Failed to allocate an IPv4 address for client with classes: %2

> This warning message is printed when Kea failed to allocate an address and the client's packet belongs to one or more classes. There may be several reasons why a lease was not assigned. One of them may be a case when all pools require packet to belong to certain classes and the incoming packet didn't belong to any of them. Another case where this information may be useful is to point out that the pool reserved to a given class has ran out of addresses. When you see this message, you may consider checking your pool size and your classification definitions.

ALLOC_ENGINE_V4_DECLINED_RECOVERED

IPv4 address %1 was recovered after %2 seconds of probation-period

> This informational message indicates that the specified address was reported as duplicate (client sent DECLINE) and the server marked this address as unvailable for a period of time. This time now has elapsed and the address has been returned to the available pool. This step concludes decline recovery process.

ALLOC_ENGINE_V4_DISCOVER_ADDRESS_CONFLICT

%1: conflicting reservation for address %2 with existing lease %3

> This warning message is issued when the DHCP server finds that the address reserved for the client can't be offered because this address is currently allocated to another client. The server will try to allocate a different address to the client to use until the conflict is resolved. The first argument includes the client identification information.

ALLOC_ENGINE_V4_DISCOVER_HR

client %1 sending DHCPDISCOVER has reservation for the address %2

> This message is issued when the allocation engine determines that the client sending the DHCPDISCOVER has a reservation for the specified address. The allocation engine will try to offer this address to the client.

ALLOC_ENGINE_V4_LEASES_RECLAMATION_COMPLETE

reclaimed %1 leases in %2

> This debug message is logged when the allocation engine completes reclamation of a set of expired leases. The maximum number of leases to be reclaimed in a single pass of the lease reclamation routine is configurable using 'max-reclaim-leases' parameter. However, the number of reclaimed leases may also be limited by the timeout value, configured with 'max-reclaim-time'. The message includes the number of reclaimed leases and the total time.

ALLOC_ENGINE_V4_LEASES_RECLAMATION_SLOW

expired leases still exist after %1 reclamations

> This warning message is issued when the server has been unable to reclaim all expired leases in a specified number of consecutive attempts. This indicates that the value of "reclaim-timer-wait-time" may be too high. However, if this is just a short burst of leases' expirations the value does not have to be modified and the server should deal with this in subsequent reclamation attempts. If this is a result of a permanent increase of the server load, the value of "reclaim-timer-wait-time" should be decreased, or the values of "max-reclaim-leases" and "max-reclaim-time" should be increased to allow processing more leases in a single cycle. Alternatively, these values may be set to 0 to remove the limitations on the number of leases and duration. However, this may result in longer periods of server's unresponsiveness to DHCP packets, while it processes the expired leases.

ALLOC_ENGINE_V4_LEASES_RECLAMATION_START

starting reclamation of expired leases (limit = %1 leases or %2 milliseconds)

> This debug message is issued when the allocation engine starts the reclamation of the expired leases. The maximum number of leases to be reclaimed and the timeout is included in the message. If any of these values is 0, it means “unlimited”.

ALLOC_ENGINE_V4_LEASES_RECLAMATION_TIMEOUT

timeout of %1 ms reached while reclaiming IPv4 leases

> This debug message is issued when the allocation engine hits the timeout for performing reclamation of the expired leases. The reclamation will now be interrupted and all leases which haven’t been reclaimed, because of the timeout, will be reclaimed when the next scheduled reclamation is started. The argument is the timeout value expressed in milliseconds.

ALLOC_ENGINE_V4_LEASE_RECLAIM

%1: reclaiming expired lease for address %2

> This debug message is issued when the server begins reclamation of the expired DHCPv4 lease. The first argument specifies the client identification information. The second argument holds the leased IPv4 address.

ALLOC_ENGINE_V4_LEASE_RECLAMATION_FAILED

failed to reclaim the lease %1: %2

> This error message is logged when the allocation engine fails to reclaim an expired lease. The reason for the failure is included in the message. The error may be triggered in the lease expiration hook or while performing the operation on the lease database.

ALLOC_ENGINE_V4_NO_MORE_EXPIRED_LEASES

all expired leases have been reclaimed

> This debug message is issued when the server reclaims all expired DHCPv4 leases in the database.

ALLOC_ENGINE_V4_OFFER_EXISTING_LEASE

allocation engine will try to offer existing lease to the client %1

> This message is issued when the allocation engine determines that the client has a lease in the lease database, it doesn’t have reservation for any other lease, and the leased address is not reserved for any other client. The allocation engine will try to offer the same lease to the client.

ALLOC_ENGINE_V4_OFFER_NEW_LEASE

allocation engine will try to offer new lease to the client %1

> This message is issued when the allocation engine will try to offer a new lease to the client. This is the case when the client doesn’t have any existing lease, it has no reservation or the existing or reserved address is leased to another client. Also, the client didn’t specify a hint, or the address in the hint is in use.

ALLOC_ENGINE_V4_OFFER_REQUESTED_LEASE

allocation engine will try to offer requested lease %1 to the client %2

> This message is issued when the allocation engine will try to offer the lease specified in the hint. This situation may occur when: (a) client doesn’t have any reservations, (b) client has reservation but the reserved address is leased to another client.

ALLOC_ENGINE_V4_RECLAIMED_LEASES_DELETE

begin deletion of reclaimed leases expired more than %1 seconds ago

This debug message is issued when the allocation engine begins deletion of the reclaimed leases which have expired more than a specified number of seconds ago. This operation is triggered periodically according to the "flush-reclaimed-timer-wait-time" parameter. The "hold-reclaimed-time" parameter defines a number of seconds for which the leases are stored before they are removed.

ALLOC_ENGINE_V4_RECLAIMED_LEASES_DELETE_COMPLETE

successfully deleted %1 expired-reclaimed leases

This debug message is issued when the server successfully deletes "expired-reclaimed" leases from the lease database. The number of deleted leases is included in the log message.

ALLOC_ENGINE_V4_RECLAIMED_LEASES_DELETE_FAILED

deletion of expired-reclaimed leases failed: %1

This error message is issued when the deletion of "expired-reclaimed" leases from the database failed. The error message is appended to the log message.

ALLOC_ENGINE_V4_REQUEST_ADDRESS_RESERVED

%1: requested address %2 is reserved

This message is issued when the allocation engine refused to allocate address requested by the client because this address is reserved for another client. The first argument includes the client identification information.

ALLOC_ENGINE_V4_REQUEST_ALLOC_REQUESTED

%1: trying to allocate requested address %2

This message is issued when the allocation engine is trying to allocate (or reuse an expired) address which has been requested by the client. The first argument includes the client identification information.

ALLOC_ENGINE_V4_REQUEST_EXTEND_LEASE

%1: extending lifetime of the lease for address %2

This message is issued when the allocation engine determines that the client already has a lease whose lifetime can be extended, and which can be returned to the client. The first argument includes the client identification information.

ALLOC_ENGINE_V4_REQUEST_INVALID

client %1 having a reservation for address %2 is requesting invalid address %3

This message is logged when the client, having a reservation for one address, is requesting a different address. The client is only allowed to do this when the reserved address is in use by another client. However, the allocation engine has determined that the reserved address is available and the client should request the reserved address.

ALLOC_ENGINE_V4_REQUEST_IN_USE

%1: requested address %2 is in use

This message is issued when the client is requesting or has a reservation for an address which is in use. The first argument includes the client identification information.

ALLOC_ENGINE_V4_REQUEST_OUT_OF_POOL

client %1, which doesn't have a reservation, requested address %2 out of the dynamic pool

This message is issued when the client has requested allocation of the address which doesn't belong to any address pool from which addresses are dynamically allocated. The client also doesn't have reservation for this address. This address could only be allocated if the client had reservation for it.

ALLOC_ENGINE_V4_REQUEST_PICK_ADDRESS

client %1 hasn't specified an address - picking available address from the pool

> This message is logged when the client hasn't specified any preferred address (the client should always do it, but Kea tries to be forgiving). The allocation engine will try to pick an available address from the dynamic pool and allocate it to the client.

ALLOC_ENGINE_V4_REQUEST_REMOVE_LEASE

%1: removing previous client's lease %2

> This message is logged when the allocation engine removes previous lease for the client because the client has been allocated new one.

ALLOC_ENGINE_V4_REQUEST_USE_HR

client %1 hasn't requested specific address, using reserved address %2

> This message is issued when the client is not requesting any specific address but the allocation engine has determined that there is a reservation for this client. The allocation engine will try to allocate the reserved address.

ALLOC_ENGINE_V4_REUSE_EXPIRED_LEASE_DATA

%1: reusing expired lease, updated lease information: %2

> This message is logged when the allocation engine is reusing an existing lease. The details of the updated lease are printed. The first argument includes the client identification information.

ALLOC_ENGINE_V6_ALLOC_ERROR

%1: error during attempt to allocate an IPv6 address: %2

> An error occurred during an attempt to allocate an IPv6 address, the reason for the failure being contained in the message. The server will return a message to the client refusing a lease. The first argument includes the client identification information.

ALLOC_ENGINE_V6_ALLOC_FAIL

%1: failed to allocate an IPv6 address after %2 attempt(s)

> The DHCP allocation engine gave up trying to allocate an IPv6 address after the specified number of attempts. This probably means that the address pool from which the allocation is being attempted is either empty, or very nearly empty. As a result, the client will have been refused a lease. The first argument includes the client identification information. This message may indicate that your address pool is too small for the number of clients you are trying to service and should be expanded. Alternatively, if the you know that the number of concurrently active clients is less than the addresses you have available, you may want to consider reducing the lease lifetime. In this way, addresses allocated to clients that are no longer active on the network will become available sooner.

ALLOC_ENGINE_V6_ALLOC_HR_LEASE_EXISTS

%1: lease type %2 for reserved address/prefix %3 already exists

> This debug message is issued when the allocation engine determines that the lease for the IPv6 address or prefix has already been allocated for the client and the client can continue using it. The first argument includes the client identification information.

ALLOC_ENGINE_V6_ALLOC_LEASES_HR

leases and static reservations found for client %1

> This message is logged when the allocation engine is in the process of allocating leases for the client, it found existing leases and static reservations for the client. The allocation engine will verify if existing

leases match reservations. Those leases that are reserved for other clients and those that are not reserved for the client will be removed. All leases matching the reservations will be renewed and returned.

ALLOC_ENGINE_V6_ALLOC_LEASES_NO_HR

no reservations found but leases exist for client %1

This message is logged when the allocation engine is in the process if allocating leases for the client, there are no static reservations, but lease(s) exist for the client. The allocation engine will remove leases which are reserved for other clients, and return all remaining leases to the client.

ALLOC_ENGINE_V6_ALLOC_NO_LEASES_HR

no leases found but reservations exist for client %1

This message is logged when the allocation engine is in the process of allocating leases for the client. It hasn't found any existing leases for this client, but the client appears to have static reservations. The allocation engine will try to allocate the reserved resources for the client.

ALLOC_ENGINE_V6_ALLOC_NO_V6_HR

%1: unable to allocate reserved leases - no IPv6 reservations

This message is logged when the allocation engine determines that the client has no IPv6 reservations and thus the allocation engine will have to try to allocate allocating leases from the dynamic pool or stop the allocation process if none can be allocated. The first argument includes the client identification information.

ALLOC_ENGINE_V6_ALLOC_UNRESERVED

no static reservations available - trying to dynamically allocate leases for client %1

This debug message is issued when the allocation engine will attempt to allocate leases from the dynamic pools. This may be due to one of (a) there are no reservations for this client, (b) there are reservations for the client but they are not usable because the addresses are in use by another client or (c) we had a reserved lease but that has now been allocated to another client.

ALLOC_ENGINE_V6_DECLINED_RECOVERED

IPv6 address %1 was recovered after %2 seconds of probation-period

This informational message indicates that the specified address was reported as duplicate (client sent DECLINE) and the server marked this address as unvailable for a period of time. This time now has elapsed and the address has been returned to the available pool. This step concludes decline recovery process.

ALLOC_ENGINE_V6_EXPIRED_HINT_RESERVED

%1: expired lease for the client's hint %2 is reserved for another client

This message is logged when the allocation engine finds that the expired lease for the client's hint can't be reused because it is reserved for another client. The first argument includes the client identification information.

ALLOC_ENGINE_V6_EXTEND_ALLOC_UNRESERVED

allocate new (unreserved) leases for the renewing client %1

This debug message is issued when the allocation engine is trying to allocate new leases for the renewing client because it was unable to renew any of the existing client's leases, e.g. because leases are reserved for another client or for any other reason.

ALLOC_ENGINE_V6_EXTEND_ERROR

%1: allocation engine experienced error with attempting to extend lease lifetime: %2

This error message indicates that an error was experienced during Renew or Rebind processing. Additional explanation is provided with this message. Depending on its nature, manual intervention may be required to continue processing messages from this particular client; other clients will be unaffected. The first argument includes the client identification information.

ALLOC_ENGINE_V6_EXTEND_LEASE

%1: extending lifetime of the lease type %2, address %3

This debug message is issued when the allocation engine is trying to extend lifetime of the lease. The first argument includes the client identification information.

ALLOC_ENGINE_V6_EXTEND_LEASE_DATA

%1: detailed information about the lease being extended: %2

This debug message prints detailed information about the lease which lifetime is being extended (renew or rebind). The first argument includes the client identification information.

ALLOC_ENGINE_V6_EXTEND_NEW_LEASE_DATA

%1: new lease information for the lease being extended: %2

This debug message prints updated information about the lease to be extended. If the lease update is successful, the information printed by this message will be stored in the database. The first argument includes the client identification information.

ALLOC_ENGINE_V6_HINT_RESERVED

%1: lease for the client's hint %2 is reserved for another client

This message is logged when the allocation engine cannot allocate the lease using the client's hint because the lease for this hint is reserved for another client. The first argument includes the client identification information.

ALLOC_ENGINE_V6_HR_ADDR_GRANTED

reserved address %1 was assigned to client %2

This informational message signals that the specified client was assigned the address reserved for it.

ALLOC_ENGINE_V6_HR_PREFIX_GRANTED

reserved prefix %1/%2 was assigned to client %3

This informational message signals that the specified client was assigned the prefix reserved for it.

ALLOC_ENGINE_V6_LEASES_RECLAMATION_COMPLETE

reclaimed %1 leases in %2

This debug message is logged when the allocation engine completes reclamation of a set of expired leases. The maximum number of leases to be reclaimed in a single pass of the lease reclamation routine is configurable using 'max-reclaim-leases' parameter. However, the number of reclaimed leases may also be limited by the timeout value, configured with 'max-reclaim-time'. The message includes the number of reclaimed leases and the total time.

ALLOC_ENGINE_V6_LEASES_RECLAMATION_SLOW

expired leases still exist after %1 reclamations

This warning message is issued when the server has been unable to reclaim all expired leases in a specified number of consecutive attempts. This indicates that the value of "reclaim-timer-wait-time" may be too high. However, if this is just a short burst of leases' expirations the value does not have to be modified and the server should deal with this in subsequent reclamation attempts. If this is a result of a permanent increase of the server load, the value of "reclaim-timer-wait-time" should be decreased, or the values of

> "max-reclaim-leases" and "max-reclaim-time" should be increased to allow processing more leases in a single cycle. Alternatively, these values may be set to 0 to remove the limitations on the number of leases and duration. However, this may result in longer periods of server's unresponsiveness to DHCP packets, while it processes the expired leases.

ALLOC_ENGINE_V6_LEASES_RECLAMATION_START

starting reclamation of expired leases (limit = %1 leases or %2 milliseconds)

> This debug message is issued when the allocation engine starts the reclamation of the expired leases. The maximum number of leases to be reclaimed and the timeout is included in the message. If any of these values is 0, it means "unlimited".

ALLOC_ENGINE_V6_LEASES_RECLAMATION_TIMEOUT

timeout of %1 ms reached while reclaiming IPv6 leases

> This debug message is issued when the allocation engine hits the timeout for performing reclamation of the expired leases. The reclamation will now be interrupted and all leases which haven't been reclaimed, because of the timeout, will be reclaimed when the next scheduled reclamation is started. The argument is the timeout value expressed in milliseconds.

ALLOC_ENGINE_V6_LEASE_RECLAIM

%1: reclaiming expired lease for prefix %2/%3

> This debug message is issued when the server begins reclamation of the expired DHCPv6 lease. The reclaimed lease may either be an address lease or delegated prefix. The first argument provides the client identification information. The other arguments specify the prefix and the prefix length for the lease. The prefix length for address lease is equal to 128.

ALLOC_ENGINE_V6_LEASE_RECLAMATION_FAILED

failed to reclaim the lease %1: %2

> This error message is logged when the allocation engine fails to reclaim an expired lease. The reason for the failure is included in the message. The error may be triggered in the lease expiration hook or while performing the operation on the lease database.

ALLOC_ENGINE_V6_NO_MORE_EXPIRED_LEASES

all expired leases have been reclaimed

> This debug message is issued when the server reclaims all expired DHCPv6 leases in the database.

ALLOC_ENGINE_V6_RECLAIMED_LEASES_DELETE

begin deletion of reclaimed leases expired more than %1 seconds ago

> This debug message is issued when the allocation engine begins deletion of the reclaimed leases which have expired more than a specified number of seconds ago. This operation is triggered periodically according to the "flush-reclaimed-timer-wait-time" parameter. The "hold-reclaimed-time" parameter defines a number of seconds for which the leases are stored before they are removed.

ALLOC_ENGINE_V6_RECLAIMED_LEASES_DELETE_COMPLETE

successfully deleted %1 expired-reclaimed leases

> This debug message is issued when the server successfully deletes "expired-reclaimed" leases from the lease database. The number of deleted leases is included in the log message.

ALLOC_ENGINE_V6_RECLAIMED_LEASES_DELETE_FAILED

deletion of expired-reclaimed leases failed: %1

> This error message is issued when the deletion of "expired-reclaimed" leases from the database failed. The error message is appended to the log message.

ALLOC_ENGINE_V6_RENEW_HR

allocating leases reserved for the client %1 as a result of Renew

> This debug message is issued when the allocation engine tries to allocate reserved leases for the client sending a Renew message. The server will also remove any leases that the client is trying to renew that are not reserved for the client.

ALLOC_ENGINE_V6_RENEW_REMOVE_RESERVED

%1: checking if existing client's leases are reserved for another client

> This message is logged when the allocation engine finds leases for the client and will check if these leases are reserved for another client. If they are, they will not be renewed for the client requesting their renewal. The first argument includes the client identification information.

ALLOC_ENGINE_V6_RENEW_REMOVE_UNRESERVED

dynamically allocating leases for the renewing client %1

> This debug message is issued as the allocation engine is trying to dynamically allocate new leases for the renewing client. This is the case when the server couldn't renew any of the existing client's leases, e.g. because leased resources are reserved for another client.

ALLOC_ENGINE_V6_REUSE_EXPIRED_LEASE_DATA

%1: reusing expired lease, updated lease information: %2

> This message is logged when the allocation engine is reusing an existing lease. The details of the updated lease are printed. The first argument includes the client identification information.

ALLOC_ENGINE_V6_REVOKED_ADDR_LEASE

address %1 was revoked from client %2 as it is reserved for client %3

> This informational message is an indication that the specified IPv6 address was used by client A but it is now reserved for client B. Client A has been told to stop using it so that it can be leased to client B. This is a normal occurrence during conflict resolution, which can occur in cases such as the system administrator adding a reservation for an address that is currently in use by another client. The server will fully recover from this situation, but clients will change their addresses.

25.2 ASIODNS

ASIODNS_FD_ADD_TCP

adding a new TCP server by opened fd %1

> A debug message informing about installing a file descriptor as a server. The file descriptor number is noted.

ASIODNS_FD_ADD_UDP

adding a new UDP server by opened fd %1

> A debug message informing about installing a file descriptor as a server. The file descriptor number is noted.

ASIODNS_FETCH_COMPLETED

upstream fetch to %1(%2) has now completed

A debug message, this records that the upstream fetch (a query made by the resolver on behalf of its client) to the specified address has completed.

ASIODNS_FETCH_STOPPED

upstream fetch to %1(%2) has been stopped

An external component has requested the halting of an upstream fetch. This is an allowed operation, and the message should only appear if debug is enabled.

ASIODNS_OPEN_SOCKET

error %1 opening %2 socket to %3(%4)

The asynchronous I/O code encountered an error when trying to open a socket of the specified protocol in order to send a message to the target address. The number of the system error that caused the problem is given in the message.

ASIODNS_READ_DATA

error %1 reading %2 data from %3(%4)

The asynchronous I/O code encountered an error when trying to read data from the specified address on the given protocol. The number of the system error that caused the problem is given in the message.

ASIODNS_READ_TIMEOUT

receive timeout while waiting for data from %1(%2)

An upstream fetch from the specified address timed out. This may happen for any number of reasons and is most probably a problem at the remote server or a problem on the network. The message will only appear if debug is enabled.

ASIODNS_SEND_DATA

error %1 sending data using %2 to %3(%4)

The asynchronous I/O code encountered an error when trying to send data to the specified address on the given protocol. The number of the system error that caused the problem is given in the message.

ASIODNS_SYNC_UDP_CLOSE_FAIL

failed to close a DNS/UDP socket: %1

This is the same to ASIODNS_UDP_CLOSE_FAIL but happens on the “synchronous UDP server”, mainly used for the authoritative DNS server daemon.

ASIODNS_TCP_ACCEPT_FAIL

failed to accept TCP DNS connection: %1

Accepting a TCP connection from a DNS client failed due to an error that could happen but should be rare. The reason for the error is included in the log message. The server still keeps accepting new connections, so unless it happens often it’s probably okay to ignore this error. If the shown error indicates something like “too many open files”, it’s probably because the run time environment is too restrictive on this limitation, so consider adjusting the limit using a tool such as ulimit. If you see other types of errors too often, there may be something overlooked; please file a bug report in that case.

ASIODNS_TCP_CLEANUP_CLOSE_FAIL

failed to close a DNS/TCP socket on port cleanup: %1

A TCP DNS server tried to close a TCP socket (one created on accepting a new connection or is already unused) as a step of cleaning up the corresponding listening port, but it failed to do that. This is generally an unexpected event and so is logged as an error. See also the description of ASIODNS_TCP_CLOSE_ACCEPTOR_FAIL.

ASIODNS_TCP_CLOSE_ACCEPTOR_FAIL

failed to close listening TCP socket: %1

> A TCP DNS server tried to close a listening TCP socket (for accepting new connections) as a step of cleaning up the corresponding listening port (e.g., on server shutdown or updating port configuration), but it failed to do that. This is generally an unexpected event and so is logged as an error. See ASIODNS_TCP_CLOSE_FAIL on the implication of related system resources.

ASIODNS_TCP_CLOSE_FAIL

failed to close DNS/TCP socket with a client: %1

> A TCP DNS server tried to close a TCP socket used to communicate with a client, but it failed to do that. While closing a socket should normally be an error-free operation, there have been known cases where this happened with a "connection reset by peer" error. This might be because of some odd client behavior, such as sending a TCP RST after establishing the connection and before the server closes the socket, but how exactly this could happen seems to be system dependent (i.e, it's not part of the standard socket API), so it's difficult to provide a general explanation. In any case, it is believed that an error on closing a socket doesn't mean leaking system resources (the kernel should clean up any internal resource related to the socket, just reporting an error detected in the close call), but, again, it seems to be system dependent. This message is logged at a debug level as it's known to happen and could be triggered by a remote node and it would be better to not be too verbose, but you might want to increase the log level and make sure there's no resource leak or other system level troubles when it's logged.

ASIODNS_TCP_CLOSE_NORESP_FAIL

failed to close DNS/TCP socket with a client: %1

> A TCP DNS server tried to close a TCP socket used to communicate with a client without returning an answer (which normally happens for zone transfer requests), but it failed to do that. See ASIODNS_TCP_CLOSE_FAIL for more details.

ASIODNS_TCP_GETREMOTE_FAIL

failed to get remote address of a DNS TCP connection: %1

> A TCP DNS server tried to get the address and port of a remote client on a connected socket but failed. It's expected to be rare but can still happen. See also ASIODNS_TCP_READLEN_FAIL.

ASIODNS_TCP_READDATA_FAIL

failed to get DNS data on a TCP socket: %1

> A TCP DNS server tried to read a DNS message (that follows a 2-byte length field) but failed. It's expected to be rare but can still happen. See also ASIODNS_TCP_READLEN_FAIL.

ASIODNS_TCP_READLEN_FAIL

failed to get DNS data length on a TCP socket: %1

> A TCP DNS server tried to get the length field of a DNS message (the first 2 bytes of a new chunk of data) but failed. This is generally expected to be rare but can still happen, e.g, due to an unexpected reset of the connection. A specific reason for the failure is included in the log message.

ASIODNS_TCP_WRITE_FAIL

failed to send DNS message over a TCP socket: %1

> A TCP DNS server tried to send a DNS message to a remote client but failed. It's expected to be rare but can still happen. See also ASIODNS_TCP_READLEN_FAIL.

ASIODNS_UDP_ASYNC_SEND_FAIL

Error sending UDP packet to %1: %2

> The low-level ASIO library reported an error when trying to send a UDP packet in asynchronous UDP mode. This can be any error reported by send_to(), and can indicate problems such as too high a load on the network, or a problem in the underlying library or system. This packet is dropped and will not be sent, but service should resume normally. If you see a single occurrence of this message, it probably does not indicate any significant problem, but if it is logged often, it is probably a good idea to inspect your network traffic.

ASIODNS_UDP_CLOSE_FAIL

failed to close a DNS/UDP socket: %1

> A UDP DNS server tried to close its UDP socket, but failed to do that. This is generally an unexpected event and so is logged as an error.

ASIODNS_UDP_RECEIVE_FAIL

failed to receive UDP DNS packet: %1

> Receiving a UDP packet from a DNS client failed due to an error that could happen but should be very rare. The server still keeps receiving UDP packets on this socket. The reason for the error is included in the log message. This log message is basically not expected to appear at all in practice; if it does, there may be some system level failure and other system logs may have to be checked.

ASIODNS_UDP_SYNC_RECEIVE_FAIL

failed to receive UDP DNS packet: %1

> This is the same to ASIODNS_UDP_RECEIVE_FAIL but happens on the “synchronous UDP server”, mainly used for the authoritative DNS server daemon.

ASIODNS_UDP_SYNC_SEND_FAIL

Error sending UDP packet to %1: %2

> The low-level ASIO library reported an error when trying to send a UDP packet in synchronous UDP mode. See ASIODNS_UDP_ASYNC_SEND_FAIL for more information.

ASIODNS_UNKNOWN_ORIGIN

unknown origin for ASIO error code %1 (protocol: %2, address %3)

> An internal consistency check on the origin of a message from the asynchronous I/O module failed. This may indicate an internal error; please submit a bug report.

25.3 BOOTP

BOOTP_BOOTP_QUERY

recognized a BOOTP query: %1

> This debug message is printed when the BOOTP query was recognized. The BOOTP client class was added and the message type set to DHCPREQUEST. The query client and transaction identification are displayed.

BOOTP_LOAD

Bootp hooks library has been loaded

> This info message indicates that the Bootp hooks library has been loaded.

BOOTP_PACKET_OPTIONS_SKIPPED

an error upacking an option, caused subsequent options to be skipped: %1

A debug message issued when an option failed to unpack correctly, making it impossible to unpack the remaining options in the DHCPv4 query. The server will still attempt to service the packet. The sole argument provides a reason for unpacking error.

BOOTP_PACKET_PACK

%1: preparing on-wire format of the packet to be sent

This debug message is issued when the server starts preparing the on-wire format of the packet to be sent back to the client. The argument specifies the client and the transaction identification information.

BOOTP_PACKET_PACK_FAIL

%1: preparing on-wire-format of the packet to be sent failed %2

This error message is issued when preparing an on-wire format of the packet has failed. The first argument identifies the client and the BOOTP transaction. The second argument includes the error string.

BOOTP_PACKET_UNPACK_FAILED

failed to parse query from %1 to %2, received over interface %3, reason: %4

This debug message is issued when received DHCPv4 query is malformed and can't be parsed by the buffer4_receive callout. The query will be dropped by the server. The first three arguments specify source IP address, destination IP address and the interface. The last argument provides a reason for failure.

25.4 COMMAND

COMMAND_ACCEPTOR_START

Starting to accept connections via unix domain socket bound to %1

This informational message is issued when the Kea server starts an acceptor via which it is going to accept new control connections. The acceptor is bound to the endpoint associated with the filename provided as an argument. If starting the acceptor fails, subsequent error messages will provide a reason for failure.

COMMAND_DEREGISTERED

Command %1 deregistered

This debug message indicates that the daemon stopped supporting specified command. This command can no longer be issued. If the command socket is open and this command is issued, the daemon will not be able to process it.

COMMAND_EXTENDED_REGISTERED

Command %1 registered

This debug message indicates that the daemon started supporting specified command. The handler for the registered command includes a parameter holding entire command to be processed.

COMMAND_PROCESS_ERROR1

Error while processing command: %1

This warning message indicates that the server encountered an error while processing received command. Additional information will be provided, if available. Additional log messages may provide more details.

COMMAND_PROCESS_ERROR2

Error while processing command: %1

This warning message indicates that the server encountered an error while processing received command. The difference, compared to COMMAND_PROCESS_ERROR1 is that the initial command was well formed and the error occurred during logic processing, not the command parsing. Additional information will be provided, if available. Additional log messages may provide more details.

COMMAND_RECEIVED

Received command '%1'

This informational message indicates that a command was received over command socket. The nature of this command and its possible results will be logged with separate messages.

COMMAND_REGISTERED

Command %1 registered

This debug message indicates that the daemon started supporting specified command. If the command socket is open, this command can now be issued.

COMMAND_RESPONSE_ERROR

Server failed to generate response for command: %1

This error message indicates that the server failed to generate response for specified command. This likely indicates a server logic error, as the server is expected to generate valid responses for all commands, even malformed ones.

COMMAND_SOCKET_ACCEPT_FAIL

Failed to accept incoming connection on command socket %1: %2

This error indicates that the server detected incoming connection and executed accept system call on said socket, but this call returned an error. Additional information may be provided by the system as second parameter.

COMMAND_SOCKET_CLOSED_BY_FOREIGN_HOST

Closed command socket %1 by foreign host, %2

This is an information message indicating that the command connection has been closed by a command control client, and whether or not any partially read data was discarded.

COMMAND_SOCKET_CONNECTION_CANCEL_FAIL

Failed to cancel read operation on socket %1: %2

This error message is issued to indicate an error to cancel asynchronous read of the control command over the control socket. The cancel operation is performed when the timeout occurs during communication with a client. The error message includes details about the reason for failure.

COMMAND_SOCKET_CONNECTION_CLOSED

Closed socket %1 for existing command connection

This is a debug message indicating that the socket created for handling client's connection is closed. This usually means that the client disconnected, but may also mean a timeout.

COMMAND_SOCKET_CONNECTION_CLOSE_FAIL

Failed to close command connection: %1

This error message is issued when an error occurred when closing a command connection and/or removing it from the connections pool. The detailed error is provided as an argument.

COMMAND_SOCKET_CONNECTION_OPENED

Opened socket %1 for incoming command connection

> This is a debug message indicating that a new incoming command connection was detected and a dedicated socket was opened for that connection.

COMMAND_SOCKET_CONNECTION_SHUTDOWN_FAIL

Encountered error %1 while trying to gracefully shutdown socket

> This message indicates an error while trying to gracefully shutdown command connection. The type of the error is included in the message.

COMMAND_SOCKET_CONNECTION_TIMEOUT

Timeout occurred for connection over socket %1

> This is an informational message that indicates that the timeout has occurred for one of the command channel connections. The response sent by the server indicates a timeout and is then closed.

COMMAND_SOCKET_READ

Received %1 bytes over command socket %2

> This debug message indicates that specified number of bytes was received over command socket identified by specified file descriptor.

COMMAND_SOCKET_READ_FAIL

Encountered error %1 while reading from command socket %2

> This error message indicates that an error was encountered while reading from command socket.

COMMAND_SOCKET_WRITE

Sent response of %1 bytes (%2 bytes left to send) over command socket %3

> This debug message indicates that the specified number of bytes was sent over command socket identifier by the specified file descriptor.

COMMAND_SOCKET_WRITE_FAIL

Error while writing to command socket %1 : %2

> This error message indicates that an error was encountered while attempting to send a response to the command socket.

COMMAND_WATCH_SOCKET_CLEAR_ERROR

watch socket failed to clear: %1

> This error message is issued when the command manager was unable to reset the ready status after completing a send. This is a programmatic error that should be reported. The command manager may or may not continue to operate correctly.

COMMAND_WATCH_SOCKET_CLOSE_ERROR

watch socket failed to close: %1

> This error message is issued when command manager attempted to close the socket used for indicating the ready status for send operations. This should not have any negative impact on the operation of the command manager as it happens when the connection is being terminated.

25.5 CTRL

CTRL_AGENT_COMMAND_FORWARDED

command %1 successfully forwarded to the service %2

This informational message is issued when the CA successfully forwards the control message to the specified Kea service and receives a response.

CTRL_AGENT_COMMAND_FORWARD_BEGIN

begin forwarding command %1 to service %2

This debug message is issued when the Control Agent starts forwarding a received command to one of the Kea servers.

CTRL_AGENT_COMMAND_FORWARD_FAILED

failed forwarding command %1: %2

This debug message is issued when the Control Agent failed forwarding a received command to one of the Kea servers. The second argument provides the details of the error.

CTRL_AGENT_CONFIG_CHECK_FAIL

Control Agent configuration check failed: %1

This error message indicates that the CA had failed configuration check. Details are provided. Additional details may be available in earlier log entries, possibly on lower levels.

CTRL_AGENT_CONFIG_FAIL

Control Agent configuration failed: %1

This error message indicates that the CA had failed configuration attempt. Details are provided. Additional details may be available in earlier log entries, possibly on lower levels.

CTRL_AGENT_FAILED

application experienced a fatal error: %1

This is a fatal error message issued when the Control Agent application encounters an unrecoverable error from within the event loop.

CTRL_AGENT_HTTP_SERVICE_STARTED

HTTP service bound to address %1:%2

This informational message indicates that the server has started HTTP service on the specified address and port. All control commands should be sent to this address and port.

CTRL_AGENT_RUN_EXIT

application is exiting the event loop

This is a debug message issued when the Control Agent exits its event loop.

25.6 DATABASE

DATABASE_CQL_CONNECTION_BEGIN_TRANSACTION

begin transaction on current connection.

The server has issued a begin transaction call.

DATABASE_CQL_CONNECTION_COMMIT

committing to Cassandra database on current connection.

A commit call been issued on the server. For Cassandra, this is a no-op.

DATABASE_CQL_CONNECTION_ROLLBACK

rolling back Cassandra database on current connection.

> The code has issued a rollback call. For Cassandra, this is a no-op.

DATABASE_CQL_DEALLOC_ERROR

An error occurred while closing the CQL connection: %1

> This is an error message issued when a DHCP server (either V4 or V6) experienced and error freeing CQL database resources as part of closing its connection to the Cassandra database. The connection is closed as part of normal server shutdown. This error is most likely a programmatic issue that is highly unlikely to occur or negatively impact server operation.

DATABASE_INVALID_ACCESS

invalid database access string: %1

> This is logged when an attempt has been made to parse a database access string and the attempt ended in error. The access string in question - which should be of the form 'keyword=value keyword=value...' is included in the message.

DATABASE_MYSQL_COMMIT

committing to MySQL database

> The code has issued a commit call. All outstanding transactions will be committed to the database. Note that depending on the MySQL settings, the committal may not include a write to disk.

DATABASE_MYSQL_FATAL_ERROR

Unrecoverable MySQL error occurred: %1 for <%2>, reason: %3 (error code: %4).

> An error message indicating that communication with the MySQL database server has been lost. If automatic recovery has been enabled, then the server will attempt to recover connectivity. If not the server wil exit with a non-zero exit code. The cause of such an error is most likely a network issue or the MySQL server has gone down.

DATABASE_MYSQL_ROLLBACK

rolling back MySQL database

> The code has issued a rollback call. All outstanding transaction will be rolled back and not committed to the database.

DATABASE_MYSQL_START_TRANSACTION

starting new MySQL transaction

> A debug message issued when a new MySQL transaction is being started. This message is typically not issued when inserting data into a single table because the server doesn't explicitly start transactions in this case. This message is issued when data is inserted into multiple tables with multiple INSERT statements and there may be a need to rollback the whole transaction if any of these INSERT statements fail.

DATABASE_PGSQL_COMMIT

committing to PostgreSQL database

> The code has issued a commit call. All outstanding transactions will be committed to the database. Note that depending on the PostgreSQL settings, the committal may not include a write to disk.

DATABASE_PGSQL_DEALLOC_ERROR

An error occurred deallocating SQL statements while closing the PostgreSQL lease database: %1

This is an error message issued when a DHCP server (either V4 or V6) experienced and error freeing database SQL resources as part of closing its connection to the PostgreSQL database. The connection is closed as part of normal server shutdown. This error is most likely a programmatic issue that is highly unlikely to occur or negatively impact server operation.

DATABASE_PGSQL_FATAL_ERROR

Unrecoverable PostgreSQL error occurred: Statement: <%1>, reason: %2 (error code: %3).

An error message indicating that communication with the PostgreSQL database server has been lost. If automatic recovery has been enabled, then the server will attempt to recover the connectivity. If not the server wil exit with a non-zero exit code. The cause of such an error is most likely a network issue or the PostgreSQL server has gone down.

DATABASE_PGSQL_ROLLBACK

rolling back PostgreSQL database

The code has issued a rollback call. All outstanding transaction will be rolled back and not committed to the database.

DATABASE_PGSQL_START_TRANSACTION

starting a new PostgreSQL transaction

A debug message issued when a new PostgreSQL transaction is being started. This message is typically not issued when inserting data into a single table because the server doesn't explicitly start transactions in this case. This message is issued when data is inserted into multiple tables with multiple INSERT statements and there may be a need to rollback the whole transaction if any of these INSERT statements fail.

25.7 DCTL

DCTL_ALREADY_RUNNING

%1 already running? %2

This is an error message that occurs when a module encounters a pre-existing PID file which contains the PID of a running process. This most likely indicates an attempt to start a second instance of a module using the same configuration file. It is possible, though unlikely, that the PID file is a remnant left behind by a server crash or power failure and the PID it contains refers to a process other than Kea process. In such an event, it would be necessary to manually remove the PID file. The first argument is the process name, the second contains the PID and PID file.

DCTL_CCSESSION_ENDING

%1 ending control channel session

This debug message is issued just before the controller attempts to disconnect from its session with the Kea control channel.

DCTL_CFG_FILE_RELOAD_ERROR

configuration reload failed: %1, reverting to current configuration.

This is an error message indicating that the application attempted to reload its configuration from file and encountered an error. This is likely due to invalid content in the configuration file. The application should continue to operate under its current configuration.

DCTL_CFG_FILE_RELOAD_SIGNAL_RECVD

OS signal %1 received, reloading configuration from file: %2

This is an informational message indicating the application has received a signal instructing it to reload its configuration from file.

DCTL_COMMAND_RECEIVED

%1 received command: %2, arguments: %3

A debug message listing the command (and possible arguments) received from the Kea control system by the controller.

DCTL_CONFIG_CHECK_COMPLETE

server has completed configuration check: %1, result: %2

This is an informational message announcing the successful processing of a new configuration check is complete. The result of that check is printed. This informational message is printed when configuration check is requested.

DCTL_CONFIG_COMPLETE

server has completed configuration: %1

This is an informational message announcing the successful processing of a new configuration. It is output during server startup, and when an updated configuration is committed by the administrator. Additional information may be provided.

DCTL_CONFIG_DEPRECATED

server configuration includes a deprecated object: %1

This error message is issued when the configuration includes a deprecated object (i.e. a top level element) which will be ignored.

DCTL_CONFIG_FETCH

Fetching configuration data from config backends.

This is an informational message emitted when the Kea server is about to begin retrieving configuration data from one or more configuration backends.

DCTL_CONFIG_FILE_LOAD_FAIL

%1 reason: %2

This fatal error message indicates that the application attempted to load its initial configuration from file and has failed. The service will exit.

DCTL_CONFIG_LOAD_FAIL

%1 configuration failed to load: %2

This critical error message indicates that the initial application configuration has failed. The service will start, but will not process requests until the configuration has been corrected.

DCTL_CONFIG_START

parsing new configuration: %1

A debug message indicating that the application process has received an updated configuration and has passed it to its configuration manager for parsing.

DCTL_CONFIG_STUB

%1 configuration stub handler called

This debug message is issued when the dummy handler for configuration events is called. This only happens during initial startup.

DCTL_CONFIG_UPDATE

%1 updated configuration received: %2

> A debug message indicating that the controller has received an updated configuration from the Kea configuration system.

DCTL_DEVELOPMENT_VERSION

This software is a development branch of Kea. It is not recommended for production use.

> This warning message is displayed when the version is a development (vs stable) one: the second number of the version is odd.

DCTL_INIT_PROCESS

%1 initializing the application

> This debug message is issued just before the controller attempts to create and initialize its application instance.

DCTL_INIT_PROCESS_FAIL

%1 application initialization failed: %2

> This error message is issued if the controller could not initialize the application and will exit.

DCTL_NOT_RUNNING

%1 application instance is not running

> A warning message is issued when an attempt is made to shut down the application when it is not running.

DCTL_OPEN_CONFIG_DB

Opening configuration database: %1

> This message is printed when the Kea server is attempting to open a configuration database. The database access string with password redacted is logged.

DCTL_PARSER_FAIL

: %1

> On receipt of a new configuration, the server failed to create a parser to decode the contents of the named configuration element, or the creation succeeded but the parsing actions and committal of changes failed. The reason for the failure is given in the message.

DCTL_PID_FILE_ERROR

%1 could not create a PID file: %2

> This is an error message that occurs when the server is unable to create its PID file. The log message should contain details sufficient to determine the underlying cause. The most likely culprits are that some portion of the pathname does not exist or a permissions issue. The default path is determined by –localstatedir or –runstatedir configure parameters but may be overridden by setting environment variable, KEA_PIDFILE_DIR. The first argument is the process name.

DCTL_PROCESS_FAILED

%1 application execution failed: %2

> The controller has encountered a fatal error while running the application and is terminating. The reason for the failure is included in the message.

DCTL_RUN_PROCESS

%1 starting application event loop

This debug message is issued just before the controller invokes the application run method.

DCTL_SESSION_FAIL

%1 controller failed to establish Kea session: %1

The controller has failed to establish communication with the rest of Kea and will exit.

DCTL_SHUTDOWN

%1 has shut down, pid: %2, version: %3

This is an informational message indicating that the service has shut down. The argument specifies a name of the service.

DCTL_SHUTDOWN_SIGNAL_RECVD

OS signal %1 received, starting shutdown

This is a debug message indicating the application has received a signal instructing it to shutdown.

DCTL_SIGNAL_ERROR

signal handler for signal %1, threw an unexpected exception: %2

This is an error message indicating that the application encountered an unexpected error after receiving a signal. This is a programmatic error and should be reported. While The application will likely continue to operating, it may be unable to respond correctly to signals.

DCTL_STANDALONE

%1 skipping message queue, running standalone

This is a debug message indicating that the controller is running in the application in standalone mode. This means it will not connected to the Kea message queue. Standalone mode is only useful during program development, and should not be used in a production environment.

DCTL_STARTING

%1 starting, pid: %2, version: %3 (%4)

This is an informational message issued when controller for the service first starts. Version is also reported.

25.8 DHCP4

DHCP4_ACTIVATE_INTERFACE

activating interface %1

This message is printed when DHCPv4 server enabled an interface to be used to receive DHCPv4 traffic. IPv4 socket on this interface will be opened once Interface Manager starts up procedure of opening sockets.

DHCP4_ALREADY_RUNNING

%1 already running? %2

This is an error message that occurs when the DHCPv4 server encounters a pre-existing PID file which contains the PID of a running process. This most likely indicates an attempt to start a second instance of the server using the same configuration file. It is possible, though unlikely that the PID file is a remnant left behind by a server crash or power failure and the PID it contains refers to a process other than the server. In such an event, it would be necessary to manually remove the PID file. The first argument is the DHCPv4 process name, the second contains the PID and PID file.

DHCP4_BUFFER_RECEIVED

received buffer from %1:%2 to %3:%4 over interface %5

> This debug message is logged when the server has received a packet over the socket. When the message is logged the contents of the received packet hasn't been parsed yet. The only available information is the interface and the source and destination IPv4 addresses/ports.

DHCP4_BUFFER_RECEIVE_FAIL

error on attempt to receive packet: %1

> The DHCPv4 server tried to receive a packet but an error occurred during this attempt. The reason for the error is included in the message.

DHCP4_BUFFER_UNPACK

parsing buffer received from %1 to %2 over interface %3

> This debug message is issued when the server starts parsing the received buffer holding the DHCPv4 message. The arguments specify the source and destination IPv4 addresses as well as the interface over which the buffer has been received.

DHCP4_BUFFER_WAIT_SIGNAL

signal received while waiting for next packet, next waiting signal is %1

> This debug message is issued when the server was waiting for the packet, but the wait has been interrupted by the signal received by the process. The signal will be handled before the server starts waiting for next packets. The argument specifies the next signal to be handled by the server.

DHCP4_CB_ON_DEMAND_FETCH_UPDATES_FAIL

error on demand attempt to fetch configuration updates from the configuration backend(s): %1

> This error message is issued when the server attempted to fetch configuration updates from the database and this on demand attempt failed. The sole argument which is returned to the config-backend-pull command caller too contains the reason for failure.

DHCP4_CB_PERIODIC_FETCH_UPDATES_FAIL

error on periodic attempt to fetch configuration updates from the configuration backend(s): %1

> This error message is issued when the server attempted to fetch configuration updates from the database and this periodic attempt failed. The server will re-try according to the configured value of the config-fetch-wait-time parameter. The sole argument contains the reason for failure.

DHCP4_CB_PERIODIC_FETCH_UPDATES_RETRIES_EXHAUSTED

maximum number of configuration fetch attempts: 10, has been exhausted without success

> This error indicates that the server has made a number of unsuccessful periodic attempts to fetch configuration updates from a configuration backend. The server will continue to operate but won't make any further attempts to fetch configuration updates. The administrator must fix the configuration in the database and reload (or restart) the server.

DHCP4_CLASS_ASSIGNED

%1: client packet has been assigned to the following class(es): %2

> This debug message informs that incoming packet has been assigned to specified class or classes. This is a normal behavior and indicates successful operation. The first argument specifies the client and transaction identification information. The second argument includes all classes to which the packet has been assigned.

DHCP4_CLASS_UNCONFIGURED

%1: client packet belongs to an unconfigured class: %2

> This debug message informs that incoming packet belongs to a class which cannot be found in the configuration. Either a hook written before the classification was added to Kea is used, or class naming is inconsistent.

DHCP4_CLASS_UNDEFINED

required class %1 has no definition

> This debug message informs that a class is listed for required evaluation but has no definition.

DHCP4_CLASS_UNTESTABLE

required class %1 has no test expression

> This debug message informs that a class was listed for required evaluation but its definition does not include a test expression to evaluate.

DHCP4_CLIENTID_IGNORED_FOR_LEASES

%1: not using client identifier for lease allocation for subnet %2

> This debug message is issued when the server is processing the DHCPv4 message for which client identifier will not be used when allocating new lease or renewing existing lease. The server is explicitly configured to not use client identifier to lookup existing leases for the client and will not record client identifier in the lease database. This mode of operation is useful when clients don't use stable client identifiers, e.g. multi stage booting. The first argument includes the client and transaction identification information. The second argument specifies the identifier of the subnet where the client is connected and for which this mode of operation is configured on the server.

DHCP4_CLIENT_FQDN_DATA

%1: Client sent FQDN option: %2

> This debug message includes the detailed information extracted from the Client FQDN option sent in the query. The first argument includes the client and transaction identification information. The second argument specifies the detailed information about the FQDN option received by the server.

DHCP4_CLIENT_FQDN_PROCESS

%1: processing Client FQDN option

> This debug message is issued when the server starts processing the Client FQDN option sent in the client's query. The argument includes the client and transaction identification information.

DHCP4_CLIENT_HOSTNAME_DATA

%1: client sent Hostname option: %2

> This debug message includes the detailed information extracted from the Hostname option sent in the query. The first argument includes the client and transaction identification information. The second argument specifies the hostname carried in the Hostname option sent by the client.

DHCP4_CLIENT_HOSTNAME_MALFORMED

%1: client hostname option malformed: %2

> This debug message is issued when the DHCP server was unable to process the the hostname option sent by the client because the content is malformed. The first argument includes the client and transaction identification information. The second argument contains a description of the data error.

DHCP4_CLIENT_HOSTNAME_PROCESS

%1: processing client's Hostname option

> This debug message is issued when the server starts processing the Hostname option sent in the client's query. The argument includes the client and transaction identification information.

DHCP4_CLIENT_NAME_PROC_FAIL

%1: failed to process the fqdn or hostname sent by a client: %2

> This debug message is issued when the DHCP server was unable to process the FQDN or Hostname option sent by a client. This is likely because the client's name was malformed or due to internal server error. The first argument contains the client and transaction identification information. The second argument holds the detailed description of the error.

DHCP4_COMMAND_RECEIVED

received command %1, arguments: %2

> A debug message listing the command (and possible arguments) received from the Kea control system by the DHCPv4 server.

DHCP4_CONFIG_COMPLETE

DHCPv4 server has completed configuration: %1

> This is an informational message announcing the successful processing of a new configuration. It is output during server startup, and when an updated configuration is committed by the administrator. Additional information may be provided.

DHCP4_CONFIG_FETCH

Fetching configuration data from config backends.

> This is an informational message emitted when the DHCPv4 server about to begin retrieving configuration data from one or more configuration backends.

DHCP4_CONFIG_LOAD_FAIL

configuration error using file: %1, reason: %2

> This error message indicates that the DHCPv4 configuration has failed. If this is an initial configuration (during server's startup) the server will fail to start. If this is a dynamic reconfiguration attempt the server will continue to use an old configuration.

DHCP4_CONFIG_NEW_SUBNET

a new subnet has been added to configuration: %1

> This is an informational message reporting that the configuration has been extended to include the specified IPv4 subnet.

DHCP4_CONFIG_OPTION_DUPLICATE

multiple options with the code %1 added to the subnet %2

> This warning message is issued on an attempt to configure multiple options with the same option code for a particular subnet. Adding multiple options is uncommon for DHCPv4, but is not prohibited.

DHCP4_CONFIG_PACKET_QUEUE

DHCPv4 packet queue info after configuration: %1

> This informational message is emitted during DHCPv4 server configuration, immediately after configuring the DHCPv4 packet queue. The information shown depends upon the packet queue type selected.

DHCP4_CONFIG_RECEIVED

received configuration %1

> A debug message listing the configuration received by the DHCPv4 server. The source of that configuration depends on used configuration backend.

DHCP4_CONFIG_START

DHCPv4 server is processing the following configuration: %1

> This is a debug message that is issued every time the server receives a configuration. That happens at start up and also when a server configuration change is committed by the administrator.

DHCP4_CONFIG_UNRECOVERABLE_ERROR

DHCPv4 server new configuration failed with an error which cannot be recovered

> This fatal error message is issued when a new configuration raised an error which cannot be recovered. A correct configuration must be applied as soon as possible as the server is no longer working. The configuration can be fixed in several ways. If the control channel is open, config-set with a valid configuration can be used. Alternatively, the original config file on disk could be fixed and SIGHUP signal could be sent (or the config-reload command issued). Finally, the server could be restarted completely.

DHCP4_CONFIG_UNSUPPORTED_OBJECT

DHCPv4 server configuration includes an unsupported object: %1

> This error message is issued when the configuration includes an unsupported object (i.e. a top level element).

DHCP4_CONFIG_UPDATE

updated configuration received: %1

> A debug message indicating that the DHCPv4 server has received an updated configuration from the Kea configuration system.

DHCP4_DB_RECONNECT_ATTEMPT_FAILED

database reconnect failed: %1

> An error message indicating that an attempt to reconnect to the lease and/or host data bases has failed. This occurs after connectivity to either one has been lost and an automatic attempt to reconnect has failed.

DHCP4_DB_RECONNECT_ATTEMPT_SCHEDULE

scheduling attempt %1 of %2 in %3 milliseconds

> An informational message indicating that the server is scheduling the next attempt to reconnect to its lease and/or host databases. This occurs when the server has lost databse connectivity and is attempting to reconnect automatically.

DHCP4_DB_RECONNECT_DISABLED

database reconnect is disabled: max-reconnect-tries %1, reconnect-wait-time %2

> This is an informational message indicating that connectivity to either the lease or host database or both and that automatic reconnect is not enabled.

DHCP4_DB_RECONNECT_NO_DB_CTL

unexpected error in database reconnect

> This is an error message indicating a programmatic error that should not occur. It will prohibit the server from attempting to reconnect to its databases if connectivy is lost, and the server will exit. This error should be reported.

DHCP4_DB_RECONNECT_RETRIES_EXHAUSTED

maximum number of database reconnect attempts: %1, has been exhausted without success, server is shutting down!

> This error indicates that the server is shutting down after failing to reconnect to the lease and/or host database(s) after making the maximum configured number of reconnect attempts. Loss of connectivity is typically a network or database server issue.

DHCP4_DDNS_REQUEST_SEND_FAILED

failed sending a request to kea-dhcp-ddns, error: %1, ncr: %2

> This error message indicates that DHCP4 server attempted to send a DDNS update request to the DHCP-DDNS server. This is most likely a configuration or networking error.

DHCP4_DEACTIVATE_INTERFACE

deactivate interface %1

> This message is printed when DHCPv4 server disables an interface from being used to receive DHCPv4 traffic. Sockets on this interface will not be opened by the Interface Manager until interface is enabled.

DHCP4_DECLINE_FAIL

%1: error on decline lease for address %2: %3

> This error message indicates that the software failed to deline a lease from the lease database due to an error during a database operation. The first argument includes the client and the transaction identification information. The second argument holds the IPv4 address which decline was attempted. The last one contains the reason for failure.

DHCP4_DECLINE_LEASE

Received DHCPDECLINE for addr %1 from client %2. The lease will be unavailable for %3 seconds.

> This informational message is printed when a client received an address, but discovered that it is being used by some other device and notified the server by sending a DHCPDECLINE message. The server checked that this address really was leased to the client and marked this address as unusable for a certain amount of time. This message may indicate a misconfiguration in a network, as there is either a buggy client or more likely a device that is using an address that it is not supposed to. The server will fully recover from this situation, but if the underlying problem of a misconfigured or rogue device is not solved, this address may be declined again in the future.

DHCP4_DECLINE_LEASE_MISMATCH

Received DHCPDECLINE for addr %1 from client %2, but the data doesn't match: received hwaddr: %3, lease hwaddr: %4, received client-id: %5, lease client-id: %6

> This informational message means that a client attempted to report his address as declined (i.e. used by unknown entity). The server has information about a lease for that address, but the client's hardware address or client identifier does not match the server's stored information. The client's request will be ignored.

DHCP4_DECLINE_LEASE_NOT_FOUND

Received DHCPDECLINE for addr %1 from client %2, but no such lease found.

> This warning message indicates that a client reported that his address was detected as a duplicate (i.e. another device in the network is using this address). However, the server does not have a record for this address. This may indicate a client's error or a server's purged database.

DHCP4_DEFERRED_OPTION_MISSING

can find deferred option code %1 in the query

This debug message is printed when a deferred option cannot be found in the query.

DHCP4_DEFERRED_OPTION_UNPACK_FAIL

An error unpacking the deferred option %1: %2

A debug message issued when deferred unpacking of an option failed, making it to be left unpacked in the packet. The first argument is the option code, the second the error.

DHCP4_DEVELOPMENT_VERSION

This software is a development branch of Kea. It is not recommended for production use.

This warning message is displayed when the version is a development (vs stable) one: the second number of the version is odd.

DHCP4_DHCP4O6_BAD_PACKET

received malformed DHCPv4o6 packet: %1

A malformed DHCPv4o6 packet was received.

DHCP4_DHCP4O6_PACKET_RECEIVED

received DHCPv4o6 packet from DHCPv4 server (type %1) for %2 on interface %3

This debug message is printed when the server is receiving a DHCPv4o6 from the DHCPv4 server over inter-process communication.

DHCP4_DHCP4O6_PACKET_SEND

%1: trying to send packet %2 (type %3) to %4 port %5 on interface %6 encapsulating %7: %8 (type %9)

The arguments specify the client identification information (HW address and client identifier), DHCPv6 message name and type, source IPv6 address and port, and interface name, DHCPv4 client identification, message name and type.

DHCP4_DHCP4O6_PACKET_SEND_FAIL

%1: failed to send DHCPv4o6 packet: %2

This error is output if the IPv4 DHCP server fails to send an DHCPv4o6 message to the IPv6 DHCP server. The reason for the error is included in the message.

DHCP4_DHCP4O6_RECEIVE_FAIL

failed to receive DHCPv4o6: %1

This debug message indicates the inter-process communication with the DHCPv6 server failed. The reason for the error is included in the message.

DHCP4_DHCP4O6_RECEIVING

receiving DHCPv4o6 packet from DHCPv6 server

This debug message is printed when the server is receiving a DHCPv4o6 from the DHCPv6 server over inter-process communication socket.

DHCP4_DHCP4O6_RESPONSE_DATA

%1: responding with packet %2 (type %3), packet details: %4

A debug message including the detailed data about the packet being sent to the DHCPv6 server to be forwarded to the client. The first argument contains the client and the transaction identification information. The second and third argument contains the packet name and type respectively. The fourth argument contains detailed packet information.

DHCP4_DYNAMIC_RECONFIGURATION

initiate server reconfiguration using file: %1, after receiving SIGHUP signal or config-reload command

> This is the info message logged when the DHCPv4 server starts reconfiguration as a result of receiving SIGHUP signal or config-reload command.

DHCP4_DYNAMIC_RECONFIGURATION_FAIL

dynamic server reconfiguration failed with file: %1

> This is a fatal error message logged when the dynamic reconfiguration of the DHCP server failed.

DHCP4_EMPTY_HOSTNAME

%1: received empty hostname from the client, skipping processing of this option

> This debug message is issued when the server received an empty Hostname option from a client. Server does not process empty Hostname options and therefore option is skipped. The argument holds the client and transaction identification information.

DHCP4_FLEX_ID

flexible identifier generated for incoming packet: %1

> This debug message is printed when host reservation type is set to flexible identifier and the expression specified in its configuration generated (was evaluated to) an identifier for incoming packet. This debug message is mainly intended as a debugging assistance for flexible identifier.

DHCP4_GENERATE_FQDN

%1: client did not send a FQDN or hostname; FQDN will be be generated for the client

> This debug message is issued when the server did not receive a Hostname option from the client and hostname generation is enabled. This provides a means to create DNS entries for unsophisticated clients.

DHCP4_HANDLE_SIGNAL_EXCEPTION

An exception was thrown while handing signal: %1

> This error message is printed when an ISC or standard exception was raised during signal processing. This likely indicates a coding error and should be reported to ISC.

DHCP4_HOOKS_LIBS_RELOAD_FAIL

reload of hooks libraries failed

> A "libreload" command was issued to reload the hooks libraries but for some reason the reload failed. Other error messages issued from the hooks framework will indicate the nature of the problem.

DHCP4_HOOK_BUFFER_RCVD_DROP

received buffer from %1 to %2 over interface %3 was dropped because a callout set the drop flag

> This debug message is printed when a callout installed on buffer4_receive hook point set the drop flag. For this particular hook point, the setting of the flag by a callout instructs the server to drop the packet. The arguments specify the source and destination IPv4 address as well as the name of the interface over which the buffer has been received.

DHCP4_HOOK_BUFFER_RCVD_SKIP

received buffer from %1 to %2 over interface %3 is not parsed because a callout set the next step to SKIP.

> This debug message is printed when a callout installed on buffer4_receive hook point set the next step to SKIP. For this particular hook point, this value set by a callout instructs the server to not parse the buffer because it was already parsed by the hook. The arguments specify the source and destination IPv4 address as well as the name of the interface over which the buffer has been received.

DHCP4_HOOK_BUFFER_SEND_SKIP

%1: prepared response is dropped because a callout set the next step to SKIP.

> This debug message is printed when a callout installed on buffer4_send hook point set the next step to SKIP. For this particular hook point, the SKIP value set by a callout instructs the server to drop the packet. Server completed all the processing (e.g. may have assigned, updated or released leases), but the response will not be send to the client.

DHCP4_HOOK_DECLINE_SKIP

Decline4 hook callouts set status to DROP, ignoring packet.

> This message indicates that the server received DHCPDECLINE message, it was verified to be correct and matching server's lease information. The server called hooks for decline4 hook point and one of the callouts set next step status to DROP. The server will now abort processing of the packet as if it was never received. The lease will continue to be assigned to this client.

DHCP4_HOOK_LEASE4_RELEASE_SKIP

%1: lease was not released because a callout set the next step to SKIP

> This debug message is printed when a callout installed on lease4_release hook point set the next step status to SKIP. For this particular hook point, the value set by a callout instructs the server to not release a lease.

DHCP4_HOOK_LEASES4_COMMITTED_DROP

%1: packet is dropped, because a callout set the next step to DROP

> This debug message is printed when a callout installed on the leases4_committed hook point sets the next step to DROP.

DHCP4_HOOK_LEASES4_COMMITTED_PARK

%1: packet is parked, because a callout set the next step to PARK

> This debug message is printed when a callout installed on the lease4_committed hook point sets the next step to PARK.

DHCP4_HOOK_PACKET_RCVD_SKIP

%1: packet is dropped, because a callout set the next step to SKIP

> This debug message is printed when a callout installed on the pkt4_receive hook point sets the next step to SKIP. For this particular hook point, the value setting of the flag instructs the server to drop the packet.

DHCP4_HOOK_PACKET_SEND_DROP

%1: prepared DHCPv4 response was not sent because a callout set the next ste to DROP

> This debug message is printed when a callout installed on the pkt4_send hook point set the next step to DROP. For this particular hook point, the setting of the value by a callout instructs the server to drop the packet. This effectively means that the client will not get any response, even though the server processed client's request and acted on it (e.g. possibly allocated a lease). The argument specifies the client and transaction identification information.

DHCP4_HOOK_PACKET_SEND_SKIP

%1: prepared response is not sent, because a callout set the next stp to SKIP

> This debug message is printed when a callout installed on the pkt4_send hook point sets the next step to SKIP. For this particular hook point, this setting instructs the server to drop the packet. This means that the client will not get any response, even though the server processed client's request and acted on it (e.g. possibly allocated a lease).

DHCP4_HOOK_SUBNET4_SELECT_DROP

%1: packet was dropped, because a callout set the next step to 'drop'

> This debug message is printed when a callout installed on the subnet4_select hook point sets the next step to 'drop' value. For this particular hook point, the setting to that value instructs the server to drop the received packet. The argument specifies the client and transaction identification information.

DHCP4_HOOK_SUBNET4_SELECT_SKIP

%1: no subnet was selected, because a callout set the next skip flag

> This debug message is printed when a callout installed on the subnet4_select hook point sets the next step to SKIP value. For this particular hook point, the setting of the flag instructs the server not to choose a subnet, an action that severely limits further processing; the server will be only able to offer global options - no addresses will be assigned. The argument specifies the client and transaction identification information.

DHCP4_INFORM_DIRECT_REPLY

%1: DHCPACK in reply to the DHCPINFORM will be sent directly to %2 over %3

> This debug message is issued when the DHCPACK will be sent directly to the client, rather than via a relay. The first argument contains the client and transaction identification information. The second argument contains the client's IPv4 address to which the response will be sent. The third argument contains the local interface name.

DHCP4_INIT_FAIL

failed to initialize Kea server: %1

> The server has failed to initialize. This may be because the configuration was not successful, or it encountered any other critical error on startup. Attached error message provides more details about the issue.

DHCP4_INIT_REBOOT

%1: client is in INIT-REBOOT state and requests address %2

> This informational message is issued when the client is in the INIT-REBOOT state and is requesting an IPv4 address it is using to be allocated for it. The first argument includes the client and transaction identification information. The second argument specifies the requested IPv4 address.

DHCP4_LEASE_ADVERT

%1: lease %2 will be advertised

> This informational message indicates that the server has found the lease to be offered to the client. It is up to the client to choose one server out of those which offered leases and continue allocation with that server. The first argument specifies the client and the transaction identification information. The second argument specifies the IPv4 address to be offered.

DHCP4_LEASE_ALLOC

%1: lease %2 has been allocated for %3 seconds

> This informational message indicates that the server successfully granted a lease in response to client's DHCPREQUEST message. The lease information will be sent to the client in the DHCPACK message. The first argument contains the client and the transaction identification information. The second argument contains the allocated IPv4 address. The third argument is the validity lifetime.

DHCP4_MULTI_THREADING_INFO

enabled: %1, number of threads: %2, queue size: %3

This is a message listing some information about the multi-threading parameters with which the server is running.

DHCP4_NCR_CREATE

%1: DDNS updates enabled, therefore sending name change requests

This debug message is issued when the server is starting to send name change requests to the D2 module to update records for the client in the DNS. This includes removal of old records and addition of the new records as required. Details of the name change requests will be logged in additional log entries. The argument includes the client and the transaction identification information.

DHCP4_NCR_CREATION_FAILED

%1: failed to generate name change requests for DNS: %2

This message indicates that server was unable to generate NameChangeRequests which should be sent to the kea-dhcp_ddns module to create new DNS records for the lease being acquired or to update existing records for the renewed lease. The first argument contains the client and transaction identification information. The second argument includes the reason for the failure.

DHCP4_NOT_RUNNING

DHCPv4 server is not running

A warning message is issued when an attempt is made to shut down the DHCPv4 server but it is not running.

DHCP4_NO_LEASE_INIT_REBOOT

%1: no lease for address %2 requested by INIT-REBOOT client

This debug message is issued when the client being in the INIT-REBOOT state requested an IPv4 address but this client is unknown. The server will not respond. The first argument includes the client and the transaction id identification information. The second argument includes the IPv4 address requested by the client.

DHCP4_NO_SOCKETS_OPEN

no interface configured to listen to DHCP traffic

This warning message is issued when current server configuration specifies no interfaces that server should listen on, or specified interfaces are not configured to receive the traffic.

DHCP4_OPEN_CONFIG_DB

Opening configuration database: %1

This message is printed when the DHCPv4 server is attempting to open a configuration database. The database access string with password redacted is logged.

DHCP4_OPEN_SOCKET

opening service sockets on port %1

A debug message issued during startup, this indicates that the DHCPv4 server is about to open sockets on the specified port.

DHCP4_OPEN_SOCKET_FAIL

failed to open socket: %1

A warning message issued when IfaceMgr fails to open and bind a socket. The reason for the failure is appended as an argument of the log message.

DHCP4_PACKET_DROP_0001

failed to parse packet from %1 to %2, received over interface %3, reason: %4

> The DHCPv4 server has received a packet that it is unable to interpret. The reason why the packet is invalid is included in the message.

DHCP4_PACKET_DROP_0002

%1, from interface %2: no suitable subnet configured for a direct client

> This info message is logged when received a message from a directly connected client but there is no suitable subnet configured for the interface on which this message has been received. The IPv4 address assigned on this interface must belong to one of the configured subnets. Otherwise received message is dropped.

DHCP4_PACKET_DROP_0003

%1, from interface %2: it contains a foreign server identifier

> This debug message is issued when received DHCPv4 message is dropped because it is addressed to a different server, i.e. a server identifier held by this message doesn't match the identifier used by our server. The arguments of this message hold the name of the transaction id and interface on which the message has been received.

DHCP4_PACKET_DROP_0004

%1, from interface %2: missing msg-type option

> This is a debug message informing that incoming DHCPv4 packet did not have mandatory DHCP message type option and thus was dropped. The arguments specify the client and transaction identification information, as well as the interface on which the message has been received.

DHCP4_PACKET_DROP_0005

%1: unrecognized type %2 in option 53

> This debug message indicates that the message type carried in DHCPv4 option 53 is unrecognized by the server. The valid message types are listed on the IANA website: http://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xhtml#message-type-53. The message will not be processed by the server. The arguments specify the client and transaction identification information, as well as the received message type.

DHCP4_PACKET_DROP_0006

%1: unsupported DHCPv4 message type %2

> This debug message indicates that the message type carried in DHCPv4 option 53 is valid but the message will not be processed by the server. This includes messages being normally sent by the server to the client, such as DHCPOFFER, DHCPACK, DHCPNAK etc. The first argument specifies the client and transaction identification information. The second argument specifies the message type.

DHCP4_PACKET_DROP_0007

%1: failed to process packet: %2

> This is a general catch-all message indicating that the processing of a received packet failed. The reason is given in the message. The server will not send a response but will instead ignore the packet. The first argument contains the client and transaction identification information. The second argument includes the details of the error.

DHCP4_PACKET_DROP_0008

%1: DHCP service is globally disabled

This debug message is issued when a packet is dropped because the DHCP service has been temporarily disabled. This affects all received DHCP packets. The service may be enabled by the "dhcp-enable" control command or automatically after a specified amount of time since receiving "dhcp-disable" command.

DHCP4_PACKET_DROP_0009

%1: Option 53 missing (no DHCP message type), is this a BOOTP packet?

> This debug message is issued when a packet is dropped because it did contain option 53 and thus has no DHCP message type. The most likely explanation is that it was BOOTP packet.

DHCP4_PACKET_DROP_0010

dropped as member of the special class 'DROP': %1

> This debug message is emitted when an incoming packet was classified into the special class 'DROP' and dropped. The packet details are displayed.

DHCP4_PACKET_DROP_0011

dropped as sent by the same client than a packet being processed by another thread: dropped %1 by thread %2 as duplicate of %3 processed by %4

> Currently multi-threading processing avoids races between packets sent by a client using the same client id option by dropping new packets until processing is finished. Packet details and thread identifiers are included for both packets in this warning message.

DHCP4_PACKET_DROP_0012

dropped as sent by the same client than a packet being processed by another thread: dropped %1 by thread %2 as duplicate of %3 processed by %4

> Currently multi-threading processing avoids races between packets sent by a client using the same hardware address by dropping new packets until processing is finished. Packet details and thread identifiers are included for both packets in this warning message.

DHCP4_PACKET_NAK_0001

%1: failed to select a subnet for incoming packet, src %2, type %3

> This error message is output when a packet was received from a subnet for which the DHCPv4 server has not been configured. The most probable cause is a misconfiguration of the server. The first argument contains the client and transaction identification information. The second argument contains the source IPv4 address of the packet. The third argument contains the name of the received packet.

DHCP4_PACKET_NAK_0002

%1: invalid address %2 requested by INIT-REBOOT

> This debug message is issued when the client being in the INIT-REBOOT state requested an IPv4 address which is not assigned to him. The server will respond to this client with DHCPNAK. The first argument contains the client and the transaction identification information. The second arguments holds the IPv4 address requested by the client.

DHCP4_PACKET_NAK_0003

%1: failed to advertise a lease, client sent ciaddr %2, requested-ip-address %3

> This message indicates that the server has failed to offer a lease to the specified client after receiving a DISCOVER message from it. There are many possible reasons for such a failure. The first argument contains the client and the transaction identification information. The second argument contains the IPv4 address in the ciaddr field. The third argument contains the IPv4 address in the requested-ip-address option (if present).

DHCP4_PACKET_NAK_0004

%1: failed to grant a lease, client sent ciaddr %2, requested-ip-address %3

> This message indicates that the server failed to grant a lease to the specified client after receiving a REQUEST message from it. There are many possible reasons for such a failure. Additional messages will indicate the reason. The first argument contains the client and the transaction identification information. The second argument contains the IPv4 address in the ciaddr field. The third argument contains the IPv4 address in the requested-ip-address option (if present).

DHCP4_PACKET_OPTIONS_SKIPPED

An error unpacking an option, caused subsequent options to be skipped: %1

> A debug message issued when an option failed to unpack correctly, making it impossible to unpack the remaining options in the packet. The server will server will still attempt to service the packet.

DHCP4_PACKET_OPTION_UNPACK_FAIL

An error unpacking the option %1: %2

> A debug message issued when an option failed to unpack correctly, making it to be left unpacked in the packet. The first argument is the option code, the second the error.

DHCP4_PACKET_PACK

%1: preparing on-wire format of the packet to be sent

> This debug message is issued when the server starts preparing the on-wire format of the packet to be sent back to the client. The argument specifies the client and the transaction identification information.

DHCP4_PACKET_PACK_FAIL

%1: preparing on-wire-format of the packet to be sent failed %2

> This error message is issued when preparing an on-wire format of the packet has failed. The first argument identifies the client and the DHCP transaction. The second argument includes the error string.

DHCP4_PACKET_PROCESS_EXCEPTION

exception occurred during packet processing

> This error message indicates that a non-standard exception was raised during packet processing that was not caught by other, more specific exception handlers. This packet will be dropped and the server will continue operation.

DHCP4_PACKET_PROCESS_STD_EXCEPTION

exception occurred during packet processing: %1

> This error message indicates that a standard exception was raised during packet processing that was not caught by other, more specific exception handlers. This packet will be dropped and the server will continue operation.

DHCP4_PACKET_QUEUE_FULL

multi-threading packet queue is full

> A debug message noting that the multi-threading packet queue is full so the oldest packet of the queue was dropped to make room for the received one.

DHCP4_PACKET_RECEIVED

%1: %2 (type %3) received from %4 to %5 on interface %6

A debug message noting that the server has received the specified type of packet on the specified interface. The first argument specifies the client and transaction identification information. The second and third argument specify the name of the DHCPv4 message and its numeric type respectively. The remaining arguments specify the source IPv4 address, destination IPv4 address and the name of the interface on which the message has been received.

DHCP4_PACKET_SEND

%1: trying to send packet %2 (type %3) from %4:%5 to %6:%7 on interface %8

The arguments specify the client identification information (HW address and client identifier), DHCP message name and type, source IPv4 address and port, destination IPv4 address and port and the interface name. This debug message is issued when the server is trying to send the response to the client. When the server is using an UDP socket to send the packet there are cases when this operation may be unsuccessful and no error message will be displayed. One such situation occurs when the server is unicasting the response to the 'ciaddr' of a DHCPINFORM message. This often requires broadcasting an ARP message to obtain the link layer address of the unicast destination. If broadcast ARP messages are blocked in the network, according to the firewall policy, the ARP message will not cause a response. Consequently, the response to the DHCPINFORM will not be sent. Since the ARP communication is under the OS control, Kea is not notified about the drop of the packet which it is trying to send and it has no means to display an error message.

DHCP4_PACKET_SEND_FAIL

%1: failed to send DHCPv4 packet: %2

This error is output if the DHCPv4 server fails to send an assembled DHCP message to a client. The first argument includes the client and the transaction identification information. The second argument includes the reason for failure.

DHCP4_PARSER_COMMIT_EXCEPTION

parser failed to commit changes

On receipt of message containing details to a change of the DHCPv4 server configuration, a set of parsers were successfully created, but one of them failed to commit its changes due to a low-level system exception being raised. Additional messages may be output indicating the reason.

DHCP4_PARSER_COMMIT_FAIL

parser failed to commit changes: %1

On receipt of message containing details to a change of the DHCPv4 server configuration, a set of parsers were successfully created, but one of them failed to commit its changes. The reason for the failure is given in the message.

DHCP4_PARSER_EXCEPTION

failed to create or run parser for configuration element %1

On receipt of message containing details to a change of its configuration, the DHCPv4 server failed to create a parser to decode the contents of the named configuration element, or the creation succeeded but the parsing actions and committal of changes failed. The message has been output in response to a non-Kea exception being raised. Additional messages may give further information.

DHCP4_PARSER_FAIL

failed to create or run parser for configuration element %1: %2

On receipt of message containing details to a change of its configuration, the DHCPv4 server failed to create a parser to decode the contents of the named configuration element, or the creation succeeded but the parsing actions and committal of changes failed. The reason for the failure is given in the message.

DHCP4_POST_ALLOCATION_NAME_UPDATE_FAIL

%1: failed to update hostname %2 in a lease after address allocation: %3

> This message indicates the failure when trying to update the lease and/or options in the server's response with the hostname generated by the server or reserved for the client belonging to a shared network. The latter is the case when the server dynamically switches to another subnet (than initially selected for allocation) from the same shared network.

DHCP4_QUERY_DATA

%1, packet details: %2

> A debug message printing the details of the received packet. The first argument includes the client and the transaction identification information.

DHCP4_RECLAIM_EXPIRED_LEASES_FAIL

failed to reclaim expired leases: %1

> This error message indicates that the reclaim expired leases operation failed and provides the cause of failure.

DHCP4_RELEASE

%1: address %2 was released properly.

> This informational message indicates that an address was released properly. It is a normal operation during client shutdown. The first argument includes the client and transaction identification information. The second argument includes the released IPv4 address.

DHCP4_RELEASE_EXCEPTION

%1: while trying to release address %2 an exception occurred: %3

> This message is output when an error was encountered during an attempt to process a DHCPRELEASE message. The error will not affect the client, which does not expect any response from the server for DHCPRELEASE messages. Depending on the nature of problem, it may affect future server operation. The first argument includes the client and the transaction identification information. The second argument includes the IPv4 address which release was attempted. The last argument includes the detailed error description.

DHCP4_RELEASE_FAIL

%1: failed to remove lease for address %2

> This error message indicates that the software failed to remove a lease from the lease database. It is probably due to an error during a database operation: resolution will most likely require administrator intervention (e.g. check if DHCP process has sufficient privileges to update the database). It may also be triggered if a lease was manually removed from the database during RELEASE message processing. The first argument includes the client and the transaction identification information. The second argument holds the IPv4 address which release was attempted.

DHCP4_RELEASE_FAIL_NO_LEASE

%1: client is trying to release non-existing lease %2

> This debug message is printed when client attempts to release a lease, but no such lease is known to the server. The first argument contains the client and transaction identification information. The second argument contains the IPv4 address which the client is trying to release.

DHCP4_RELEASE_FAIL_WRONG_CLIENT

%1: client is trying to release the lease %2 which belongs to a different client

This debug message is issued when a client is trying to release the lease for the address which is currently used by another client, i.e. the 'client identifier' or 'chaddr' doesn't match between the client and the lease. The first argument includes the client and the transaction identification information. The second argument specifies the leased address.

DHCP4_RESERVED_HOSTNAME_ASSIGNED

%1: server assigned reserved hostname %2

This debug message is issued when the server found a hostname reservation for a client and uses this reservation in a hostname option sent back to this client. The reserved hostname is qualified with a value of 'qualifying-suffix' parameter, if this parameter is specified.

DHCP4_RESPONSE_DATA

%1: responding with packet %2 (type %3), packet details: %4

A debug message including the detailed data about the packet being sent to the client. The first argument contains the client and the transaction identification information. The second and third argument contains the packet name and type respectively. The fourth argument contains detailed packet information.

DHCP4_RESPONSE_FQDN_DATA

%1: including FQDN option in the server's response: %2

This debug message is issued when the server is adding the Client FQDN option in its response to the client. The first argument includes the client and transaction identification information. The second argument includes the details of the FQDN option being included. Note that the name carried in the FQDN option may be modified by the server when the lease is acquired for the client.

DHCP4_RESPONSE_HOSTNAME_DATA

%1: including Hostname option in the server's response: %2

This debug message is issued when the server is adding the Hostname option in its response to the client. The first argument includes the client and transaction identification information. The second argument includes the details of the FQDN option being included. Note that the name carried in the Hostname option may be modified by the server when the lease is acquired for the client.

DHCP4_RESPONSE_HOSTNAME_GENERATE

%1: server has generated hostname %2 for the client

This debug message includes the auto-generated hostname which will be used for the client which message is processed. Hostnames may need to be generated when required by the server's configuration or when the client hasn't supplied its hostname. The first argument includes the client and the transaction identification information. The second argument holds the generated hostname.

DHCP4_SERVER_FAILED

server failed: %1

The DHCPv4 server has encountered a fatal error and is terminating. The reason for the failure is included in the message.

DHCP4_SHUTDOWN

server shutdown

The DHCPv4 server has terminated normally.

DHCP4_SHUTDOWN_REQUEST

shutdown of server requested

This debug message indicates that a shutdown of the DHCPv4 server has been requested via a call to the 'shutdown' method of the core Dhcpv4Srv object.

DHCP4_SRV_CONSTRUCT_ERROR

error creating Dhcpv4Srv object, reason: %1

This error message indicates that during startup, the construction of a core component within the DHCPv4 server (the Dhcpv4 server object) has failed. As a result, the server will exit. The reason for the failure is given within the message.

DHCP4_SRV_D2STOP_ERROR

error stopping IO with DHCP_DDNS during shutdown: %1

This error message indicates that during shutdown, an error occurred while stopping IO between the DHCPv4 server and the DHCP_DDNS server. This is probably due to a programmatic error is not likely to impact either server upon restart. The reason for the failure is given within the message.

DHCP4_SRV_DHCP4O6_ERROR

error stopping IO with DHCPv4o6 during shutdown: %1

This error message indicates that during shutdown, an error occurred while stopping IO between the DHCPv4 server and the DHCPv6o6 server. This is probably due to a programmatic error is not likely to impact either server upon restart. The reason for the failure is given within the message.

DHCP4_SRV_UNLOAD_LIBRARIES_ERROR

error unloading hooks libraries during shutdown: %1

This error message indicates that during shutdown, unloading hooks libraries failed to close them. If the list of libraries is empty it is a programmatic error in the server code. If it is not empty it could be a programmatic error in one of the hooks libraries which could lead to a crash during finalization.

DHCP4_STARTED

Kea DHCPv4 server version %1 started

This informational message indicates that the DHCPv4 server has processed all configuration information and is ready to process DHCPv4 packets. The version is also printed.

DHCP4_STARTING

Kea DHCPv4 server version %1 (%2) starting

This informational message indicates that the DHCPv4 server has processed any command-line switches and is starting. The version is also printed.

DHCP4_START_INFO

pid: %1, server port: %2, client port: %3, verbose: %4

This is a debug message issued during the DHCPv4 server startup. It lists some information about the parameters with which the server is running.

DHCP4_SUBNET_DATA

%1: the selected subnet details: %2

This debug message includes the details of the subnet selected for the client. The first argument includes the client and the transaction identification information. The second arguments includes the subnet details.

DHCP4_SUBNET_DYNAMICALLY_CHANGED

%1: changed selected subnet %2 to subnet %3 from shared network %4 for client assignments

This debug message indicates that the server is using another subnet than initially selected for client assignments. This newly selected subnet belongs to the same shared network as the original subnet. Some reasons why the new subnet was selected include: address pool exhaustion in the original subnet or the fact that the new subnet includes some static reservations for this client.

DHCP4_SUBNET_SELECTED

%1: the subnet with ID %2 was selected for client assignments

This is a debug message noting the selection of a subnet to be used for address and option assignment. Subnet selection is one of the early steps in the processing of incoming client message. The first argument includes the client and the transaction identification information. The second argument holds the selected subnet id.

DHCP4_SUBNET_SELECTION_FAILED

%1: failed to select subnet for the client

This debug message indicates that the server failed to select the subnet for the client which has sent a message to the server. The server will not be able to offer any lease to the client and will drop its message if the received message was DHCPDISCOVER, and will send DHCPNAK if the received message was DHCPREQUEST. The argument includes the client and the transaction identification information.

DHCP4_TESTING_MODE_SEND_TO_SOURCE_ENABLED

All packets will be send to source address of an incoming packet - use only for testing

This message is printed then KEA_TEST_SEND_RESPONSES_TO_SOURCE environment variable is set. It's causing Kea to send packets to source address of incoming packet. Usable just in testing environment to simulate multiple subnet traffic from single source.

25.9 DHCP6

DHCP6_ACTIVATE_INTERFACE

activating interface %1

This message is printed when DHCPv6 server enabled an interface to be used to receive DHCPv6 traffic. IPv6 socket on this interface will be opened once Interface Manager starts up procedure of opening sockets.

DHCP6_ADD_GLOBAL_STATUS_CODE

%1: adding Status Code to DHCPv6 packet: %2

This message is logged when the server is adding the top-level Status Code option. The first argument includes the client and the transaction identification information. The second argument includes the details of the status code.

DHCP6_ADD_STATUS_CODE_FOR_IA

%1: adding Status Code to IA with iaid=%2: %3

This message is logged when the server is adding the Status Code option to an IA. The first argument includes the client and the transaction identification information. The second argument specifies the IAID. The third argument includes the details of the status code.

DHCP6_ALREADY_RUNNING

%1 already running? %2

This is an error message that occurs when the DHCPv6 server encounters a pre-existing PID file which contains the PID of a running process. This most likely indicates an attempt to start a second instance of the server using the same configuration file. It is possible, though unlikely that the PID file is a remnant left behind by a server crash or power failure and the PID it contains refers to a process other than the server. In such an event, it would be necessary to manually remove the PID file. The first argument is the DHCPv6 process name, the second contains the PID and PID file.

DHCP6_BUFFER_RECEIVED

received buffer from %1:%2 to %3:%4 over interface %5

This debug message is logged when the server has received a packet over the socket. When the message is logged the contents of the received packet hasn't been parsed yet. The only available information is the interface and the source and destination addresses/ports.

DHCP6_BUFFER_UNPACK

parsing buffer received from %1 to %2 over interface %3

This debug message is issued when the server starts parsing the received buffer holding the DHCPv6 message. The arguments specify the source and destination addresses as well as the interface over which the buffer has been received.

DHCP6_BUFFER_WAIT_SIGNAL

signal received while waiting for next packet, next waiting signal is %1

This debug message is issued when the server was waiting for the packet, but the wait has been interrupted by the signal received by the process. The signal will be handled before the server starts waiting for next packets. The argument specifies the next signal to be handled by the server.

DHCP6_CB_ON_DEMAND_FETCH_UPDATES_FAIL

error on demand attempt to fetch configuration updates from the configuration backend(s): %1

This error message is issued when the server attempted to fetch configuration updates from the database and this on demand attempt failed. The sole argument which is returned to the config-backend-pull command caller too contains the reason for failure.

DHCP6_CB_PERIODIC_FETCH_UPDATES_FAIL

error on periodic attempt to fetch configuration updates from the configuration backend(s): %1

This error message is issued when the server attempted to fetch configuration updates from the database and this periodic attempt failed. The server will re-try according to the configured value of the config-fetch-wait-time parameter. The sole argument contains the reason for failure.

DHCP6_CB_PERIODIC_FETCH_UPDATES_RETRIES_EXHAUSTED

maximum number of configuration fetch attempts: 10, has been exhausted without success

This error indicates that the server has made a number of unsuccessful periodic attempts to fetch configuration updates from a configuration backend. The server will continue to operate but won't make any further attempts to fetch configuration updates. The administrator must fix the configuration in the database and reload (or restart) the server.

DHCP6_CLASS_ASSIGNED

%1: client packet has been assigned to the following class(es): %2

This debug message informs that incoming packet has been assigned to specified class or classes. This is a normal behavior and indicates successful operation. The first argument specifies the client and transaction identification information. The second argument includes all classes to which the packet has been assigned.

DHCP6_CLASS_UNCONFIGURED

%1: client packet belongs to an unconfigured class: %2

> This debug message informs that incoming packet belongs to a class which cannot be found in the configuration. Either a hook written before the classification was added to Kea is used, or class naming is inconsistent.

DHCP6_CLASS_UNDEFINED

required class %1 has no definition

> This debug message informs that a class is listed for required evaluation but has no definition.

DHCP6_CLASS_UNTESTABLE

required class %1 has no test expression

> This debug message informs that a class was listed for required evaluation but its definition does not include a test expression to evaluate.

DHCP6_COMMAND_RECEIVED

received command %1, arguments: %2

> A debug message listing the command (and possible arguments) received from the Kea control system by the IPv6 DHCP server.

DHCP6_CONFIG_COMPLETE

DHCPv6 server has completed configuration: %1

> This is an informational message announcing the successful processing of a new configuration. it is output during server startup, and when an updated configuration is committed by the administrator. Additional information may be provided.

DHCP6_CONFIG_LOAD_FAIL

configuration error using file: %1, reason: %2

> This error message indicates that the DHCPv6 configuration has failed. If this is an initial configuration (during server's startup) the server will fail to start. If this is a dynamic reconfiguration attempt the server will continue to use an old configuration.

DHCP6_CONFIG_PACKET_QUEUE

DHCPv6 packet queue info after configuration: %1

> This informational message is emitted during DHCPv6 server configuration, immediately after configuring the DHCPv6 packet queue. The information shown depends upon the packet queue type selected.

DHCP6_CONFIG_RECEIVED

received configuration: %1

> A debug message listing the configuration received by the DHCPv6 server. The source of that configuration depends on used configuration backend.

DHCP6_CONFIG_START

DHCPv6 server is processing the following configuration: %1

> This is a debug message that is issued every time the server receives a configuration. That happens start up and also when a server configuration change is committed by the administrator.

DHCP6_CONFIG_UNRECOVERABLE_ERROR

DHCPv6 server new configuration failed with an error which cannot be recovered

This fatal error message is issued when a new configuration raised an error which cannot be recovered. A correct configuration must be applied as soon as possible as the server is no longer working. The configuration can be fixed in several ways. If the control channel is open, config-set with a valid configuration can be used. Alternatively, the original config file on disk could be fixed and SIGHUP signal could be sent (or the config-reload command issued). Finally, the server could be restarted completely.

DHCP6_CONFIG_UNSUPPORTED_OBJECT

DHCPv6 server configuration includes an unsupported object: %1

This error message is issued when the configuration includes an unsupported object (i.e. a top level element).

DHCP6_CONFIG_UPDATE

updated configuration received: %1

A debug message indicating that the IPv6 DHCP server has received an updated configuration from the Kea configuration system.

DHCP6_DB_BACKEND_STARTED

lease database started (type: %1, name: %2)

This informational message is printed every time the IPv6 DHCP server is started. It indicates what database backend type is being to store lease and other information.

DHCP6_DB_RECONNECT_ATTEMPT_FAILED

database reconnect failed: %1

An error message indicating that an attempt to reconnect to the lease and/or host data bases has failed. This occurs after connectivity to either one has been lost and an automatic attempt to reconnect has failed.

DHCP6_DB_RECONNECT_ATTEMPT_SCHEDULE

scheduling attempt %1 of %2 in %3 milliseconds

An informational message indicating that the server is scheduling the next attempt to reconnect to its lease and/or host databases. This occurs when the server has lost databse connectivity and is attempting to reconnect automatically.

DHCP6_DB_RECONNECT_DISABLED

database reconnect is disabled: max-reconnect-tries %1, reconnect-wait-time %2

This is an informational message indicating that connectivity to either the lease or host database or both and that automatic reconnect is not enabled.

DHCP6_DB_RECONNECT_NO_DB_CTL

unexpected error in database reconnect

This is an error message indicating a programmatic error that should not occur. It will prohibit the server from attempting to reconnect to its databases if connectivy is lost, and the server will exit. This error should be reported.

DHCP6_DB_RECONNECT_RETRIES_EXHAUSTED

maximum number of database reconnect attempts: %1, has been exhausted without success, server is shutting down!

This error indicates that the server is shutting down after failing to reconnect to the lease and/or host database(s) after making the maximum configured number of reconnect attempts. Loss of connectivity is typically a network or database server issue.

DHCP6_DDNS_CREATE_ADD_NAME_CHANGE_REQUEST

created name change request: %1

> This debug message is logged when the new Name Change Request has been created to perform the DNS Update, which adds new RRs.

DHCP6_DDNS_FQDN_GENERATED

%1: generated FQDN for the client: %2

> This debug message is logged when the server generated FQDN (name) for the client which message is processed. The names may be generated by the server when required by the server's policy or when the client doesn't provide any specific FQDN in its message to the server. The first argument includes the client and transaction identification information. The second argument includes the generated FQDN.

DHCP6_DDNS_GENERATED_FQDN_UPDATE_FAIL

%1: failed to update the lease using address %2, after generating FQDN for a client, reason: %3

> This message indicates the failure when trying to update the lease and/or options in the server's response with the hostname generated by the server from the acquired address. The first argument includes the client and the transaction identification information. The second argument is a leased address. The third argument includes the reason for the failure.

DHCP6_DDNS_GENERATE_FQDN

%1: client did not send a FQDN option; FQDN will be

> generated for the client. This debug message is issued when the server did not receive a FQDN option from the client and client name replacement is enabled. This provides a means to create DNS entries for unsophisticated clients.

DHCP6_DDNS_RECEIVE_FQDN

%1: received DHCPv6 Client FQDN option: %2

> This debug message is logged when server has found the DHCPv6 Client FQDN Option sent by a client and started processing it. The first argument includes the client and transaction identification information. The second argument includes the received FQDN.

DHCP6_DDNS_REMOVE_OLD_LEASE_FQDN

%1: FQDN for a lease: %2 has changed. New values: hostname = %3, reverse mapping = %4, forward mapping = %5

> This debug message is logged during lease renewal when an old lease that is no longer being offered has a different FQDN than the renewing lesae. Thus the old DNS entries need to be removed. The first argument includes the client and the transaction identification information. The second argument holds the details about the lease for which the FQDN information and/or mappings have changed. The remaining arguments hold the new FQDN information and flags for mappings.

DHCP6_DDNS_REQUEST_SEND_FAILED

failed sending a request to kea-dhcp-ddns, error: %1, ncr: %2

> This error message indicates that IPv6 DHCP server failed to send a DDNS update request to the DHCP-DDNS server. This is most likely a configuration or networking error.

DHCP6_DDNS_RESPONSE_FQDN_DATA

%1: including FQDN option in the server's response: %2

> This debug message is issued when the server is adding the Client FQDN option in its response to the client. The first argument includes the client and transaction identification information. The second

argument includes the details of the FQDN option being included. Note that the name carried in the FQDN option may be modified by the server when the lease is acquired for the client.

DHCP6_DDNS_SEND_FQDN

sending DHCPv6 Client FQDN Option to the client: %1

This debug message is logged when server includes an DHCPv6 Client FQDN Option in its response to the client.

DHCP6_DEACTIVATE_INTERFACE

deactivate interface %1

This message is printed when DHCPv6 server disables an interface from being used to receive DHCPv6 traffic. Sockets on this interface will not be opened by the Interface Manager until interface is enabled.

DHCP6_DECLINE_FAIL

%1: error on decline lease for address %2: %3

This error message indicates that the software failed to deline a lease from the lease database due to an error during a database operation. The first argument includes the client and the transaction identification information. The second argument holds the IPv6 address which decline was attempted. The last one contains the reason for failure.

DHCP6_DECLINE_FAIL_DUID_MISMATCH

Client %1 sent DECLINE for address %2, but it belongs to client with DUID %3

This informational message is printed when a client attempts to decline a lease, but that lease belongs to a different client. The decline request will be rejected.

DHCP6_DECLINE_FAIL_IAID_MISMATCH

Client %1 sent DECLINE for address %2, but used a wrong IAID (%3), instead of expected %4

This informational message is printed when a client attempts to decline a lease. The server has a lease for this address, it belongs to this client, but the recorded IAID does not match what client has sent. This means the server will reject this Decline.

DHCP6_DECLINE_FAIL_LEASE_WITHOUT_DUID

Client %1 sent DECLINE for address %2, but the associated lease has no DUID

This error condition likely indicates database corruption, as every IPv6 lease is supposed to have a DUID, even if it is an empty one.

DHCP6_DECLINE_FAIL_NO_LEASE

Client %1 sent DECLINE for address %2, but there's no lease for it

This informational message is printed when a client tried to decline an address, but the server has no lease for said address. This means that the server's and client's perception of the leases are different. The likely causes of this could be: a confused (e.g. skewed clock) or broken client (e.g. client moved to a different location and didn't notice) or possibly an attack (a rogue client is trying to decline random addresses). The server will inform the client that his decline request was rejected and client should be able to recover from that.

DHCP6_DECLINE_LEASE

Client %1 sent DECLINE for address %2 and the server marked it as declined. The lease will be recovered in %3 seconds.

This informational message indicates that the client leased an address, but discovered that it is being used by some other device and reported this to the server by sending a Decline message. The server marked the lease as declined. This likely indicates a misconfiguration in the network. Either the server is configured with an incorrect pool or there are devices that have statically assigned addresses that are supposed to be assigned by the DHCP server. Both client (will request a different address) and server (will recover the lease after decline-probation-time elapses) will recover automatically. However, if the underlying problem is not solved, the conditions leading to this message may reappear.

DHCP6_DECLINE_PROCESS_IA

Processing of IA (IAID: %1) from client %2 started.

This debug message is printed when the server starts processing an IA_NA option received in Decline message. It's expected that the option will contain an address that is being declined. Specific information will be printed in a separate message.

DHCP6_DEVELOPMENT_VERSION

This software is a development branch of Kea. It is not recommended for production use.

This warning message is displayed when the version is a development (vs stable) one: the second number of the version is odd.

DHCP6_DHCP4O6_PACKET_RECEIVED

received DHCPv4o6 packet from DHCPv4 server (type %1) for %2 port %3 on interface %4

This debug message is printed when the server is receiving a DHCPv4o6 from the DHCPv4 server over inter-process communication.

DHCP6_DHCP4O6_RECEIVE_FAIL

failed to receive DHCPv4o6: %1

This debug message indicates the inter-process communication with the DHCPv4 server failed. The reason for the error is included in the message.

DHCP6_DHCP4O6_RECEIVING

receiving DHCPv4o6 packet from DHCPv4 server

This debug message is printed when the server is receiving a DHCPv4o6 from the DHCPv4 server over inter-process communication socket.

DHCP6_DHCP4O6_SEND_FAIL

failed to send DHCPv4o6 packet: %1

This error is output if the IPv6 DHCP server fails to send an assembled DHCPv4o6 message to a client. The reason for the error is included in the message.

DHCP6_DYNAMIC_RECONFIGURATION

initiate server reconfiguration using file: %1, after receiving SIGHUP signal or config-reload command

This is the info message logged when the DHCPv6 server starts reconfiguration as a result of receiving SIGHUP signal or config-reload command.

DHCP6_DYNAMIC_RECONFIGURATION_FAIL

dynamic server reconfiguration failed with file: %1

This is a fatal error message logged when the dynamic reconfiguration of the DHCP server failed.

DHCP6_FLEX_ID

flexible identifier generated for incoming packet: %1

This debug message is printed when host reservation type is set to flexible identifier and the expression specified in its configuration generated (was evaluated to) an identifier for incoming packet. This debug message is mainly intended as a debugging assistance for flexible identifier.

DHCP6_HANDLE_SIGNAL_EXCEPTION

An exception was thrown while handing signal: %1

This error message is printed when an exception was raised during signal processing. This likely indicates a coding error and should be reported to ISC.

DHCP6_HOOKS_LIBS_RELOAD_FAIL

reload of hooks libraries failed

A “libreload” command was issued to reload the hooks libraries but for some reason the reload failed. Other error messages issued from the hooks framework will indicate the nature of the problem.

DHCP6_HOOK_BUFFER_RCVD_DROP

received buffer from %1 to %2 over interface %3 was dropped because a callout set the drop flag

This debug message is printed when a callout installed on buffer6_receive hook point set the drop flag. For this particular hook point, the setting of the flag by a callout instructs the server to drop the packet. The arguments specify the source and destination address as well as the name of the interface over which the buffer has been received.

DHCP6_HOOK_BUFFER_RCVD_SKIP

received buffer from %1 to %2 over interface %3 is not parsed because a callout set the next step to SKIP

This debug message is printed when a callout installed on buffer6_receive hook point set the next step status to skip. For this particular hook point, this value set by a callout instructs the server to not parse the buffer because it was already parsed by the hook. The arguments specify the source and destination address as well as the name of the interface over which the buffer has been received.

DHCP6_HOOK_BUFFER_SEND_SKIP

%1: prepared DHCPv6 response was dropped because a callout set the next step to SKIP

This debug message is printed when a callout installed on buffer6_send hook point set the next step to SKIP value. For this particular hook point, the SKIP setting a callout instructs the server to drop the packet. Server completed all the processing (e.g. may have assigned, updated or released leases), but the response will not be send to the client. The argument includes the client and transaction identification information.

DHCP6_HOOK_DECLINE_DROP

During Decline processing (client=%1, interface=%2, addr=%3) hook callout set next step to DROP, dropping packet.

This message indicates that the server received DECLINE message, it was verified to be correct and matching server’s lease information. The server called hooks for the lease6_decline hook point and one of the callouts set next step status to DROP. The server will now abort processing of the packet as if it was never received. The lease will continue to be assigned to this client.

DHCP6_HOOK_DECLINE_SKIP

During Decline processing (client=%1, interface=%2, addr=%3) hook callout set status to SKIP, skipping decline.

This message indicates that the server received DECLINE message, it was verified to be correct and matching server’s lease information. The server called hooks for the lease6_decline hook point and one of the callouts set next step status to SKIP. The server will skip the operation of moving the lease to the declined state and will continue processing the packet. In particular, it will send a REPLY message as if the decline actually took place.

DHCP6_HOOK_LEASE6_RELEASE_NA_SKIP

%1: DHCPv6 address lease was not released because a callout set the next step to SKIP

> This debug message is printed when a callout installed on the lease6_release hook point set the next step to SKIP. For this particular hook point, this setting by a callout instructs the server to not release a lease. If a client requested the release of multiples leases (by sending multiple IA options), the server will retain this particular lease and proceed with other releases as usual. The argument holds the client and transaction identification information.

DHCP6_HOOK_LEASE6_RELEASE_PD_SKIP

%1: prefix lease was not released because a callout set the next step to SKIP

> This debug message is printed when a callout installed on lease6_release hook point set the next step to SKIP value. For this particular hook point, that setting by a callout instructs the server to not release a lease. If client requested release of multiples leases (by sending multiple IA options), the server will retains this particular lease and will proceed with other renewals as usual. The argument holds the client and transaction identification information.

DHCP6_HOOK_LEASES6_COMMITTED_DROP

%1: packet is dropped, because a callout set the next step to DROP

> This debug message is printed when a callout installed on the leases6_committed hook point sets the next step to DROP.

DHCP6_HOOK_LEASES6_COMMITTED_PARK

%1: packet is parked, because a callout set the next step to PARK

> This debug message is printed when a callout installed on the lease6_committed hook point sets the next step to PARK.

DHCP6_HOOK_PACKET_RCVD_SKIP

%1: packet is dropped, because a callout set the next step to SKIP

> This debug message is printed when a callout installed on the pkt6_receive hook point sets the next step to SKIP. For this particular hook point, the value setting instructs the server to drop the packet.

DHCP6_HOOK_PACKET_SEND_DROP

%1: prepared DHCPv6 response was not sent because a callout set the next ste to DROP

> This debug message is printed when a callout installed on the pkt6_send hook point set the next step to DROP. For this particular hook point, the setting of the value by a callout instructs the server to drop the packet. This effectively means that the client will not get any response, even though the server processed client's request and acted on it (e.g. possibly allocated a lease). The argument specifies the client and transaction identification information.

DHCP6_HOOK_PACKET_SEND_SKIP

%1: prepared DHCPv6 response is not built because a callout set the next step to SKIP

> This debug message is printed when a callout installed on the pkt6_send hook point set the next step to SKIP. For this particular hook point, the setting of the value by a callout instructs the server to not build the wire data (pack) because it was already done by the book. The argument specifies the client and transaction identification information.

DHCP6_HOOK_SUBNET6_SELECT_DROP

%1: packet was dropped because a callout set the drop flag

This debug message is printed when a callout installed on the subnet6_select hook point set the drop flag. For this particular hook point, the setting of the flag instructs the server to drop the received packet. The argument holds the client and transaction identification information.

DHCP6_HOOK_SUBNET6_SELECT_SKIP

%1: no subnet was selected because a callout set the next step to SKIP

This debug message is printed when a callout installed on the subnet6_select hook point set the next step to SKIP value. For this particular hook point, the setting of this value instructs the server not to choose a subnet, an action that severely limits further processing; the server will be only able to offer global options - no addresses or prefixes will be assigned. The argument holds the client and transaction identification information.

DHCP6_INIT_FAIL

failed to initialize Kea server: %1

The server has failed to establish communication with the rest of Kea, failed to read JSON configuration file or encountered any other critical issue that prevents it from starting up properly. Attached error message provides more details about the issue.

DHCP6_LEASE_ADVERT

%1: lease for address %2 and iaid=%3 will be advertised

This informational message indicates that the server will advertise an address to the client in the ADVERTISE message. The client will request allocation of this address with the REQUEST message sent in the next message exchange. The first argument includes the client and transaction identification information. The remaining arguments hold the allocated address and IAID.

DHCP6_LEASE_ADVERT_FAIL

%1: failed to advertise an address lease for iaid=%2

This message indicates that in response to a received SOLICIT, the server failed to advertise a non-temporary lease for a given client. There may be many reasons for such failure. Each failure is logged in a separate log entry. The first argument holds the client and transaction identification information. The second argument holds the IAID.

DHCP6_LEASE_ALLOC

%1: lease for address %2 and iaid=%3 has been allocated for %4 seconds

This informational message indicates that in response to a client's REQUEST message, the server successfully granted a non-temporary address lease. This is a normal behavior and indicates successful operation. The first argument includes the client and transaction identification information. The remaining arguments hold the allocated address, IAID and validity lifetime.

DHCP6_LEASE_ALLOC_FAIL

%1: failed to grant an address lease for iaid=%2

This message indicates that in response to a received REQUEST, the server failed to grant a non-temporary address lease for the client. There may be many reasons for such failure. Each failure is logged in a separate log entry. The first argument holds the client and transaction identification information. The second argument holds the IAID.

DHCP6_LEASE_DATA

%1: detailed lease information for iaid=%2: %3

This debug message is used to print the detailed information about the allocated lease or a lease which will be advertised to the client. The first argument holds the client and the transaction identification information. The second argument holds the IAID. The third argument holds the detailed lease information.

DHCP6_LEASE_NA_WITHOUT_DUID

%1: address lease for address %2 does not have a DUID

This error message indicates a database consistency problem. The lease database has an entry indicating that the given address is in use, but the lease does not contain any client identification. This is most likely due to a software error: please raise a bug report. As a temporary workaround, manually remove the lease entry from the database. The first argument includes the client and transaction identification information. The second argument holds the address to be released.

DHCP6_LEASE_PD_WITHOUT_DUID

%1: lease for prefix %2/%3 does not have a DUID

This error message indicates a database consistency failure. The lease database has an entry indicating that the given prefix is in use, but the lease does not contain any client identification. This is most likely due to a software error: please raise a bug report. As a temporary workaround, manually remove the lease entry from the database. The first argument includes client and transaction identification information. The second and third argument hold the prefix and the prefix length.

DHCP6_LEASE_RENEW

%1: lease for address %2 and iaid=%3 has been allocated

This informational message indicates that in response to a client's REQUEST message, the server successfully renewed a non-temporary address lease. This is a normal behavior and indicates successful operation. The first argument includes the client and transaction identification information. The remaining arguments hold the allocated address and IAID.

DHCP6_MULTI_THREADING_INFO

enabled: %1, number of threads: %2, queue size: %3

This is a message listing some information about the multi-threading parameters with which the server is running.

DHCP6_NOT_RUNNING

IPv6 DHCP server is not running

A warning message is issued when an attempt is made to shut down the IPv6 DHCP server but it is not running.

DHCP6_NO_INTERFACES

failed to detect any network interfaces

During startup the IPv6 DHCP server failed to detect any network interfaces and is therefore shutting down.

DHCP6_NO_SOCKETS_OPEN

no interface configured to listen to DHCP traffic

This warning message is issued when current server configuration specifies no interfaces that server should listen on, or specified interfaces are not configured to receive the traffic.

DHCP6_OPEN_SOCKET

opening service sockets on port %1

A debug message issued during startup, this indicates that the IPv6 DHCP server is about to open sockets on the specified port.

DHCP6_OPEN_SOCKET_FAIL

failed to open socket: %1

A warning message issued when IfaceMgr fails to open and bind a socket. The reason for the failure is appended as an argument of the log message.

DHCP6_PACKET_DROP_DHCP_DISABLED

%1: DHCP service is globally disabled

This debug message is issued when a packet is dropped because the DHCP service has been temporarily disabled. This affects all received DHCP packets. The service may be enabled by the "dhcp-enable" control command or automatically after a specified amount of time since receiving "dhcp-disable" command.

DHCP6_PACKET_DROP_DROP_CLASS

dropped as member of the special class 'DROP': %1

This debug message is emitted when an incoming packet was classified into the special class 'DROP' and dropped. The packet details are displayed.

DHCP6_PACKET_DROP_DUPLICATE

dropped as sent by the same client than a packet being processed by another thread: dropped %1 by thread %2 as duplicate of %3 processed by %4

Currently multi-threading processing avoids races between packets sent by the same client by dropping new packets until processing is finished. Packet details and thread identifiers are included for both packets in this warning message.

DHCP6_PACKET_DROP_PARSE_FAIL

failed to parse packet from %1 to %2, received over interface %3, reason: %4

The DHCPv6 server has received a packet that it is unable to interpret. The reason why the packet is invalid is included in the message.

DHCP6_PACKET_DROP_SERVERID_MISMATCH

%1: dropping packet with server identifier: %2, server is using: %3

A debug message noting that server has received message with server identifier option that not matching server identifier that server is using.

DHCP6_PACKET_DROP_UNICAST

%1: dropping unicast %2 packet as this packet should be sent to multicast

This debug message is issued when the server drops the unicast packet, because packets of this type must be sent to multicast. The first argument specifies the client and transaction identification information, the second argument specifies packet type.

DHCP6_PACKET_OPTIONS_SKIPPED

An error upacking an option, caused subsequent options to be skipped: %1

A debug message issued when an option failed to unpack correctly, making it impossible to unpack the remaining options in the packet. The server will server will still attempt to service the packet.

DHCP6_PACKET_PROCESS_EXCEPTION

exception occurred during packet processing

This error message indicates that a non-standard exception was raised during packet processing that was not caught by other, more specific exception handlers. This packet will be dropped and the server will continue operation.

DHCP6_PACKET_PROCESS_FAIL

processing of %1 message received from %2 failed: %3

This is a general catch-all message indicating that the processing of the specified packet type from the indicated address failed. The reason is given in the message. The server will not send a response but will instead ignore the packet.

DHCP6_PACKET_PROCESS_STD_EXCEPTION

exception occurred during packet processing: %1

This error message indicates that a standard exception was raised during packet processing that was not caught by other, more specific exception handlers. This packet will be dropped and the server will continue operation.

DHCP6_PACKET_QUEUE_FULL

multi-threading packet queue is full

A debug message noting that the multi-threading packet queue is full so the oldest packet of the queue was dropped to make room for the received one.

DHCP6_PACKET_RECEIVED

%1: %2 (type %3) received from %4 to %5 on interface %6

A debug message noting that the server has received the specified type of packet on the specified interface. The first argument specifies the client and transaction identification information. The second and third argument specify the name of the DHCPv6 message and its numeric type respectively. The remaining arguments specify the source address, destination IP address and the name of the interface on which the message has been received.

DHCP6_PACKET_RECEIVE_FAIL

error on attempt to receive packet: %1

The IPv6 DHCP server tried to receive a packet but an error occurred during this attempt. The reason for the error is included in the message.

DHCP6_PACKET_SEND

%1: trying to send packet %2 (type %3) from [%4]:%5 to [%6]:%7 on interface %8

The arguments specify the client identification information (HW address and client identifier), DHCP message name and type, source IPv6 address and port, destination IPv6 address and port and the interface name.

DHCP6_PACKET_SEND_FAIL

failed to send DHCPv6 packet: %1

This error is output if the IPv6 DHCP server fails to send an assembled DHCP message to a client. The reason for the error is included in the message.

DHCP6_PACK_FAIL

failed to assemble response correctly

This error is output if the server failed to assemble the data to be returned to the client into a valid packet. The reason is most likely to be to a programming error: please raise a bug report.

DHCP6_PARSER_COMMIT_EXCEPTION

parser failed to commit changes

> On receipt of message containing details to a change of the IPv6 DHCP server configuration, a set of parsers were successfully created, but one of them failed to commit its changes due to a low-level system exception being raised. Additional messages may be output indicating the reason.

DHCP6_PARSER_COMMIT_FAIL

parser failed to commit changes: %1

> On receipt of message containing details to a change of the IPv6 DHCP server configuration, a set of parsers were successfully created, but one of them failed to commit its changes. The reason for the failure is given in the message.

DHCP6_PARSER_EXCEPTION

failed to create or run parser for configuration element %1

> On receipt of message containing details to a change of its configuration, the IPv6 DHCP server failed to create a parser to decode the contents of the named configuration element, or the creation succeeded but the parsing actions and committal of changes failed. The message has been output in response to a non-Kea exception being raised. Additional messages may give further information. The most likely cause of this is that the specification file for the server (which details the allowable contents of the configuration) is not correct for this version of Kea. This may be the result of an interrupted installation of an update to Kea.

DHCP6_PARSER_FAIL

failed to create or run parser for configuration element %1: %2

> On receipt of message containing details to a change of its configuration, the IPv6 DHCP server failed to create a parser to decode the contents of the named configuration element, or the creation succeeded but the parsing actions and committal of changes failed. The reason for the failure is given in the message.

DHCP6_PD_LEASE_ADVERT

%1: lease for prefix %2/%3 and iaid=%4 will be advertised

> This informational message indicates that the server will advertise a prefix to the client in the ADVERTISE message. The client will request allocation of this prefix with the REQUEST message sent in the next message exchange. The first argument includes the client and transaction identification information. The remaining arguments hold the allocated prefix, prefix length and IAID.

DHCP6_PD_LEASE_ADVERT_FAIL

%1: failed to advertise a prefix lease for iaid=%2

> This message indicates that in response to a received SOLICIT, the server failed to advertise a prefix lease for a given client. There may be many reasons for such failure. Each failure is logged in a separate log entry. The first argument holds the client and transaction identification information. The second argument holds the IAID.

DHCP6_PD_LEASE_ALLOC

%1: lease for prefix %2/%3 and iaid=%4 has been allocated for %5 seconds

> This informational message indicates that in response to a client's REQUEST message, the server successfully granted a prefix lease. This is a normal behavior and indicates successful operation. The first argument includes the client and transaction identification information. The remaining arguments hold the allocated prefix, prefix length, IAID and validity lifetime.

DHCP6_PD_LEASE_ALLOC_FAIL

%1: failed to grant a prefix lease for iaid=%2

> This message indicates that in response to a received REQUEST, the server failed to grant a prefix lease for the client. There may be many reasons for such failure. Each failure is logged in a separate log entry. The first argument holds the client and transaction identification information. The second argument holds the IAID.

DHCP6_PD_LEASE_RENEW

%1: lease for prefix %2/%3 and iaid=%4 has been allocated

> This informational message indicates that in response to a client's REQUEST message, the server successfully renewed a prefix lease. This is a normal behavior and indicates successful operation. The first argument includes the client and transaction identification information. The remaining arguments hold the allocated prefix, prefix length and and IAID.

DHCP6_PROCESS_IA_NA_EXTEND

%1: extending lease lifetime for IA_NA option with iaid=%2

> This message is logged when the server is starting to extend the lifetime of the address lease associated with the particular IAID. The first argument includes the client and transaction identification information. The second argument contains the IAID.

DHCP6_PROCESS_IA_NA_RELEASE

%1: releasing lease for IA_NA option with iaid=%2

> This message is logged when the server is trying to release the client's as a result of receiving the RELEASE message. The first argument includes the client and transaction identification information. The second argument contains the IAID.

DHCP6_PROCESS_IA_NA_REQUEST

%1: server is processing IA_NA option with iaid=%2 and hint=%3

> This is a debug message that indicates the processing of a received IA_NA option. The first argument contains the client and the transaction identification information. The second argument holds the IAID of the IA_NA option. The third argument may hold the hint for the server about the address that the client would like to have allocated. If there is no hint, the argument should provide the text indicating that the hint hasn't been sent.

DHCP6_PROCESS_IA_PD_EXTEND

%1: extending lease lifetime for IA_PD option with iaid=%2

> This message is logged when the server is starting to extend the lifetime of the prefix lease associated with the particular IAID. The first argument includes the client and transaction identification information. The second argument contains the IAID.

DHCP6_PROCESS_IA_PD_REQUEST

%1: server is processing IA_PD option with iaid=%2 and hint=%3

> This is a debug message that indicates a processing of received IA_PD option. The first argument contains the client and the transaction identification information. The second argument holds the IAID of the IA_PD option. The third argument may hold the hint for the server about the prefix that the client would like to have allocated. If there is no hint, the argument should provide the text indicating that the hint hasn't been sent.

DHCP6_QUERY_DATA

%1, packet details: %2

A debug message printing the details of the received packet. The first argument includes the client and the transaction identification information.

DHCP6_RAPID_COMMIT

%1: Rapid Commit option received, following 2-way exchange

This debug message is issued when the server found a Rapid Commit option in the client's message and 2-way exchanges are supported by the server for the subnet on which the client is connected. The argument specifies the client and transaction identification information.

DHCP6_RECLAIM_EXPIRED_LEASES_FAIL

failed to reclaim expired leases: %1

This error message indicates that the reclaim expired leases operation failed and provides the cause of failure.

DHCP6_RELEASE_NA

%1: binding for address %2 and iaid=%3 was released properly

This informational message indicates that an address was released properly. It is a normal operation during client shutdown.

DHCP6_RELEASE_NA_FAIL

%1: failed to remove address lease for address %2 and iaid=%3

This error message indicates that the software failed to remove an address lease from the lease database. It probably due to an error during a database operation: resolution will most likely require administrator intervention (e.g. check if DHCP process has sufficient privileges to update the database). It may also be triggered if a lease was manually removed from the database during RELEASE message processing. The first argument holds the client and transaction identification information. The second and third argument hold the released address and IAID respectively.

DHCP6_RELEASE_NA_FAIL_WRONG_DUID

%1: client tried to release address %2, but it belongs to another client using duid=%3

This warning message indicates that a client tried to release an address that belongs to a different client. This should not happen in normal circumstances and may indicate a misconfiguration of the client. However, since the client releasing the address will stop using it anyway, there is a good chance that the situation will correct itself.

DHCP6_RELEASE_NA_FAIL_WRONG_IAID

%1: client tried to release address %2, but it used wrong IAID (expected %3, but got %4)

This warning message indicates that client tried to release an address that does belong to it, but the address was expected to be in a different IA (identity association) container. This probably means that the client's support for multiple addresses is flawed.

DHCP6_RELEASE_PD

%1: prefix %2/%3 for iaid=%4 was released properly

This informational message indicates that a prefix was released properly. It is a normal operation during client shutdown. The first argument holds the client and transaction identification information. The second and third argument define the prefix and its length. The fourth argument holds IAID.

DHCP6_RELEASE_PD_FAIL

%1: failed to release prefix %2/%3 for iaid=%4

This error message indicates that the software failed to remove a prefix lease from the lease database. It probably due to an error during a database operation: resolution will most likely require administrator intervention (e.g. check if DHCP process has sufficient privileges to update the database). It may also be triggered if a lease was manually removed from the database during RELEASE message processing. The first argument hold the client and transaction identification information. The second and third argument define the prefix and its length. The fourth argument holds the IAID.

DHCP6_RELEASE_PD_FAIL_WRONG_DUID

%1: client tried to release prefix %2/%3, but it belongs to another client (duid=%4)

This warning message indicates that client tried to release a prefix that belongs to a different client. This should not happen in normal circumstances and may indicate a misconfiguration of the client. However, since the client releasing the prefix will stop using it anyway, there is a good chance that the situation will correct itself. The first argument includes the client and the transaction identification information. The second and third argument include the prefix and prefix length. The last argument holds the DUID of the client holding the lease.

DHCP6_RELEASE_PD_FAIL_WRONG_IAID

%1: client tried to release prefix %2/%3, but it used wrong IAID (expected %4, but got %5)

This warning message indicates that client tried to release a prefix that does belong to it, but the address was expected to be in a different IA (identity association) container. This probably means that the client's support for multiple prefixes is flawed. The first argument includes the client and transaction identification information. The second and third argument identify the prefix. The fourth and fifth argument hold the expected IAID and IAID found respectively.

DHCP6_REQUIRED_OPTIONS_CHECK_FAIL

%1 message received from %2 failed the following check: %3

This message indicates that received DHCPv6 packet is invalid. This may be due to a number of reasons, e.g. the mandatory client-id option is missing, the server-id forbidden in that particular type of message is present, there is more than one instance of client-id or server-id present, etc. The exact reason for rejecting the packet is included in the message.

DHCP6_RESPONSE_DATA

responding with packet type %1 data is %2

A debug message listing the data returned to the client.

DHCP6_SERVER_FAILED

server failed: %1

The IPv6 DHCP server has encountered a fatal error and is terminating. The reason for the failure is included in the message.

DHCP6_SHUTDOWN

server shutdown

The IPv6 DHCP server has terminated normally.

DHCP6_SHUTDOWN_REQUEST

shutdown of server requested

This debug message indicates that a shutdown of the IPv6 server has been requested via a call to the 'shutdown' method of the core Dhcpv6Srv object.

DHCP6_SOCKET_UNICAST

server is about to open socket on address %1 on interface %2

> This is a debug message that inform that a unicast socket will be opened.

DHCP6_SRV_CONSTRUCT_ERROR

error creating Dhcpv6Srv object, reason: %1

> This error message indicates that during startup, the construction of a core component within the IPv6 DHCP server (the Dhcpv6 server object) has failed. As a result, the server will exit. The reason for the failure is given within the message.

DHCP6_SRV_D2STOP_ERROR

error stopping IO with DHCP_DDNS during shutdown: %1

> This error message indicates that during shutdown, an error occurred while stopping IO between the DHCPv6 server and the DHCP_DDNS server. This is probably due to a programmatic error is not likely to impact either server upon restart. The reason for the failure is given within the message.

DHCP6_SRV_UNLOAD_LIBRARIES_ERROR

error unloading hooks libraries during shutdown: %1

> This error message indicates that during shutdown, unloading hooks libraries failed to close them. If the list of libraries is empty it is a programmatic error in the server code. If it is not empty it could be a programmatic error in one of the hooks libraries which could lead to a crash during finalization.

DHCP6_STANDALONE

skipping message queue, running standalone

> This is a debug message indicating that the IPv6 server is running in standalone mode, not connected to the message queue. Standalone mode is only useful during program development, and should not be used in a production environment.

DHCP6_STARTED

Kea DHCPv6 server version %1 started

> This informational message indicates that the IPv6 DHCP server has processed all configuration information and is ready to process DHCPv6 packets. The version is also printed.

DHCP6_STARTING

Kea DHCPv6 server version %1 (%2) starting

> This informational message indicates that the IPv6 DHCP server has processed any command-line switches and is starting. The version is also printed.

DHCP6_START_INFO

pid: %1, server port: %2, client port: %3, verbose: %4

> This is a debug message issued during the IPv6 DHCP server startup. It lists some information about the parameters with which the server is running.

DHCP6_SUBNET_DATA

%1: the selected subnet details: %2

> This debug message includes the details of the subnet selected for the client. The first argument includes the client and the transaction identification information. The second argument includes the subnet details.

DHCP6_SUBNET_DYNAMICALLY_CHANGED

%1: changed selected subnet %2 to subnet %3 from shared network %4 for client assignments

> This debug message indicates that the server is using another subnet than initially selected for client assignments. This newly selected subnet belongs to the same shared network as the original subnet. Some reasons why the new subnet was selected include: address pool exhaustion in the original subnet or the fact that the new subnet includes some static reservations for this client.

DHCP6_SUBNET_SELECTED

%1: the subnet with ID %2 was selected for client assignments

> This is a debug message noting the selection of a subnet to be used for address and option assignment. Subnet selection is one of the early steps in the processing of incoming client message. The first argument includes the client and the transaction identification information. The second argument holds the selected subnet id.

DHCP6_SUBNET_SELECTION_FAILED

%1: failed to select subnet for the client

> This debug message indicates that the server failed to select the subnet for the client which has sent a message to the server. The cause is likely due to a misconfiguration of the server. The packet processing will continue, but the response will only contain generic configuration and no addresses or prefixes. The argument includes the client and the transaction identification information.

DHCP6_UNKNOWN_MSG_RECEIVED

received unknown message (type %1) on interface %2

> This debug message is printed when server receives a message of unknown type. That could either mean missing functionality or invalid or broken relay or client. The list of formally defined message types is available here: http://www.iana.org/assignments/dhcpv6-parameters.

25.10 DHCPSRV

DHCPSRV_CFGMGR_ADD_IFACE

listening on interface %1

> An info message issued when a new interface is being added to the collection of interfaces on which the server listens to DHCP messages.

DHCPSRV_CFGMGR_ADD_SUBNET4

adding subnet %1

> A debug message reported when the DHCP configuration manager is adding the specified IPv4 subnet to its database.

DHCPSRV_CFGMGR_ADD_SUBNET6

adding subnet %1

> A debug message reported when the DHCP configuration manager is adding the specified IPv6 subnet to its database.

DHCPSRV_CFGMGR_ALL_IFACES_ACTIVE

enabling listening on all interfaces

> A debug message issued when the server is being configured to listen on all interfaces.

DHCPSRV_CFGMGR_CFG_DHCP_DDNS

Setting DHCP-DDNS configuration to: %1

> A debug message issued when the server's DHCP-DDNS settings are changed.

DHCPSRV_CFGMGR_CLEAR_ACTIVE_IFACES

stop listening on all interfaces

> A debug message issued when configuration manager clears the internal list of active interfaces. This doesn't prevent the server from listening to the DHCP traffic through open sockets, but will rather be used by Interface Manager to select active interfaces when sockets are re-opened.

DHCPSRV_CFGMGR_CONFIG4_MERGED

Configuration backend data has been merged.

> This is an informational message emitted when the DHCPv4 server has successfully merged configuration data retrieved from its configuration backends into the current configuration.

DHCPSRV_CFGMGR_CONFIG6_MERGED

Configuration backend data has been merged.

> This is an informational message emitted when the DHCPv6 server has successfully merged configuration data retrieved from its configuration backends into the current configuration.

DHCPSRV_CFGMGR_CONFIGURE_SERVERID

server configuration includes specification of a server identifier

> This warning message is issued when the server specified configuration of a server identifier. If this new configuration overrides an existing server identifier, this will affect existing bindings of the clients. Clients will use old server identifier when they renew their bindings. The server will not respond to those renews, and the clients will eventually transition to rebinding state. The server should reassign existing bindings and the clients will subsequently use new server identifier. It is recommended to not modify the server identifier, unless there is a good reason for it, to avoid increased number of renewals and a need for rebinding (increase of multicast traffic, which may be received by multiple servers).

DHCPSRV_CFGMGR_DDNS_PARAMETER_IGNORED

dhpd-ddns:%1 is deprecated, using existing global:%2

> This is an informational message issued during configuration parsing when the server detects that a deprecated parameter has been specified in the "dhcp-ddns" element which conflicts with its corresponding global parameter. When this occurs the server simply ignores the value from dhcp-ddns. The log message shows be the deprecated and the supported parameter names. Note the configuration change only affects the in memory configuration. You should modify your configuration to comply the supported parameters.

DHCPSRV_CFGMGR_DDNS_PARAMETER_MOVED

dhcp-ddns:%1 is deprecated, moving it to global:%2

> This is an informational message issued during configuration parsing when the server detects that a deprecated parameter has been specified in the "dhcp-ddns" element for which no corresponding global value exists. When this occurs the server removes the parameter from dhcp-ddns and inserts the parameter into the global scope. The log message shows be the deprecated and the supported parameter names. Note the configuration change only affects the in memory configuration. You should modify your configuration to comply the supported parameters.

DHCPSRV_CFGMGR_DEL_SUBNET4

IPv4 subnet %1 removed

This debug message is issued when a subnet is successfully removed from the server configuration. The argument identifies the subnet removed.

DHCPSRV_CFGMGR_DEL_SUBNET6

IPv6 subnet %1 removed

This debug message is issued when a subnet is successfully removed from the

DHCPSRV_CFGMGR_NEW_SUBNET4

a new subnet has been added to configuration: %1

This is an informational message reporting that the configuration has been extended to include the specified IPv4 subnet.

DHCPSRV_CFGMGR_NEW_SUBNET6

a new subnet has been added to configuration: %1

This is an informational message reporting that the configuration has been extended to include the specified subnet.

DHCPSRV_CFGMGR_NO_SUBNET4

no suitable subnet is defined for address hint %1

This debug message is output when the DHCP configuration manager has received a request for an IPv4 subnet for the specified address, but no such subnet exists.

DHCPSRV_CFGMGR_NO_SUBNET6

no suitable subnet is defined for address hint %1

This debug message is output when the DHCP configuration manager has received a request for an IPv6 subnet for the specified address, but no such subnet exists.

DHCPSRV_CFGMGR_ONLY_SUBNET4

retrieved subnet %1 for address hint %2

This is a debug message reporting that the DHCP configuration manager has returned the specified IPv4 subnet when given the address hint specified because it is the only subnet defined.

DHCPSRV_CFGMGR_ONLY_SUBNET6

retrieved subnet %1 for address hint %2

This is a debug message reporting that the DHCP configuration manager has returned the specified IPv6 subnet when given the address hint specified because it is the only subnet defined.

DHCPSRV_CFGMGR_OPTION_DUPLICATE

multiple options with the code: %1 added to the subnet: %2

This warning message is issued on an attempt to configure multiple options with the same option code for the particular subnet. Adding multiple options is uncommon for DHCPv6, but it is not prohibited.

DHCPSRV_CFGMGR_RELAY_IP_ADDRESS_DEPRECATED

“relay” uses “ip-address”, which has been deprecated, please use “ip-addresses”: %1

This is debug message issued when the “relay” element being parse contains “ip-address” rather than its replacement, “ip-addresses”. The server will still honor the value but users are encouraged to move to the new list parameter.

DHCPSRV_CFGMGR_SOCKET_RAW_UNSUPPORTED

use of raw sockets is unsupported on this OS, UDP sockets will be used

> This warning message is logged when the user specified that the DHCPv4 server should use the raw sockets to receive the DHCP messages and respond to the clients, but the use of raw sockets is not supported on the particular environment. The raw sockets are useful when the server must respond to the directly connected clients which don't have an address yet. If the raw sockets are not supported by Kea on the particular platform, Kea will fall back to use of the IP/UDP sockets. The responses to the directly connected clients will be broadcast. The responses to relayed clients will be unicast as usual.

DHCPSRV_CFGMGR_SOCKET_TYPE_DEFAULT

"dhcp-socket-type" not specified , using default socket type %1

> This informational message is logged when the administrator hasn't specified the "dhcp-socket-type" parameter in configuration for interfaces. In such case, the default socket type will be used.

DHCPSRV_CFGMGR_SOCKET_TYPE_SELECT

using socket type %1

> This informational message is logged when the DHCPv4 server selects the socket type to be used for all sockets that will be opened on the interfaces. Typically, the socket type is specified by the server administrator. If the socket type hasn't been specified, the raw socket will be selected. If the raw socket has been selected but Kea doesn't support the use of raw sockets on the particular OS, it will use an UDP socket instead.

DHCPSRV_CFGMGR_SUBNET4

retrieved subnet %1 for address hint %2

> This is a debug message reporting that the DHCP configuration manager has returned the specified IPv4 subnet when given the address hint specified as the address is within the subnet.

DHCPSRV_CFGMGR_SUBNET4_ADDR

selected subnet %1 for packet received by matching address %2

> This is a debug message reporting that the DHCP configuration manager has returned the specified IPv4 subnet for a received packet. This particular subnet was selected, because an IPv4 address was matched which belonged to that subnet.

DHCPSRV_CFGMGR_SUBNET4_IFACE

selected subnet %1 for packet received over interface %2

> This is a debug message reporting that the DHCP configuration manager has returned the specified IPv4 subnet for a packet received over the given interface. This particular subnet was selected, because it was specified as being directly reachable over the given interface. (see 'interface' parameter in the subnet4 definition).

DHCPSRV_CFGMGR_SUBNET4_RELAY

selected subnet %1, because of matching relay addr %2

> This is a debug message reporting that the DHCP configuration manager has returned the specified IPv4 subnet, because detected relay agent address matches value specified for this subnet.

DHCPSRV_CFGMGR_SUBNET6

retrieved subnet %1 for address hint %2

> This is a debug message reporting that the DHCP configuration manager has returned the specified IPv6 subnet when given the address hint specified as the address is within the subnet.

DHCPSRV_CFGMGR_SUBNET6_IFACE

selected subnet %1 for packet received over interface %2

> This is a debug message reporting that the DHCP configuration manager has returned the specified IPv6 subnet for a packet received over given interface. This particular subnet was selected, because it was specified as being directly reachable over given interface. (see 'interface' parameter in the subnet6 definition).

DHCPSRV_CFGMGR_SUBNET6_IFACE_ID

selected subnet %1 (interface-id match) for incoming packet

> This is a debug message reporting that the DHCP configuration manager has returned the specified IPv6 subnet for a received packet. This particular subnet was selected, because value of interface-id option matched what was configured in the server's interface-id option for that selected subnet6. (see 'interface-id' parameter in the subnet6 definition).

DHCPSRV_CFGMGR_SUBNET6_RELAY

selected subnet %1, because of matching relay addr %2

> This is a debug message reporting that the DHCP configuration manager has returned the specified IPv6 subnet, because detected relay agent address matches value specified for this subnet.

DHCPSRV_CFGMGR_UNICAST_LINK_LOCAL

specified link local address %1 for unicast traffic on interface %2

> This warning message is logged when user specified a link-local address to receive unicast traffic. The warning message is issued because it is an uncommon use.

DHCPSRV_CFGMGR_UPDATE_SUBNET4

updating subnet %1 (result %2)

> A debug message reported when the DHCP configuration manager is updating the specified IPv4 subnet in its current configuration. Subnet ID and result (expected to be true) are displayed.

DHCPSRV_CFGMGR_UPDATE_SUBNET6

updating subnet %1 (result %2)

> A debug message reported when the DHCP configuration manager is replacing the specified IPv6 subnet in its current configuration. Subnet ID and result (expected to be true) are displayed.

DHCPSRV_CFGMGR_USE_ADDRESS

listening on address %1, on interface %2

> A message issued when the server is configured to listen on the explicitly specified IP address on the given interface.

DHCPSRV_CFGMGR_USE_UNICAST

listening on unicast address %1, on interface %2

> An info message issued when configuring the DHCP server to listen on the unicast address on the specific interface.

DHCPSRV_CLOSE_DB

closing currently open %1 database

> This is a debug message, issued when the DHCP server closes the currently open lease database. It is issued at program shutdown and whenever the database access parameters are changed: in the latter case, the server closes the currently open database, and opens a database using the new parameters.

DHCPSRV_CQL_ADD_ADDR4

adding IPv4 lease with address %1

> A debug message issued when the server is about to add an IPv4 lease with the specified address to the Cassandra backend database.

DHCPSRV_CQL_ADD_ADDR6

adding IPv6 lease with address %1

> A debug message issued when the server is about to add an IPv6 lease with the specified address to the Cassandra backend database.

DHCPSRV_CQL_COMMIT

committing to Cassandra database.

> A commit call been issued on the server. For Cassandra, this is a no-op.

DHCPSRV_CQL_CONNECTION_BEGIN_TRANSACTION

begin transaction on current connection.

> The server has issued a begin transaction call.

DHCPSRV_CQL_CONNECTION_COMMIT

committing to Cassandra database on current connection.

> A commit call been issued on the server. For Cassandra, this is a no-op.

DHCPSRV_CQL_CONNECTION_ROLLBACK

rolling back Cassandra database on current connection.

> The code has issued a rollback call. For Cassandra, this is a no-op.

DHCPSRV_CQL_DB

opening Cassandra lease database: %1

> This informational message is logged when a DHCP server (either V4 or V6) is about to open a Cassandra lease database. The parameters of the connection including database name and username needed to access it (but not the password if any) are logged.

DHCPSRV_CQL_DEALLOC_ERROR

An error occurred while closing the CQL connection: %1

> This is an error message issued when a DHCP server (either V4 or V6) experienced and error freeing CQL database resources as part of closing its connection to the Cassandra database. The connection is closed as part of normal server shutdown. This error is most likely a programmatic issue that is highly unlikely to occur or negatively impact server operation.

DHCPSRV_CQL_DELETE_ADDR

deleting lease for address %1

> A debug message issued when the server is attempting to delete a lease from the Cassandra database for the specified address.

DHCPSRV_CQL_DELETE_EXPIRED_RECLAIMED4

deleting reclaimed IPv4 leases that expired more than %1 seconds ago

A debug message issued when the server is removing reclaimed DHCPv4 leases which have expired longer than a specified period of time. The argument is the amount of time Kea waits after a reclaimed lease expires before considering its removal.

DHCPSRV_CQL_DELETE_EXPIRED_RECLAIMED6

deleting reclaimed IPv6 leases that expired more than %1 seconds ago

A debug message issued when the server is removing reclaimed DHCPv6 leases which have expired longer than a specified period of time. The argument is the amount of time Kea waits after a reclaimed lease expires before considering its removal.

DHCPSRV_CQL_GET4

obtaining all IPv4 leases

A debug message issued when the server is attempting to obtain all IPv4 leases from the Cassandra database.

DHCPSRV_CQL_GET_ADDR4

obtaining IPv4 lease for address %1

A debug message issued when the server is attempting to obtain an IPv4 lease from the Cassandra database for the specified address.

DHCPSRV_CQL_GET_ADDR6

obtaining IPv6 lease for address %1 and lease type %2

A debug message issued when the server is attempting to obtain an IPv6 lease from the Cassandra database for the specified address.

DHCPSRV_CQL_GET_CLIENTID

obtaining IPv4 leases for client ID %1

A debug message issued when the server is attempting to obtain a set of IPv4 leases from the Cassandra database for a client with the specified client identification.

DHCPSRV_CQL_GET_CLIENTID_HWADDR_SUBID

obtaining IPv4 lease for client ID %1, hardware address %2 and subnet ID %3

A debug message issued when the server is attempting to obtain an IPv4 lease from the Cassandra database for a client with the specified client ID, hardware address and subnet ID.

DHCPSRV_CQL_GET_EXPIRED4

obtaining maximum %1 of expired IPv4 leases

A debug message issued when the server is attempting to obtain expired IPv4 leases to reclaim them. The maximum number of leases to be retrieved is logged in the message.

DHCPSRV_CQL_GET_EXPIRED6

obtaining maximum %1 of expired IPv6 leases

A debug message issued when the server is attempting to obtain expired IPv6 leases to reclaim them. The maximum number of leases to be retrieved is logged in the message.

DHCPSRV_CQL_GET_HOSTNAME4

obtaining IPv4 leases for hostname %1

A debug message issued when the server is attempting to obtain a set of IPv4 leases from the Cassandra database for a client with the specified hostname.

DHCPSRV_CQL_GET_HOSTNAME6

obtaining IPv6 leases for hostname %1

> A debug message issued when the server is attempting to obtain a set of IPv6 leases from the Cassandra database for a client with the specified hostname.

DHCPSRV_CQL_GET_HWADDR

obtaining IPv4 leases for hardware address %1

> A debug message issued when the server is attempting to obtain a set of IPv4 leases from the Cassandra database for a client with the specified hardware address.

DHCPSRV_CQL_GET_IAID_DUID

obtaining IPv6 leases for IAID %1 and DUID %2 and lease type %3

> A debug message issued when the server is attempting to obtain a set of IPv6 leases from the Cassandra database for a client with the specified IAID (Identity Association ID) and DUID (DHCP Unique Identifier).

DHCPSRV_CQL_GET_IAID_SUBID_DUID

obtaining IPv6 leases for IAID %1, Subnet ID %2, DUID %3 and lease type %4

> A debug message issued when the server is attempting to obtain an IPv6 lease from the Cassandra database for a client with the specified IAID (Identity Association ID), Subnet ID and DUID (DHCP Unique Identifier).

DHCPSRV_CQL_GET_PAGE4

obtaining at most %1 IPv4 leases starting from address %2

> A debug message issued when the server is attempting to obtain a page of leases beginning with the specified address.

DHCPSRV_CQL_GET_PAGE6

obtaining at most %1 IPv6 leases starting from address %2

> A debug message issued when the server is attempting to obtain a page of leases beginning with the specified address.

DHCPSRV_CQL_GET_SUBID4

obtaining IPv4 leases for subnet ID %1

> A debug message issued when the server is attempting to obtain all IPv4 leases for a given subnet identifier from the Cassandra database.

DHCPSRV_CQL_GET_SUBID_CLIENTID

obtaining IPv4 lease for subnet ID %1 and client ID %2

> A debug message issued when the server is attempting to obtain an IPv4 lease from the Cassandra database for a client with the specified subnet ID and client ID.

DHCPSRV_CQL_GET_SUBID_HWADDR

obtaining IPv4 lease for subnet ID %1 and hardware address %2

> A debug message issued when the server is attempting to obtain an IPv4 lease from the Cassandra database for a client with the specified subnet ID and hardware address.

DHCPSRV_CQL_GET_VERSION

obtaining schema version information

A debug message issued when the server is about to obtain schema version information from the Cassandra database.

DHCPSRV_CQL_HOST_ADD

Adding host information to the database

An informational message logged when options belonging to any reservation from a single host are inserted.

DHCPSRV_CQL_HOST_DB

Connecting to CQL hosts database: %1

An informational message logged when the CQL hosts database is about to be connected to. The parameters of the connection including database name and username needed to access it (but not the password if any) are logged.

DHCPSRV_CQL_HOST_DB_GET_VERSION

obtaining schema version information for the CQL hosts database

A debug message issued when the server is about to obtain schema version information from the CQL hosts database.

DHCPSRV_CQL_HOST_GET4

Retrieving one DHCPv4 host from a CQL database

An informational message logged when a DHCP server is about to retrieve one host from a CQL database by IPv4 criteria.

DHCPSRV_CQL_HOST_GET6

Retrieving one DHCPv6 host from a CQL database

An informational message logged when a DHCP server is about to retrieve one host from a CQL database by IPv6 criteria.

DHCPSRV_CQL_HOST_GET_ALL

Retrieving multiple hosts from a CQL database

An informational message logged when multiple hosts from a CQL database are retrieved.

DHCPSRV_CQL_LEASE_EXCEPTION_THROWN

Exception thrown during Cassandra operation: %1

DHCPSRV_CQL_ROLLBACK

rolling back Cassandra database.

The code has issued a rollback call. For Cassandra, this is a no-op.

DHCPSRV_CQL_UPDATE_ADDR4

updating IPv4 lease for address %1

A debug message issued when the server is attempting to update IPv4 lease from the Cassandra database for the specified address.

DHCPSRV_CQL_UPDATE_ADDR6

updating IPv6 lease for address %1

A debug message issued when the server is attempting to update IPv6 lease from the Cassandra database for the specified address.

DHCPSRV_DHCP4O6_RECEIVED_BAD_PACKET

received bad DHCPv4o6 packet: %1

> A bad DHCPv4o6 packet was received.

DHCPSRV_DHCP_DDNS_ERROR_EXCEPTION

error handler for DHCP_DDNS IO generated an expected exception: %1

> This is an error message that occurs when an attempt to send a request to kea-dhcp-ddns fails there registered error handler threw an uncaught exception. This is a programmatic error which should not occur. By convention, the error handler should not propagate exceptions. Please report this error.

DHCPSRV_DHCP_DDNS_HANDLER_NULL

error handler for DHCP_DDNS IO is not set.

> This is an error message that occurs when an attempt to send a request to kea-dhcp-ddns fails and there is no registered error handler. This is a programmatic error which should never occur and should be reported.

DHCPSRV_DHCP_DDNS_NCR_REJECTED

NameChangeRequest rejected by the sender: %1, ncr: %2

> This is an error message indicating that NameChangeSender used to deliver DDNS update requests to kea-dhcp-ddns rejected the request. This most likely cause is the sender's queue has reached maximum capacity. This would imply that requests are being generated faster than they can be delivered.

DHCPSRV_DHCP_DDNS_NCR_SENT

NameChangeRequest sent to kea-dhcp-ddns: %1

> A debug message issued when a NameChangeRequest has been successfully sent to kea-dhcp-ddns.

DHCPSRV_DHCP_DDNS_SENDER_STARTED

NameChangeRequest sender has been started: %1

> A informational message issued when a communications with kea-dhcp-ddns has been successfully started.

DHCPSRV_DHCP_DDNS_SENDER_STOPPED

NameChangeRequest sender has been stopped.

> A informational message issued when a communications with kea-dhcp-ddns has been stopped. This normally occurs during reconfiguration and as part of normal shutdown. It may occur if kea-dhcp-ddns communications breakdown.

DHCPSRV_DHCP_DDNS_SUSPEND_UPDATES

DHCP_DDNS updates are being suspended.

> This is a warning message indicating the DHCP_DDNS updates have been turned off. This should only occur if IO errors communicating with kea-dhcp-ddns have been experienced. Any such errors should have preceding entries in the log with details. No further attempts to communicate with kea-dhcp-ddns will be made without intervention.

DHCPSRV_HOOK_LEASE4_RECOVER_SKIP

DHCPv4 lease %1 was not recovered from the declined state because a callout set the skip status.

> This debug message is printed when a callout installed on lease4_recover hook point set the next step status to SKIP. For this particular hook point, this indicates that the server should not recover the lease from declined state. The server will leave the lease as it is, in the declined state. The server will attempt to recover it the next time decline recovery procedure takes place.

DHCPSRV_HOOK_LEASE4_RENEW_SKIP

DHCPv4 lease was not renewed because a callout set the skip flag.

> This debug message is printed when a callout installed on lease4_renew hook point set the skip flag. For this particular hook point, the setting of the flag by a callout instructs the server to not renew a lease. The server will use existing lease as it is, without extending its lifetime.

DHCPSRV_HOOK_LEASE4_SELECT_SKIP

Lease4 creation was skipped, because of callout skip flag.

> This debug message is printed when a callout installed on lease4_select hook point sets the skip flag. It means that the server was told that no lease4 should be assigned. The server will not put that lease in its database and the client will get a NAK packet.

DHCPSRV_HOOK_LEASE6_EXTEND_SKIP

DHCPv6 lease lifetime was not extended because a callout set the skip flag for message %1

> This debug message is printed when a callout installed on lease6_renew or the lease6_rebind hook point set the skip flag. For this particular hook point, the setting of the flag by a callout instructs the server to not extend the lifetime for a lease. If the client requested renewal of multiple leases (by sending multiple IA options), the server will skip the renewal of the one in question and will proceed with other renewals as usual.

DHCPSRV_HOOK_LEASE6_RECOVER_SKIP

DHCPv6 lease %1 was not recovered from declined state because a callout set the skip status.

> This debug message is printed when a callout installed on lease6_recover hook point set the next step status to SKIP. For this particular hook point, this indicates that the server should not recover the lease from declined state. The server will leave the lease as it is, in the declined state. The server will attempt to recover it the next time decline recovery procedure takes place.

DHCPSRV_HOOK_LEASE6_SELECT_SKIP

Lease6 (non-temporary) creation was skipped, because of callout skip flag.

> This debug message is printed when a callout installed on lease6_select hook point sets the skip flag. It means that the server was told that no lease6 should be assigned. The server will not put that lease in its database and the client will get a NoAddrsAvail for that IA_NA option.

DHCPSRV_INVALID_ACCESS

invalid database access string: %1

> This is logged when an attempt has been made to parse a database access string and the attempt ended in error. The access string in question - which should be of the form 'keyword=value keyword=value...' is included in the message.

DHCPSRV_LEASE_SANITY_FAIL

The lease %1 with subnet-id %2 failed subnet-id checks (%3).

> This warning message is printed when the lease being loaded does not match the configuration. Due to lease-checks value, the lease will be loaded, but it will most likely be unused by Kea, as there is no subnet that matches the IP address associated with the lease.

DHCPSRV_LEASE_SANITY_FAIL_DISCARD

The lease %1 with subnet-id %2 failed subnet-id checks (%3) and was dropped.

> This warning message is printed when a lease was loaded, but Kea was told (by setting lease-checks parameter) to discard leases with inconsistent data. The lease was discarded, because either there is no subnet configured with matching subnet-id or the address of the lease does not belong to the subnet.

DHCPSRV_LEASE_SANITY_FIXED

The lease %1 with subnet-id %2 failed subnet-id checks, but was corrected to subnet-id %3.

> This informational message is printed when a lease was loaded, but had incorrect subnet-id value. The lease-checks parameter was set to a value that told Kea to try to correct the problem. There is a matching subnet, so Kea updated subnet-id and loaded the lease successfully.

DHCPSRV_MEMFILE_ADD_ADDR4

adding IPv4 lease with address %1

> A debug message issued when the server is about to add an IPv4 lease with the specified address to the memory file backend database.

DHCPSRV_MEMFILE_ADD_ADDR6

adding IPv6 lease with address %1

> A debug message issued when the server is about to add an IPv6 lease with the specified address to the memory file backend database.

DHCPSRV_MEMFILE_BEGIN_TRANSACTION

committing to memory file database

> The code has issued a begin transaction call. For the memory file database, this is a no-op.

DHCPSRV_MEMFILE_COMMIT

committing to memory file database

> The code has issued a commit call. For the memory file database, this is a no-op.

DHCPSRV_MEMFILE_CONVERTING_LEASE_FILES

running LFC now to convert lease files to the current schema: %1.%2

> A warning message issued when the server has detected lease files that need to be either upgraded or downgraded to match the server's schema, and that the server is automatically running the LFC process to perform the conversion. This should only occur the first time the server is launched following a Kea installation upgrade (or downgrade).

DHCPSRV_MEMFILE_DB

opening memory file lease database: %1

> This informational message is logged when a DHCP server (either V4 or V6) is about to open a memory file lease database. The parameters of the connection including database name and username needed to access it (but not the password if any) are logged.

DHCPSRV_MEMFILE_DELETE_ADDR

deleting lease for address %1

> A debug message issued when the server is attempting to delete a lease for the specified address from the memory file database for the specified address.

DHCPSRV_MEMFILE_DELETE_EXPIRED_RECLAIMED4

deleting reclaimed IPv4 leases that expired more than %1 seconds ago

A debug message issued when the server is removing reclaimed DHCPv4 leases which have expired longer than a specified period of time. The argument is the amount of time Kea waits after a reclaimed lease expires before considering its removal.

DHCPSRV_MEMFILE_DELETE_EXPIRED_RECLAIMED6

deleting reclaimed IPv6 leases that expired more than %1 seconds ago

A debug message issued when the server is removing reclaimed DHCPv6 leases which have expired longer than a specified period of time. The argument is the amount of time Kea waits after a reclaimed lease expires before considering its removal.

DHCPSRV_MEMFILE_DELETE_EXPIRED_RECLAIMED_START

starting deletion of %1 expired-reclaimed leases

A debug message issued when the server has found expired-reclaimed leases to be removed. The number of leases to be removed is logged in the message.

DHCPSRV_MEMFILE_GET4

obtaining all IPv4 leases

A debug message issued when the server is attempting to obtain all IPv4 leases from the memory file database.

DHCPSRV_MEMFILE_GET6

obtaining all IPv6 leases

A debug message issued when the server is attempting to obtain all IPv6 leases from the memory file database.

DHCPSRV_MEMFILE_GET6_DUID

obtaining IPv6 leases for DUID %1

A debug message issued when the server is attempting to obtain IPv6 leases from the memory file database for the DUID.

DHCPSRV_MEMFILE_GET_ADDR4

obtaining IPv4 lease for address %1

A debug message issued when the server is attempting to obtain an IPv4 lease from the memory file database for the specified address.

DHCPSRV_MEMFILE_GET_ADDR6

obtaining IPv6 lease for address %1 and lease type %2

A debug message issued when the server is attempting to obtain an IPv6 lease from the memory file database for the specified address.

DHCPSRV_MEMFILE_GET_CLIENTID

obtaining IPv4 leases for client ID %1

A debug message issued when the server is attempting to obtain a set of IPv4 leases from the memory file database for a client with the specified client identification.

DHCPSRV_MEMFILE_GET_CLIENTID_HWADDR_SUBID

obtaining IPv4 lease for client ID %1, hardware address %2 and subnet ID %3

A debug message issued when the server is attempting to obtain an IPv4 lease from the memory file database for a client with the specified client ID, hardware address and subnet ID.

DHCPSRV_MEMFILE_GET_EXPIRED4

obtaining maximum %1 of expired IPv4 leases

> A debug message issued when the server is attempting to obtain expired IPv4 leases to reclaim them. The maximum number of leases to be retrieved is logged in the message.

DHCPSRV_MEMFILE_GET_EXPIRED6

obtaining maximum %1 of expired IPv6 leases

> A debug message issued when the server is attempting to obtain expired IPv6 leases to reclaim them. The maximum number of leases to be retrieved is logged in the message.

DHCPSRV_MEMFILE_GET_HOSTNAME4

obtaining IPv4 leases for hostname %1

> A debug message issued when the server is attempting to obtain a set of IPv4 leases from the memory file database for a client with the specified hostname.

DHCPSRV_MEMFILE_GET_HOSTNAME6

obtaining IPv6 leases for hostname %1

> A debug message issued when the server is attempting to obtain a set of IPv6 leases from the memory file database for a client with the specified hostname.

DHCPSRV_MEMFILE_GET_HWADDR

obtaining IPv4 leases for hardware address %1

> A debug message issued when the server is attempting to obtain a set of IPv4 leases from the memory file database for a client with the specified hardware address.

DHCPSRV_MEMFILE_GET_IAID_DUID

obtaining IPv6 leases for IAID %1 and DUID %2 and lease type %3

> A debug message issued when the server is attempting to obtain a set of IPv6 leases from the memory file database for a client with the specified IAID (Identity Association ID) and DUID (DHCP Unique Identifier).

DHCPSRV_MEMFILE_GET_IAID_SUBID_DUID

obtaining IPv6 leases for IAID %1, Subnet ID %2, DUID %3 and lease type %4

> A debug message issued when the server is attempting to obtain an IPv6 lease from the memory file database for a client with the specified IAID (Identity Association ID), Subnet ID and DUID (DHCP Unique Identifier).

DHCPSRV_MEMFILE_GET_PAGE4

obtaining at most %1 IPv4 leases starting from address %2

> A debug message issued when the server is attempting to obtain a page of leases beginning with the specified address.

DHCPSRV_MEMFILE_GET_PAGE6

obtaining at most %1 IPv6 leases starting from address %2

> A debug message issued when the server is attempting to obtain a page of leases beginning with the specified address.

DHCPSRV_MEMFILE_GET_SUBID4

obtaining IPv4 leases for subnet ID %1

A debug message issued when the server is attempting to obtain all IPv4 leases for a given subnet identifier from the memory file database.

DHCPSRV_MEMFILE_GET_SUBID6

obtaining IPv6 leases for subnet ID %1

A debug message issued when the server is attempting to obtain all IPv6 leases for a given subnet identifier from the memory file database.

DHCPSRV_MEMFILE_GET_SUBID_CLIENTID

obtaining IPv4 lease for subnet ID %1 and client ID %2

A debug message issued when the server is attempting to obtain an IPv4 lease from the memory file database for a client with the specified subnet ID and client ID.

DHCPSRV_MEMFILE_GET_SUBID_HWADDR

obtaining IPv4 lease for subnet ID %1 and hardware address %2

A debug message issued when the server is attempting to obtain an IPv4 lease from the memory file database for a client with the specified subnet ID and hardware address.

DHCPSRV_MEMFILE_GET_VERSION

obtaining schema version information

A debug message issued when the server is about to obtain schema version information from the memory file database.

DHCPSRV_MEMFILE_LEASE_FILE_LOAD

loading leases from file %1

An info message issued when the server is about to start reading DHCP leases from the lease file. All leases currently held in the memory will be replaced by those read from the file.

DHCPSRV_MEMFILE_LEASE_LOAD

loading lease %1

A debug message issued when DHCP lease is being loaded from the file to memory.

DHCPSRV_MEMFILE_LEASE_LOAD_ROW_ERROR

discarding row %1, error: %2

An error message issued the the DHCP lease being loaded from the given row of the lease file fails. The log message should contain the specific reason the row was discarded. The server will continue loading the remaining data. This may indicate a corrupt lease file.

DHCPSRV_MEMFILE_LFC_EXECUTE

executing Lease File Cleanup using: %1

An informational message issued when the Memfile lease database backend starts a new process to perform Lease File Cleanup.

DHCPSRV_MEMFILE_LFC_LEASE_FILE_RENAME_FAIL

failed to rename the current lease file %1 to %2, reason: %3

An error message logged when the Memfile lease database backend fails to move the current lease file to a new file on which the cleanup should be performed. This effectively means that the lease file cleanup will not take place.

DHCPSRV_MEMFILE_LFC_LEASE_FILE_REOPEN_FAIL

failed to reopen lease file %1 after preparing input file for lease file cleanup, reason: %2, new leases will not be persisted!

> An error message logged when the Memfile lease database backend failed to re-open or re-create the lease file after renaming the lease file for lease file cleanup. The server will continue to operate but leases will not be persisted to disk.

DHCPSRV_MEMFILE_LFC_SETUP

setting up the Lease File Cleanup interval to %1 sec

> An informational message logged when the Memfile lease database backend configures the LFC to be executed periodically. The argument holds the interval in seconds in which the LFC will be executed.

DHCPSRV_MEMFILE_LFC_SPAWN_FAIL

lease file cleanup failed to run because kea-lfc process couldn't be spawned

> This error message is logged when the Kea server fails to run kea-lfc, the program that cleans up the lease file. The server will try again the next time a lease file cleanup is scheduled. Although this message should not appear and the reason why it did investigated, the occasional failure to start the lease file cleanup will not impact operations. Should the failure persist however, the size of the lease file will increase without bound.

DHCPSRV_MEMFILE_LFC_START

starting Lease File Cleanup

> An informational message issued when the Memfile lease database backend starts the periodic Lease File Cleanup.

DHCPSRV_MEMFILE_LFC_UNREGISTER_TIMER_FAILED

failed to unregister timer 'memfile-lfc': %1

> This debug message is logged when Memfile backend fails to unregister timer used for lease file cleanup scheduling. There are several reasons why this could occur, although the most likely cause is that the system is being shut down and some other component has unregistered the timer. The message includes the reason for this error.

DHCPSRV_MEMFILE_NEEDS_DOWNGRADING

version of lease file: %1 schema is later than version %2

> A warning message issued when the schema of the lease file loaded by the server is newer than the memfile schema of the server. The server converts the lease data from newer schemas to its schema as it is read, therefore the lease information in use by the server will be correct. Note though, that any data stored in newer schema fields will be dropped. What remains is for the file itself to be rewritten using the current schema.

DHCPSRV_MEMFILE_NEEDS_UPGRADING

version of lease file: %1 schema is earlier than version %2

> A warning message issued when the schema of the lease file loaded by the server pre-dates the memfile schema of the server. Note that the server converts the lease data from older schemas to the current schema as it is read, therefore the lease information in use by the server will be correct. What remains is for the file itself to be rewritten using the current schema.

DHCPSRV_MEMFILE_NO_STORAGE

running in non-persistent mode, leases will be lost after restart

A warning message issued when writes of leases to disk have been disabled in the configuration. This mode is useful for some kinds of performance testing but should not be enabled in normal circumstances. Non-persistence mode is enabled when 'persist4=no persist6=no' parameters are specified in the database access string.

DHCPSRV_MEMFILE_READ_HWADDR_FAIL

failed to read hardware address from lease file: %1

A warning message issued when read attempt of the hardware address stored in a disk file failed. The parameter should provide the exact nature of the failure. The database read will continue, but that particular lease will no longer have hardware address associated with it.

DHCPSRV_MEMFILE_ROLLBACK

rolling back memory file database

The code has issued a rollback call. For the memory file database, this is a no-op.

DHCPSRV_MEMFILE_UPDATE_ADDR4

updating IPv4 lease for address %1

A debug message issued when the server is attempting to update IPv4 lease from the memory file database for the specified address.

DHCPSRV_MEMFILE_UPDATE_ADDR6

updating IPv6 lease for address %1

A debug message issued when the server is attempting to update IPv6 lease from the memory file database for the specified address.

DHCPSRV_MEMFILE_WIPE_LEASES4

removing all IPv4 leases from subnet %1

This informational message is printed when removal of all leases from specified IPv4 subnet is commencing. This is a result of receiving administrative command.

DHCPSRV_MEMFILE_WIPE_LEASES4_FINISHED

removing all IPv4 leases from subnet %1 finished, removed %2 leases

This informational message is printed when removal of all leases from a specified IPv4 subnet has finished. The number of removed leases is printed.

DHCPSRV_MEMFILE_WIPE_LEASES6

removing all IPv6 leases from subnet %1

This informational message is printed when removal of all leases from specified IPv6 subnet is commencing. This is a result of receiving administrative command.

DHCPSRV_MEMFILE_WIPE_LEASES6_FINISHED

removing all IPv6 leases from subnet %1 finished, removed %2 leases

This informational message is printed when removal of all leases from a specified IPv6 subnet has finished. The number of removed leases is printed.

DHCPSRV_MULTIPLE_RAW_SOCKETS_PER_IFACE

current configuration will result in opening multiple broadcast capable sockets on some interfaces and some DHCP messages may be duplicated

A warning message issued when the current configuration indicates that multiple sockets, capable of receiving broadcast traffic, will be opened on some of the interfaces. It must be noted that this may lead to receiving and processing the same DHCP message multiple times, as it will be received by each socket individually.

DHCPSRV_MYSQL_ADD_ADDR4

adding IPv4 lease with address %1

A debug message issued when the server is about to add an IPv4 lease with the specified address to the MySQL backend database.

DHCPSRV_MYSQL_ADD_ADDR6

adding IPv6 lease with address %1, lease type %2

A debug message issued when the server is about to add an IPv6 lease with the specified address to the MySQL backend database.

DHCPSRV_MYSQL_BEGIN_TRANSACTION

committing to MySQL database

The code has issued a begin transaction call.

DHCPSRV_MYSQL_COMMIT

committing to MySQL database

The code has issued a commit call. All outstanding transactions will be committed to the database. Note that depending on the MySQL settings, the commit may not include a write to disk.

DHCPSRV_MYSQL_DB

opening MySQL lease database: %1

This informational message is logged when a DHCP server (either V4 or V6) is about to open a MySQL lease database. The parameters of the connection including database name and username needed to access it (but not the password if any) are logged.

DHCPSRV_MYSQL_DELETED_EXPIRED_RECLAIMED

deleted %1 reclaimed leases from the database

A debug message issued when the server has removed a number of reclaimed leases from the database. The number of removed leases is included in the message.

DHCPSRV_MYSQL_DELETE_ADDR

deleting lease for address %1

A debug message issued when the server is attempting to delete a lease for the specified address from the MySQL database for the specified address.

DHCPSRV_MYSQL_DELETE_EXPIRED_RECLAIMED4

deleting reclaimed IPv4 leases that expired more than %1 seconds ago

A debug message issued when the server is removing reclaimed DHCPv4 leases which have expired longer than a specified period of time. The argument is the amount of time Kea waits after a reclaimed lease expires before considering its removal.

DHCPSRV_MYSQL_DELETE_EXPIRED_RECLAIMED6

deleting reclaimed IPv6 leases that expired more than %1 seconds ago

A debug message issued when the server is removing reclaimed DHCPv6 leases which have expired longer than a specified period of time. The argument is the amount of time Kea waits after a reclaimed lease expires before considering its removal.

DHCPSRV_MYSQL_FATAL_ERROR

Unrecoverable MySQL error occurred: %1 for <%2>, reason: %3 (error code: %4).

An error message indicating that communication with the MySQL database server has been lost. If automatic recovery has been enabled, then the server will attempt to recover the connectivity. If not the server will exit with a non-zero exit code. The cause of such an error is most likely a network issue or the MySQL server has gone down.

DHCPSRV_MYSQL_GET4

obtaining all IPv4 leases

A debug message issued when the server is attempting to obtain all IPv4 leases from the MySQL database.

DHCPSRV_MYSQL_GET6

obtaining all IPv6 leases

A debug message issued when the server is attempting to obtain all IPv6 leases from the MySQL database.

DHCPSRV_MYSQL_GET_ADDR4

obtaining IPv4 lease for address %1

A debug message issued when the server is attempting to obtain an IPv4 lease from the MySQL database for the specified address.

DHCPSRV_MYSQL_GET_ADDR6

obtaining IPv6 lease for address %1, lease type %2

A debug message issued when the server is attempting to obtain an IPv6 lease from the MySQL database for the specified address.

DHCPSRV_MYSQL_GET_CLIENTID

obtaining IPv4 leases for client ID %1

A debug message issued when the server is attempting to obtain a set of IPv4 leases from the MySQL database for a client with the specified client identification.

DHCPSRV_MYSQL_GET_DUID

obtaining IPv6 lease for duid %1,

A debug message issued when the server is attempting to obtain an IPv6 lease from the MySQL database for the specified duid.

DHCPSRV_MYSQL_GET_EXPIRED4

obtaining maximum %1 of expired IPv4 leases

A debug message issued when the server is attempting to obtain expired IPv4 leases to reclaim them. The maximum number of leases to be retrieved is logged in the message.

DHCPSRV_MYSQL_GET_EXPIRED6

obtaining maximum %1 of expired IPv6 leases

A debug message issued when the server is attempting to obtain expired IPv6 leases to reclaim them. The maximum number of leases to be retrieved is logged in the message.

DHCPSRV_MYSQL_GET_HOSTNAME4

obtaining IPv4 leases for hostname %1

> A debug message issued when the server is attempting to obtain a set of IPv4 leases from the MySQL database for a client with the specified hostname.

DHCPSRV_MYSQL_GET_HOSTNAME6

obtaining IPv6 leases for hostname %1

> A debug message issued when the server is attempting to obtain a set of IPv6 leases from the MySQL database for a client with the specified hostname.

DHCPSRV_MYSQL_GET_HWADDR

obtaining IPv4 leases for hardware address %1

> A debug message issued when the server is attempting to obtain a set of IPv4 leases from the MySQL database for a client with the specified hardware address.

DHCPSRV_MYSQL_GET_IAID_DUID

obtaining IPv6 leases for IAID %1, DUID %2, lease type %3

> A debug message issued when the server is attempting to obtain a set of IPv6 leases from the MySQL database for a client with the specified IAID (Identity Association ID) and DUID (DHCP Unique Identifier).

DHCPSRV_MYSQL_GET_IAID_SUBID_DUID

obtaining IPv6 leases for IAID %1, Subnet ID %2, DUID %3, lease type %4

> A debug message issued when the server is attempting to obtain an IPv6 lease from the MySQL database for a client with the specified IAID (Identity Association ID), Subnet ID and DUID (DHCP Unique Identifier).

DHCPSRV_MYSQL_GET_PAGE4

obtaining at most %1 IPv4 leases starting from address %2

> A debug message issued when the server is attempting to obtain a page of leases beginning with the specified address.

DHCPSRV_MYSQL_GET_PAGE6

obtaining at most %1 IPv6 leases starting from address %2

> A debug message issued when the server is attempting to obtain a page of leases beginning with the specified address.

DHCPSRV_MYSQL_GET_SUBID4

obtaining IPv4 leases for subnet ID %1

> A debug message issued when the server is attempting to obtain all IPv4 leases for a given subnet identifier from the MySQL database.

DHCPSRV_MYSQL_GET_SUBID6

obtaining IPv6 leases for subnet ID %1

> A debug message issued when the server is attempting to obtain all IPv6 leases for a given subnet identifier from the MySQL database.

DHCPSRV_MYSQL_GET_SUBID_CLIENTID

obtaining IPv4 lease for subnet ID %1 and client ID %2

A debug message issued when the server is attempting to obtain an IPv4 lease from the MySQL database for a client with the specified subnet ID and client ID.

DHCPSRV_MYSQL_GET_SUBID_HWADDR

obtaining IPv4 lease for subnet ID %1 and hardware address %2

A debug message issued when the server is attempting to obtain an IPv4 lease from the MySQL database for a client with the specified subnet ID and hardware address.

DHCPSRV_MYSQL_GET_VERSION

obtaining schema version information

A debug message issued when the server is about to obtain schema version information from the MySQL database.

DHCPSRV_MYSQL_HOST_DB

opening MySQL hosts database: %1

This informational message is logged when a DHCP server (either V4 or V6) is about to open a MySQL hosts database. The parameters of the connection including database name and username needed to access it (but not the password if any) are logged.

DHCPSRV_MYSQL_HOST_DB_GET_VERSION

obtaining schema version information for the MySQL hosts database

A debug message issued when the server is about to obtain schema version information from the MySQL hosts database.

DHCPSRV_MYSQL_HOST_DB_READONLY

MySQL host database opened for read access only

This informational message is issued when the user has configured the MySQL database in read-only mode. Kea will not be able to insert or modify host reservations but will be able to retrieve existing ones and assign them to the clients communicating with the server.

DHCPSRV_MYSQL_NEGATIVE_LEASES_STAT

recount of leases returned a negative value

This warning message is issued when the recount of leases using counters in the MySQL database returned a negative value. This shows a problem which can be fixed only by an offline direct recount on the database. This message is issued only once.

DHCPSRV_MYSQL_ROLLBACK

rolling back MySQL database

The code has issued a rollback call. All outstanding transaction will be rolled back and not committed to the database.

DHCPSRV_MYSQL_START_TRANSACTION

starting new MySQL transaction

A debug message issued when a new MySQL transaction is being started. This message is typically not issued when inserting data into a single table because the server doesn't explicitly start transactions in this case. This message is issued when data is inserted into multiple tables with multiple INSERT statements and there may be a need to rollback the whole transaction if any of these INSERT statements fail.

DHCPSRV_MYSQL_UPDATE_ADDR4

updating IPv4 lease for address %1

> A debug message issued when the server is attempting to update IPv4 lease from the MySQL database for the specified address.

DHCPSRV_MYSQL_UPDATE_ADDR6

updating IPv6 lease for address %1, lease type %2

> A debug message issued when the server is attempting to update IPv6 lease from the MySQL database for the specified address.

DHCPSRV_NOTYPE_DB

no ‘type’ keyword to determine database backend: %1

> This is an error message, logged when an attempt has been made to access a database backend, but where no ‘type’ keyword has been included in the access string. The access string (less any passwords) is included in the message.

DHCPSRV_NO_SOCKETS_OPEN

no interface configured to listen to DHCP traffic

> This warning message is issued when the current server configuration specifies no interfaces that the server should listen on, or when the specified interfaces are not configured to receive the traffic.

DHCPSRV_OPEN_SOCKET_FAIL

failed to open socket: %1

> A warning message issued when IfaceMgr fails to open and bind a socket. The reason for the failure is appended as an argument of the log message.

DHCPSRV_PGSQL_ADD_ADDR4

adding IPv4 lease with address %1

> A debug message issued when the server is about to add an IPv4 lease with the specified address to the PostgreSQL backend database.

DHCPSRV_PGSQL_ADD_ADDR6

adding IPv6 lease with address %1, lease type %2

> A debug message issued when the server is about to add an IPv6 lease with the specified address to the PostgreSQL backend database.

DHCPSRV_PGSQL_BEGIN_TRANSACTION

committing to PostgreSQL database

> The code has issued a begin transaction call.

DHCPSRV_PGSQL_COMMIT

committing to PostgreSQL database

> The code has issued a commit call. All outstanding transactions will be committed to the database. Note that depending on the PostgreSQL settings, the commit may not include a write to disk.

DHCPSRV_PGSQL_DB

opening PostgreSQL lease database: %1

> This informational message is logged when a DHCP server (either V4 or V6) is about to open a PostgreSQL lease database. The parameters of the connection including database name and username needed to access it (but not the password if any) are logged.

DHCPSRV_PGSQL_DEALLOC_ERROR

An error occurred deallocating SQL statements while closing the PostgreSQL lease database: %1

> This is an error message issued when a DHCP server (either V4 or V6) experienced and error freeing database SQL resources as part of closing its connection to the PostgreSQL database. The connection is closed as part of normal server shutdown. This error is most likely a programmatic issue that is highly unlikely to occur or negatively impact server operation.

DHCPSRV_PGSQL_DELETE_ADDR

deleting lease for address %1

> A debug message issued when the server is attempting to delete a lease for the specified address from the PostgreSQL database for the specified address.

DHCPSRV_PGSQL_DELETE_EXPIRED_RECLAIMED4

deleting reclaimed IPv4 leases that expired more than %1 seconds ago

> A debug message issued when the server is removing reclaimed DHCPv4 leases which have expired longer than a specified period of time. The argument is the amount of time Kea waits after a reclaimed lease expires before considering its removal.

DHCPSRV_PGSQL_DELETE_EXPIRED_RECLAIMED6

deleting reclaimed IPv6 leases that expired more than %1 seconds ago

> A debug message issued when the server is removing reclaimed DHCPv6 leases which have expired longer than a specified period of time. The argument is the amount of time Kea waits after a reclaimed lease expires before considering its removal.

DHCPSRV_PGSQL_FATAL_ERROR

Unrecoverable PostgreSQL error occurred: Statement: <%1>, reason: %2 (error code: %3).

> An error message indicating that communication with the PostgreSQL database server has been lost. If automatic recovery has been enabled, then the server will attempt to recover the connectivity. If not the server will exit with a non-zero exit code. The cause of such an error is most likely a network issue or the PostgreSQL server has gone down.

DHCPSRV_PGSQL_GET4

obtaining all IPv4 leases

> A debug message issued when the server is attempting to obtain all IPv4 leases from the PostgreSQL database.

DHCPSRV_PGSQL_GET6

obtaining all IPv6 leases

> A debug message issued when the server is attempting to obtain all IPv6 leases from the PostgreSQL database.

DHCPSRV_PGSQL_GET_ADDR4

obtaining IPv4 lease for address %1

> A debug message issued when the server is attempting to obtain an IPv4 lease from the PostgreSQL database for the specified address.

DHCPSRV_PGSQL_GET_ADDR6

obtaining IPv6 lease for address %1 (lease type %2)

A debug message issued when the server is attempting to obtain an IPv6 lease from the PostgreSQL database for the specified address.

DHCPSRV_PGSQL_GET_CLIENTID

obtaining IPv4 leases for client ID %1

A debug message issued when the server is attempting to obtain a set of IPv4 leases from the PostgreSQL database for a client with the specified client identification.

DHCPSRV_PGSQL_GET_DUID

obtaining IPv6 leases for DUID %1,

A debug message issued when the server is attempting to obtain a set of IPv6 leases from the PostgreSQL database for a client with the specified DUID (DHCP Unique Identifier).

DHCPSRV_PGSQL_GET_EXPIRED4

obtaining maximum %1 of expired IPv4 leases

A debug message issued when the server is attempting to obtain expired IPv4 leases to reclaim them. The maximum number of leases to be retrieved is logged in the message.

DHCPSRV_PGSQL_GET_EXPIRED6

obtaining maximum %1 of expired IPv6 leases

A debug message issued when the server is attempting to obtain expired IPv6 leases to reclaim them. The maximum number of leases to be retrieved is logged in the message.

DHCPSRV_PGSQL_GET_HOSTNAME4

obtaining IPv4 leases for hostname %1

A debug message issued when the server is attempting to obtain a set of IPv4 leases from the PostgreSQL database for a client with the specified hostname.

DHCPSRV_PGSQL_GET_HOSTNAME6

obtaining IPv6 leases for hostname %1

A debug message issued when the server is attempting to obtain a set of IPv6 leases from the PostgreSQL database for a client with the specified hostname.

DHCPSRV_PGSQL_GET_HWADDR

obtaining IPv4 leases for hardware address %1

A debug message issued when the server is attempting to obtain a set of IPv4 leases from the PostgreSQL database for a client with the specified hardware address.

DHCPSRV_PGSQL_GET_IAID_DUID

obtaining IPv4 leases for IAID %1 and DUID %2, lease type %3

A debug message issued when the server is attempting to obtain a set of IPv6 leases from the PostgreSQL database for a client with the specified IAID (Identity Association ID) and DUID (DHCP Unique Identifier).

DHCPSRV_PGSQL_GET_IAID_SUBID_DUID

obtaining IPv4 leases for IAID %1, Subnet ID %2, DUID %3, and lease type %4

A debug message issued when the server is attempting to obtain an IPv6 lease from the PostgreSQL database for a client with the specified IAID (Identity Association ID), Subnet ID and DUID (DHCP Unique Identifier).

DHCPSRV_PGSQL_GET_PAGE4

obtaining at most %1 IPv4 leases starting from address %2

> A debug message issued when the server is attempting to obtain a page of leases beginning with the specified address.

DHCPSRV_PGSQL_GET_PAGE6

obtaining at most %1 IPv6 leases starting from address %2

> A debug message issued when the server is attempting to obtain a page of leases beginning with the specified address.

DHCPSRV_PGSQL_GET_SUBID4

obtaining IPv4 leases for subnet ID %1

> A debug message issued when the server is attempting to obtain all IPv4 leases for a given subnet identifier from the PostgreSQL database.

DHCPSRV_PGSQL_GET_SUBID6

obtaining IPv6 leases for subnet ID %1

> A debug message issued when the server is attempting to obtain all IPv6 leases for a given subnet identifier from the PostgreSQL database.

DHCPSRV_PGSQL_GET_SUBID_CLIENTID

obtaining IPv4 lease for subnet ID %1 and client ID %2

> A debug message issued when the server is attempting to obtain an IPv4 lease from the PostgreSQL database for a client with the specified subnet ID and client ID.

DHCPSRV_PGSQL_GET_SUBID_HWADDR

obtaining IPv4 lease for subnet ID %1 and hardware address %2

> A debug message issued when the server is attempting to obtain an IPv4 lease from the PostgreSQL database for a client with the specified subnet ID and hardware address.

DHCPSRV_PGSQL_GET_VERSION

obtaining schema version information

> A debug message issued when the server is about to obtain schema version information from the PostgreSQL database.

DHCPSRV_PGSQL_HOST_DB

opening PostgreSQL hosts database: %1

> This informational message is logged when a DHCP server (either V4 or V6) is about to open a PostgreSQL hosts database. The parameters of the connection including database name and username needed to access it (but not the password if any) are logged.

DHCPSRV_PGSQL_HOST_DB_GET_VERSION

obtaining schema version information for the PostgreSQL hosts database

> A debug message issued when the server is about to obtain schema version information from the PostgreSQL hosts database.

DHCPSRV_PGSQL_HOST_DB_READONLY

PostgreSQL host database opened for read access only

This informational message is issued when the user has configured the PostgreSQL database in read-only mode. Kea will not be able to insert or modify host reservations but will be able to retrieve existing ones and assign them to the clients communicating with the server.

DHCPSRV_PGSQL_NEGATIVE_LEASES_STAT

recount of leases returned a negative value

This warning message is issued when the recount of leases using counters in the PostgreSQL database returned a negative value. This shows a problem which can be fixed only by an offline direct recount on the database. This message is issued only once.

DHCPSRV_PGSQL_ROLLBACK

rolling back PostgreSQL database

The code has issued a rollback call. All outstanding transaction will be rolled back and not committed to the database.

DHCPSRV_PGSQL_START_TRANSACTION

starting a new PostgreSQL transaction

A debug message issued when a new PostgreSQL transaction is being started. This message is typically not issued when inserting data into a single table because the server doesn't explicitly start transactions in this case. This message is issued when data is inserted into multiple tables with multiple INSERT statements and there may be a need to rollback the whole transaction if any of these INSERT statements fail.

DHCPSRV_PGSQL_UPDATE_ADDR4

updating IPv4 lease for address %1

A debug message issued when the server is attempting to update IPv4 lease from the PostgreSQL database for the specified address.

DHCPSRV_PGSQL_UPDATE_ADDR6

updating IPv6 lease for address %1, lease type %2

A debug message issued when the server is attempting to update IPv6 lease from the PostgreSQL database for the specified address.

DHCPSRV_QUEUE_NCR

%1: name change request to %2 DNS entry queued: %3

A debug message which is logged when the NameChangeRequest to add or remove a DNS entries for a particular lease has been queued. The first argument includes the client identification information. The second argument indicates whether the DNS entry is to be added or removed. The third argument carries the details of the NameChangeRequest.

DHCPSRV_QUEUE_NCR_FAILED

%1: queuing %2 name change request failed for lease %3: %4

This error message is logged when sending a name change request to DHCP DDNS failed. The first argument includes the client identification information. The second argument indicates whether the DNS entry is to be added or removed. The third argument specifies the leased address. The last argument provides the reason for failure.

DHCPSRV_QUEUE_NCR_SKIP

%1: skip queuing name change request for lease: %2

This debug message is issued when the server decides to not queue the name change request because the lease doesn't include the FQDN, the forward and reverse update is disabled for this lease or the DNS updates are disabled in the configuration. The first argument includes the client identification information. The second argument includes the leased address.

DHCPSRV_TIMERMGR_CALLBACK_FAILED

running handler for timer %1 caused exception: %2

This error message is emitted when the timer elapsed and the operation associated with this timer has thrown an exception. The timer name and the reason for exception is logged.

DHCPSRV_TIMERMGR_REGISTER_TIMER

registering timer: %1, using interval: %2 ms

A debug message issued when the new interval timer is registered in the Timer Manager. This timer will have a callback function associated with it, and this function will be executed according to the interval specified. The unique name of the timer and the interval at which the callback function will be executed is included in the message.

DHCPSRV_TIMERMGR_RUN_TIMER_OPERATION

running operation for timer: %1

A debug message issued when the Timer Manager is about to run a periodic operation associated with the given timer. An example of such operation is a periodic cleanup of expired leases. The name of the timer is included in the message.

DHCPSRV_TIMERMGR_START_TIMER

starting timer: %1

A debug message issued when the registered interval timer is being started. If this operation is successful the timer will periodically execute the operation associated with it. The name of the started timer is included in the message.

DHCPSRV_TIMERMGR_STOP_TIMER

stopping timer: %1

A debug message issued when the registered interval timer is being stopped. The timer remains registered and can be restarted if necessary. The name of the timer is included in the message.

DHCPSRV_TIMERMGR_UNREGISTER_ALL_TIMERS

unregistering all timers

A debug message issued when all registered interval timers are being unregistered from the Timer Manager.

DHCPSRV_TIMERMGR_UNREGISTER_TIMER

unregistering timer: %1

A debug message issued when one of the registered interval timers is unregistered from the Timer Manager. The name of the timer is included in the message.

DHCPSRV_UNEXPECTED_NAME

database access parameters passed through '%1', expected 'lease-database'

The parameters for access the lease database were passed to the server through the named configuration parameter, but the code was expecting them to be passed via the parameter named "lease-database". If the database opens successfully, there is no impact on server operation. However, as this does indicate an error in the source code, please submit a bug report.

25.11 DHCP

DHCP_DDNS_ADD_FAILED

DHCP_DDNS Request ID %1: Transaction outcome %2

> This is an error message issued after DHCP_DDNS attempts to submit DNS mapping entry additions have failed. The precise reason for the failure should be documented in preceding log entries.

DHCP_DDNS_ADD_SUCCEEDED

DHCP_DDNS Request ID %1: successfully added the DNS mapping addition for this request: %2

> This is an informational message issued after DHCP_DDNS has submitted DNS mapping additions which were received and accepted by an appropriate DNS server.

DHCP_DDNS_ALREADY_RUNNING

%1 already running? %2

> This is an error message that occurs when DHCP_DDNS encounters a pre-existing PID file which contains the PID of a running process. This most likely indicates an attempt to start a second instance of DHCP_DDNS using the same configuration file. It is possible, though unlikely, that the PID file is a remnant left behind by a server crash or power failure and the PID it contains refers to a process other than DHCP_DDNS. In such an event, it would be necessary to manually remove the PID file. The first argument is the DHCP_DDNS process name, the second contains the PID and PID file.

DHCP_DDNS_AT_MAX_TRANSACTIONS

application has %1 queued requests but has reached maximum number of %2 concurrent transactions

> This is a debug message that indicates that the application has DHCP_DDNS requests in the queue but is working as many concurrent requests as allowed.

DHCP_DDNS_CLEARED_FOR_SHUTDOWN

application has met shutdown criteria for shutdown type: %1

> This is a debug message issued when the application has been instructed to shutdown and has met the required criteria to exit.

DHCP_DDNS_COMMAND

command directive received, command: %1 - args: %2

> This is a debug message issued when the DHCP-DDNS application command method has been invoked.

DHCP_DDNS_CONFIGURE

configuration %1 received: %2

> This is a debug message issued when the DHCP-DDNS application configure method has been invoked.

DHCP_DDNS_CONFIG_CHECK_FAIL

DHCP-DDNS server configuration check failed: %1

> This error message indicates that the DHCP-DDNS had failed configuration check. Details are provided. Additional details may be available in earlier log entries, possibly on lower levels.

DHCP_DDNS_CONFIG_FAIL

DHCP-DDNS server configuration failed: %1

> This error message indicates that the DHCP-DDNS had failed configuration attempt. Details are provided. Additional details may be available in earlier log entries, possibly on lower levels.

DHCP_DDNS_FAILED

application experienced a fatal error: %1

> This is a debug message issued when the DHCP-DDNS application encounters an unrecoverable error from within the event loop.

DHCP_DDNS_FORWARD_ADD_BAD_DNSCLIENT_STATUS

DHCP_DDNS Request ID %1: received an unknown DNSClient status: %2, while adding a forward address mapping for FQDN %3 to DNS server %4

> This is an error message issued when DNSClient returns an unrecognized status while DHCP_DDNS was adding a forward address mapping. The request will be aborted. This is most likely a programmatic issue and should be reported.

DHCP_DDNS_FORWARD_ADD_BUILD_FAILURE

DNS Request ID %1: update message to add a forward DNS entry could not be constructed for this request: %2, reason: %3

> This is an error message issued when an error occurs attempting to construct the server bound packet requesting a forward address addition. This is due to invalid data contained in the NameChangeRequest. The request will be aborted. This is most likely a configuration issue.

DHCP_DDNS_FORWARD_ADD_IO_ERROR

DHCP_DDNS Request ID %1: encountered an IO error sending a forward mapping add for FQDN %2 to DNS server %3

> This is an error message issued when a communication error occurs while DHCP_DDNS is carrying out a forward address update. The application will retry against the same server or others as appropriate.

DHCP_DDNS_FORWARD_ADD_REJECTED

DNS Request ID %1: Server, %2, rejected a DNS update request to add the address mapping for FQDN, %3, with an RCODE: %4

> This is an error message issued when an update was rejected by the DNS server it was sent to for the reason given by the RCODE. The rcode values are defined in RFC 2136.

DHCP_DDNS_FORWARD_ADD_RESP_CORRUPT

DHCP_DDNS Request ID %1: received a corrupt response from the DNS server, %2, while adding forward address mapping for FQDN, %3

> This is an error message issued when the response received by DHCP_DDNS, to a update request to add a forward address mapping, is mangled or malformed. The application will retry against the same server or others as appropriate.

DHCP_DDNS_FORWARD_REMOVE_ADDRS_BAD_DNSCLIENT_STATUS

DHCP_DDNS Request ID %1: received an unknown DNSClient status: %2, while removing a forward address mapping for FQDN %3 to DNS server %4

> This is an error message issued when DNSClient returns an unrecognized status while DHCP_DDNS was removing a forward address mapping. The request will be aborted. This is most likely a programmatic issue and should be reported.

DHCP_DDNS_FORWARD_REMOVE_ADDRS_BUILD_FAILURE

DNS Request ID %1: update message to remove a forward DNS Address entry could not be constructed for this request: %2, reason: %3

This is an error message issued when an error occurs attempting to construct the server bound packet requesting a forward address (A or AAAA) removal. This is due to invalid data contained in the NameChangeRequest. The request will be aborted. This is most likely a configuration issue. */sar/*

DHCP_DDNS_FORWARD_REMOVE_ADDRS_IO_ERROR

DHCP_DDNS Request ID %1: encountered an IO error sending a forward mapping address removal for FQDN %2 to DNS server %3

This is an error message issued when a communication error occurs while DHCP_DDNS is carrying out a forward address remove. The application will retry against the same server or others as appropriate.

DHCP_DDNS_FORWARD_REMOVE_ADDRS_REJECTED

DNS Request ID %1: Server, %2, rejected a DNS update request to remove the forward address mapping for FQDN, %3, with an RCODE: %4

This is an error message issued when an update was rejected by the DNS server it was sent to for the reason given by the RCODE. The rcode values are defined in RFC 2136.

DHCP_DDNS_FORWARD_REMOVE_ADDRS_RESP_CORRUPT

DHCP_DDNS Request ID %1: received a corrupt response from the DNS server, %2, while removing forward address mapping for FQDN, %3

This is an error message issued when the response received by DHCP_DDNS, to a update request to remove a forward address mapping, is mangled or malformed. The application will retry against the same server or others as appropriate.

DHCP_DDNS_FORWARD_REMOVE_RRS_BAD_DNSCLIENT_STATUS

DHCP_DDNS Request ID %1: received an unknown DNSClient status: %2, while removing forward RRs for FQDN %3 to DNS server %4

This is an error message issued when DNSClient returns an unrecognized status while DHCP_DDNS was removing forward RRs. The request will be aborted. This is most likely a programmatic issue and should be reported.

DHCP_DDNS_FORWARD_REMOVE_RRS_BUILD_FAILURE

DNS Request ID %1: update message to remove forward DNS RR entries could not be constructed for this request: %2, reason: %3

This is an error message issued when an error occurs attempting to construct the server bound packet requesting forward RR (DHCID RR) removal. This is due to invalid data contained in the NameChangeRequest. The request will be aborted. This is most likely a configuration issue.

DHCP_DDNS_FORWARD_REMOVE_RRS_IO_ERROR

DHCP_DDNS Request ID %1: encountered an IO error sending a forward RR removal for FQDN %2 to DNS server %3

This is an error message issued when a communication error occurs while DHCP_DDNS is carrying out a forward RR remove. The application will retry against the same server.

DHCP_DDNS_FORWARD_REMOVE_RRS_REJECTED

DNS Request ID %1: Server, %2, rejected a DNS update request to remove forward RR entries for FQDN, %3, with an RCODE: %4

This is an error message issued when an update was rejected by the DNS server it was sent to for the reason given by the RCODE. The rcode values are defined in RFC 2136.

DHCP_DDNS_FORWARD_REMOVE_RRS_RESP_CORRUPT

DHCP_DDNS Request ID %1: received a corrupt response from the DNS server, %2, while removing forward RRs for FQDN, %3

> This is an error message issued when the response received by DHCP_DDNS, to a update request to remove forward RRs mapping, is mangled or malformed. The application will retry against the same server or others as appropriate. */sar/*

DHCP_DDNS_FORWARD_REPLACE_BAD_DNSCLIENT_STATUS

DHCP_DDNS Request ID %1: received an unknown DNSClient status: %2, while replacing forward address mapping for FQDN %3 to DNS server %4

> This is an error message issued when DNSClient returns an unrecognized status while DHCP_DDNS was replacing a forward address mapping. The request will be aborted. This is most likely a programmatic issue and should be reported.

DHCP_DDNS_FORWARD_REPLACE_BUILD_FAILURE

DNS Request ID %1: update message to replace a forward DNS entry could not be constructed from this request: %2, reason: %3

> This is an error message issued when an error occurs attempting to construct the server bound packet requesting a forward address replacement. This is due to invalid data contained in the NameChangeRequest. The request will be aborted. This is most likely a configuration issue.

DHCP_DDNS_FORWARD_REPLACE_IO_ERROR

DHCP_DDNS Request ID %1: encountered an IO error sending a forward mapping replace for FQDN %2 to DNS server %3

> This is an error message issued when a communication error occurs while DHCP_DDNS is carrying out a forward address update. The application will retry against the same server or others as appropriate.

DHCP_DDNS_FORWARD_REPLACE_REJECTED

DNS Request ID %1: Server, %2, rejected a DNS update request to replace the address mapping for FQDN, %3, with an RCODE: %4

> This is an error message issued when an update was rejected by the DNS server it was sent to for the reason given by the RCODE. The rcode values are defined in RFC 2136.

DHCP_DDNS_FORWARD_REPLACE_RESP_CORRUPT

DHCP_DDNS Request ID %1: received a corrupt response from the DNS server, %2, while replacing forward address mapping for FQDN, %3

> This is an error message issued when the response received by DHCP_DDNS, to a update request to replace a forward address mapping, is mangled or malformed. The application will retry against the same server or others as appropriate.

DHCP_DDNS_FWD_REQUEST_IGNORED

Request ID %1: Forward updates are disabled, the forward portion of request will be ignored: %2

> This is a debug message issued when forward DNS updates are disabled and DHCP_DDNS receives an update request containing a forward DNS update. The forward update will not performed.

DHCP_DDNS_INVALID_NCR

application received an invalid DNS update request: %1

> This is an error message that indicates that an invalid request to update a DNS entry was received by the application. Either the format or the content of the request is incorrect. The request will be ignored.

DHCP_DDNS_INVALID_RESPONSE

received response to DNS Update message is malformed: %1

> This is a debug message issued when the DHCP-DDNS application encountered an error while decoding a response to DNS Update message. Typically, this error will be encountered when a response message is malformed.

DHCP_DDNS_NCR_FLUSH_IO_ERROR

DHCP-DDNS Last send before stopping did not complete successfully: %1

> This is an error message that indicates the DHCP-DDNS client was unable to complete the last send prior to exiting send mode. This is a programmatic error, highly unlikely to occur, and should not impair the application's ability to process requests.

DHCP_DDNS_NCR_LISTEN_CLOSE_ERROR

application encountered an error while closing the listener used to receive NameChangeRequests : %1

> This is an error message that indicates the application was unable to close the listener connection used to receive NameChangeRequests. Closure may occur during the course of error recovery or during normal shutdown procedure. In either case the error is unlikely to impair the application's ability to process requests but it should be reported for analysis.

DHCP_DDNS_NCR_RECV_NEXT_ERROR

application could not initiate the next read following a request receive.

> This is a error message indicating that NameChangeRequest listener could not start another read after receiving a request. While possible, this is highly unlikely and is probably a programmatic error. The application should recover on its own.

DHCP_DDNS_NCR_SEND_CLOSE_ERROR

DHCP-DDNS client encountered an error while closing the sender connection used to send NameChangeRequests: %1

> This is an error message that indicates the DHCP-DDNS client was unable to close the connection used to send NameChangeRequests. Closure may occur during the course of error recovery or during normal shutdown procedure. In either case the error is unlikely to impair the client's ability to send requests but it should be reported for analysis.

DHCP_DDNS_NCR_SEND_NEXT_ERROR

DHCP-DDNS client could not initiate the next request send following send completion: %1

> This is a error message indicating that NameChangeRequest sender could not start another send after completing the send of the previous request. While possible, this is highly unlikely and is probably a programmatic error. The application should recover on its own.

DHCP_DDNS_NCR_UDP_CLEAR_READY_ERROR

NCR UDP watch socket failed to clear: %1

> This is an error message that indicates the application was unable to reset the UDP NCR sender ready status after completing a send. This is programmatic error that should be reported. The application may or may not continue to operate correctly.

DHCP_DDNS_NCR_UDP_RECV_CANCELED

UDP socket receive was canceled while listening for DNS Update requests

> This is a debug message indicating that the listening on a UDP socket for DNS update requests has been canceled. This is a normal part of suspending listening operations.

DHCP_DDNS_NCR_UDP_RECV_ERROR

UDP socket receive error while listening for DNS Update requests: %1

> This is an error message indicating that an I/O error occurred while listening over a UDP socket for DNS update requests. This could indicate a network connectivity or system resource issue.

DHCP_DDNS_NCR_UDP_SEND_CANCELED

UDP socket send was canceled while sending a DNS Update request to DHCP_DDNS: %1

> This is an informational message indicating that sending requests via UDP socket to DHCP_DDNS has been interrupted. This is a normal part of suspending send operations.

DHCP_DDNS_NCR_UDP_SEND_ERROR

UDP socket send error while sending a DNS Update request: %1

> This is an error message indicating that an IO error occurred while sending a DNS update request to DHCP_DDNS over a UDP socket. This could indicate a network connectivity or system resource issue.

DHCP_DDNS_NOT_ON_LOOPBACK

the DHCP-DDNS server has been configured to listen on %1 which is not the local loopback. This is an insecure configuration supported for testing purposes only

> This is a warning message issued when the DHCP-DDNS server is configured to listen at an address other than the loopback address (127.0.0.1 or ::1). It is possible for a malicious attacker to send bogus NameChangeRequests to it and change entries in the DNS. For this reason, addresses other than the IPv4 or IPv6 loopback addresses should only be used for testing purposes. A future version of Kea will implement authentication to guard against such attacks.

DHCP_DDNS_NO_ELIGIBLE_JOBS

although there are queued requests, there are pending transactions for each, Queue count: %1 Transaction count: %2

> This is a debug message issued when all of the queued requests represent clients for which there is a an update already in progress. This may occur under normal operations but should be temporary situation.

DHCP_DDNS_NO_FWD_MATCH_ERROR

Request ID %1: the configured list of forward DDNS domains does not contain a match for: %2 The request has been discarded.

> This is an error message that indicates that DHCP_DDNS received a request to update a the forward DNS information for the given FQDN but for which there are no configured DDNS domains in the DHCP_DDNS configuration. Either the DHCP_DDNS configuration needs to be updated or the source of the FQDN itself should be investigated.

DHCP_DDNS_NO_MATCH

No DNS servers match FQDN %1

> This is warning message issued when there are no domains in the configuration which match the cited fully qualified domain name (FQDN). The DNS Update request for the FQDN cannot be processed.

DHCP_DDNS_NO_REV_MATCH_ERROR

Request ID %1: the configured list of reverse DDNS domains does not contain a match for: %2 The request has been discarded.

> This is an error message that indicates that DHCP_DDNS received a request to update a the reverse DNS information for the given FQDN but for which there are no configured DDNS domains in the DHCP_DDNS configuration. Either the DHCP_DDNS configuration needs to be updated or the source of the FQDN itself should be investigated.

DHCP_DDNS_PROCESS_INIT

application init invoked

> This is a debug message issued when the DHCP-DDNS application enters its initialization method.

DHCP_DDNS_QUEUE_MGR_QUEUE_FULL

application request queue has reached maximum number of entries %1

> This an error message indicating that DHCP-DDNS is receiving DNS update requests faster than they can be processed. This may mean the maximum queue needs to be increased, the DHCP-DDNS clients are simply generating too many requests too quickly, or perhaps upstream DNS servers are experiencing load issues.

DHCP_DDNS_QUEUE_MGR_QUEUE_RECEIVE

Request ID %1: received and queued a request.

> This is an informational message indicating that the NameChangeREquest listener used by DHCP-DDNS to receive a request has received a request and queued it for further processing.

DHCP_DDNS_QUEUE_MGR_RECONFIGURING

application is reconfiguring the queue manager

> This is an informational message indicating that DHCP_DDNS is reconfiguring the queue manager as part of normal startup or in response to a new configuration.

DHCP_DDNS_QUEUE_MGR_RECOVERING

application is attempting to recover from a queue manager IO error

> This is an informational message indicating that DHCP_DDNS is attempting to restart the queue manager after it suffered an IO error while receiving requests.

DHCP_DDNS_QUEUE_MGR_RECV_ERROR

application's queue manager was notified of a request receive error by its listener.

> This is an error message indicating that the NameChangeRequest listener used by DHCP-DDNS to receive requests encountered an IO error. There should be corresponding log messages from the listener layer with more details. This may indicate a network connectivity or system resource issue.

DHCP_DDNS_QUEUE_MGR_RESUME_ERROR

application could not restart the queue manager, reason: %1

> This is an error message indicating that DHCP_DDNS's Queue Manager could not be restarted after stopping due to a full receive queue. This means that the application cannot receive requests. This is most likely due to DHCP_DDNS configuration parameters referring to resources such as an IP address or port, that is no longer unavailable. DHCP_DDNS will attempt to restart the queue manager if given a new configuration.

DHCP_DDNS_QUEUE_MGR_RESUMING

application is resuming listening for requests now that the request queue size has reached %1 of a maximum %2 allowed

> This is an informational message indicating that DHCP_DDNS, which had stopped accepting new requests, has processed enough entries from the receive queue to resume accepting requests.

DHCP_DDNS_QUEUE_MGR_STARTED

application's queue manager has begun listening for requests.

This is a debug message indicating that DHCP_DDNS's Queue Manager has successfully started and is now listening for NameChangeRequests.

DHCP_DDNS_QUEUE_MGR_START_ERROR

application could not start the queue manager, reason: %1

> This is an error message indicating that DHCP_DDNS's Queue Manager could not be started. This means that the application cannot receive requests. This is most likely due to DHCP_DDNS configuration parameters referring to resources such as an IP address or port, that are unavailable. DHCP_DDNS will attempt to restart the queue manager if given a new configuration.

DHCP_DDNS_QUEUE_MGR_STOPPED

application's queue manager has stopped listening for requests.

> This is a debug message indicating that DHCP_DDNS's Queue Manager has stopped listening for NameChangeRequests. This may be because of normal event such as reconfiguration or as a result of an error. There should be log messages preceding this one to indicate why it has stopped.

DHCP_DDNS_QUEUE_MGR_STOPPING

application is stopping the queue manager for %1

> This is an informational message indicating that DHCP_DDNS is stopping the queue manager either to reconfigure it or as part of application shutdown.

DHCP_DDNS_QUEUE_MGR_STOP_ERROR

application encountered an error stopping the queue manager: %1

> This is an error message indicating that DHCP_DDNS encountered an error while trying to stop the queue manager. This error is unlikely to occur or to impair the application's ability to function but it should be reported for analysis.

DHCP_DDNS_QUEUE_MGR_UNEXPECTED_HANDLER_ERROR

application's queue manager request receive handler experienced an unexpected exception %1:

> This is an error message indicating that an unexpected error occurred within the DHCP_DDNS's Queue Manager request receive completion handler. This is most likely a programmatic issue that should be reported. The application may recover on its own.

DHCP_DDNS_QUEUE_MGR_UNEXPECTED_STOP

application's queue manager receive was

> aborted unexpectedly while queue manager state is: %1 This is an error message indicating that DHCP_DDNS's Queue Manager request receive was unexpected interrupted. Normally, the read is receive is only interrupted as a normal part of stopping the queue manager. This is most likely a programmatic issue that should be reported.

DHCP_DDNS_REMOVE_FAILED

DHCP_DDNS Request ID %1: Transaction outcome: %2

> This is an error message issued after DHCP_DDNS attempts to submit DNS mapping entry removals have failed. The precise reason for the failure should be documented in preceding log entries.

DHCP_DDNS_REMOVE_SUCCEEDED

DHCP_DDNS Request ID %1: successfully removed the DNS mapping addition for this request: %2

> This is an informational message issued after DHCP_DDNS has submitted DNS mapping removals which were received and accepted by an appropriate DNS server.

DHCP_DDNS_REQUEST_DROPPED

Request ID %1: Request contains no enabled update requests and will be dropped: %2

> This is a debug message issued when DHCP_DDNS receives a request which does not contain updates in a direction that is enabled. In other words, if only forward updates are enabled and request is received that asks only for reverse updates then the request is dropped.

DHCP_DDNS_REVERSE_REMOVE_BAD_DNSCLIENT_STATUS

DHCP_DDNS Request ID %1: received an unknown DNSClient status: %2, while removing reverse address mapping for FQDN %3 to DNS server %4

> This is an error message issued when DNSClient returns an unrecognized status while DHCP_DDNS was removing a reverse address mapping. The request will be aborted. This is most likely a programmatic issue and should be reported.

DHCP_DDNS_REVERSE_REMOVE_BUILD_FAILURE

DNS Request ID %1: update message to remove a reverse DNS entry could not be constructed from this request: %2, reason: %3

> This is an error message issued when an error occurs attempting to construct the server bound packet requesting a reverse PTR removal. This is due to invalid data contained in the NameChangeRequest. The request will be aborted. This is most likely a configuration issue.

DHCP_DDNS_REVERSE_REMOVE_IO_ERROR

DHCP_DDNS Request ID %1: encountered an IO error sending a reverse mapping remove for FQDN %2 to DNS server %3

> This is an error message issued when a communication error occurs while DHCP_DDNS is carrying out a reverse address update. The application will retry against the same server or others as appropriate.

DHCP_DDNS_REVERSE_REMOVE_REJECTED

DNS Request ID %1: Server, %2, rejected a DNS update request to remove the reverse mapping for FQDN, %3, with an RCODE: %4

> This is an error message issued when an update was rejected by the DNS server it was sent to for the reason given by the RCODE. The rcode values are defined in RFC 2136.

DHCP_DDNS_REVERSE_REMOVE_RESP_CORRUPT

DHCP_DDNS Request ID %1: received a corrupt response from the DNS server, %2, while removing reverse address mapping for FQDN, %3

> This is an error message issued when the response received by DHCP_DDNS, to a update request to remove a reverse address, is mangled or malformed. The application will retry against the same server or others as appropriate.

DHCP_DDNS_REVERSE_REPLACE_BAD_DNSCLIENT_STATUS

DHCP_DDNS Request ID %1: received an unknown DNSClient status: %2, while replacing reverse address mapping for FQDN %3 to DNS server %4

> This is an error message issued when DNSClient returns an unrecognized status while DHCP_DDNS was replacing a reverse address mapping. The request will be aborted. This is most likely a programmatic issue and should be reported.

DHCP_DDNS_REVERSE_REPLACE_BUILD_FAILURE

DNS Request ID %1: update message to replace a reverse DNS entry could not be constructed from this request: %2, reason: %3

This is an error message issued when an error occurs attempting to construct the server bound packet requesting a reverse PTR replacement. This is due to invalid data contained in the NameChangeRequest. The request will be aborted. This is most likely a configuration issue.

DHCP_DDNS_REVERSE_REPLACE_IO_ERROR

DHCP_DDNS Request ID %1: encountered an IO error sending a reverse mapping replacement for FQDN %2 to DNS server %3

This is an error message issued when a communication error occurs while DHCP_DDNS is carrying out a reverse address update. The application will retry against the same server or others as appropriate.

DHCP_DDNS_REVERSE_REPLACE_REJECTED

DNS Request ID %1: Server, %2, rejected a DNS update request to replace the reverse mapping for FQDN, %3, with an RCODE: %4

This is an error message issued when an update was rejected by the DNS server it was sent to for the reason given by the RCODE. The rcode values are defined in RFC 2136.

DHCP_DDNS_REVERSE_REPLACE_RESP_CORRUPT

DHCP_DDNS Request ID %1: received a corrupt response from the DNS server, %2, while replacing reverse address mapping for FQDN, %3

This is an error message issued when the response received by DHCP_DDNS, to a update request to replace a reverse address, is mangled or malformed. The application will retry against the same server or others as appropriate.

DHCP_DDNS_REV_REQUEST_IGNORED

Request ID %1: Reverse updates are disabled, the reverse portion of request will be ignored: %2

This is a debug message issued when reverse DNS updates are disabled and DHCP_DDNS receives an update request containing a reverse DNS update. The reverse update will not performed.

DHCP_DDNS_RUN_EXIT

application is exiting the event loop

This is a debug message issued when the DHCP-DDNS server exits its event lo

DHCP_DDNS_SHUTDOWN_COMMAND

application received shutdown command with args: %1

This is a debug message issued when the application has been instructed to shut down by the controller.

DHCP_DDNS_STARTED

Kea DHCP-DDNS server version %1 started

This informational message indicates that the DHCP-DDNS server has processed all configuration information and is ready to begin processing. The version is also printed.

DHCP_DDNS_STARTING_TRANSACTION

Request ID %1:

This is a debug message issued when DHCP-DDNS has begun a transaction for a given request.

DHCP_DDNS_STATE_MODEL_UNEXPECTED_ERROR

Request ID %1: application encountered an unexpected error while carrying out a NameChangeRequest: %2

> This is error message issued when the application fails to process a NameChangeRequest correctly. Some or all of the DNS updates requested as part of this update did not succeed. This is a programmatic error and should be reported.

DHCP_DDNS_TRANS_SEND_ERROR

Request ID %1: application encountered an unexpected error while attempting to send a DNS update: %2

> This is error message issued when the application is able to construct an update message but the attempt to send it suffered an unexpected error. This is most likely a programmatic error, rather than a communications issue. Some or all of the DNS updates requested as part of this request did not succeed.

DHCP_DDNS_UDP_SENDER_WATCH_SOCKET_CLOSE_ERROR

watch socket failed to close: %1

> This is an error message that indicates the application was unable to close the inbound or outbound side of a NCR sender's watch socket. While technically possible the error is highly unlikely to occur and should not impair the application's ability to process requests.

DHCP_DDNS_UNCAUGHT_NCR_RECV_HANDLER_ERROR

unexpected exception thrown from the application receive completion handler: %1

> This is an error message that indicates that an exception was thrown but not caught in the application's request receive completion handler. This is a programmatic error that needs to be reported. Dependent upon the nature of the error the application may or may not continue operating normally.

DHCP_DDNS_UPDATE_REQUEST_SENT

Request ID %1: %2 to server: %3

> This is a debug message issued when DHCP_DDNS sends a DNS request to a DNS server.

25.12 EVAL

EVAL_DEBUG_AND

Popping %1 and %2 pushing %3

> This debug message indicates that two values are popped from the value stack. Then are then combined via logical and and the result is pushed onto the value stack.

EVAL_DEBUG_CONCAT

Popping %1 and %2 pushing %3

> This debug message indicates that the two strings are being popped off of the stack. They are then concatenated and the resulting string is pushed onto the stack. The strings are displayed in hex.

EVAL_DEBUG_EQUAL

Popping %1 and %2 pushing result %3

> This debug message indicates that the two strings are being popped off of the value stack and the result of comparing them is being pushed onto the value stack. The strings are displayed in hex.

EVAL_DEBUG_HEXSTRING

Pushing hex string %1

> This debug message indicates that the given binary string is being pushed onto the value stack. The string is displayed in hex.

EVAL_DEBUG_IFELSE_FALSE

Popping %1 (false) and %2, leaving %3

> This debug message indicates that the condition is false so the iftrue branch value is removed and the ifelse branch value is left on the value stack.

EVAL_DEBUG_IFELSE_TRUE

Popping %1 (true) and %2, leaving %3

> This debug message indicates that the condition is true so the ifelse branch value is removed and the iftrue branch value is left on the value stack.

EVAL_DEBUG_IPADDRESS

Pushing IPAddress %1

> This debug message indicates that the given binary string is being pushed onto the value stack. This represents either an IPv4 or IPv6 address. The string is displayed in hex.

EVAL_DEBUG_MEMBER

Checking membership of '%1', pushing result %2

> This debug message indicates that the membership of the packet for the client class was checked.

EVAL_DEBUG_NOT

Popping %1 pushing %2

> This debug message indicates that the first value is popped from the value stack, negated and then pushed onto the value stack. The string is displayed in text.

EVAL_DEBUG_OPTION

Pushing option %1 with value %2

> This debug message indicates that the given string representing the value of the requested option is being pushed onto the value stack. The string may be the text or binary value of the string based on the representation type requested (.text or .hex) or "true" or "false" if the requested type is .exists. The option code may be for either an option or a sub-option as requested in the classification statement.

EVAL_DEBUG_OR

Popping %1 and %2 pushing %3

> This debug message indicates that two values are popped from the value stack. Then are then combined via logical or and the result is pushed onto the value stack. The string is displayed in text.

EVAL_DEBUG_PKT

Pushing PKT meta data %1 with value %2

> This debug message indicates that the given binary string representing the value of the requested meta data is being pushed onto the value stack. The string is displayed in hex at the exception of interface name.

EVAL_DEBUG_PKT4

Pushing PKT4 field %1 with value %2

> This debug message indicates that the given binary string representing the value of the requested field is being pushed onto the value stack. The string is displayed in hex.

EVAL_DEBUG_PKT6

Pushing PKT6 field %1 with value %2

This debug message indicates that the given binary string representing the value of the requested field is being pushed onto the value stack. The string is displayed in hex.

EVAL_DEBUG_RELAY6

Pushing PKT6 relay field %1 nest %2 with value %3

This debug message indicates that the given binary string representing the value of the requested fied is being pushed onto the value stack. The string is displayed in hex.

EVAL_DEBUG_RELAY6_RANGE

Pushing PKT6 relay field %1 nest %2 with value %3

This debug message is generated if the nest field is out of range. The empty string will always be the value pushed onto the stack.

EVAL_DEBUG_STRING

Pushing text string %1

This debug message indicates that the given text string is being pushed onto the value stack. The string is displayed in text.

EVAL_DEBUG_SUBSTRING

Popping length %1, start %2, string %3 pushing result %4

This debug message indicates that three values are being popped from the value stack and a result is being pushed onto the value stack. The values being popped are the starting point and length of a substring to extract from the given string. The resulting string is pushed onto the stack. The strings are displayed in hex.

EVAL_DEBUG_SUBSTRING_EMPTY

Popping length %1, start %2, string %3 pushing result %4

This debug message indicates that the string popped from the stack was empty and so the result will also be empty. The start, length and string are still popped from the stack and the result is still pushed.

EVAL_DEBUG_SUBSTRING_RANGE

Popping length %1, start %2, string %3 pushing result %4

This debug message indicates that the value of start is outside of the string and an empty result will be pushed onto the stack. The start, length and string are still popped from the stack and the result is still pushed. The strings are displayed in hex.

EVAL_DEBUG_SUB_OPTION

Pushing option %1 sub-option %2 with value %3

This debug message indicates that the given string representing the value of the requested sub-option of the requested parent option is being pushed onto the value stack. The string may be the text or binary value of the string based on the representation type requested (.text or .hex) or “true” or “false” if the requested type is .exists. The codes are the parent option and the sub-option codes as requested in the classification statement.

EVAL_DEBUG_SUB_OPTION_NO_OPTION

Requested option %1 sub-option %2, but the parent option is not present, pushing result %3

This debug message indicates that the parent option was not found. The codes are the parent option and the sub-option codes as requested in the classification statement.

EVAL_DEBUG_TOHEXSTRING

Popping binary value %1 and separator %2, pushing result %3

> This debug message indicates that two values are being popped from the value stack and a result is being pushed onto the value stack. The values being popped are the binary value to convert and the separator. The binary value is converted to its hexadecimal string representation and pushed onto the stack. The binary value is displayed in hex.

EVAL_DEBUG_VENDOR_CLASS_DATA

Data %1 (out of %2 received) in vendor class found, pushing result '%3'

> This debug message indicates that vendor class option was found and passed enterprise-id checks and has sufficient number of data chunks. The total number of chunks and value pushed are reported as debugging aid.

EVAL_DEBUG_VENDOR_CLASS_DATA_NOT_FOUND

Requested data index %1, but option with enterprise-id %2 has only %3 data tuple(s), pushing result '%4'

> This debug message indicates that vendor class option was found and passed enterprise-id checks, but does not have sufficient number of data chunks. Note that the index starts at 0, so there has to be at least (index + 1) data chunks.

EVAL_DEBUG_VENDOR_CLASS_ENTERPRISE_ID

Pushing enterprise-id %1 as result 0x%2

> This debug message indicates that the expression has been evaluated and vendor class option was found and its enterprise-id is being reported.

EVAL_DEBUG_VENDOR_CLASS_ENTERPRISE_ID_MISMATCH

Was looking for %1, option had %2, pushing result '%3'

> This debug message indicates that the expression has been evaluated and vendor class option was found, but has different enterprise-id than specified in the expression.

EVAL_DEBUG_VENDOR_CLASS_EXISTS

Option with enterprise-id %1 found, pushing result '%2'

> This debug message indicates that the expression has been evaluated and vendor class option was found.

EVAL_DEBUG_VENDOR_CLASS_NO_OPTION

Option with code %1 missing, pushing result '%2'

> This debug message indicates that the expression has been evaluated and vendor class option was not found.

EVAL_DEBUG_VENDOR_ENTERPRISE_ID

Pushing enterprise-id %1 as result 0x%2

> This debug message indicates that the expression has been evaluated and vendor option was found and its enterprise-id is being reported.

EVAL_DEBUG_VENDOR_ENTERPRISE_ID_MISMATCH

Was looking for %1, option had %2, pushing result '%3'

> This debug message indicates that the expression has been evaluated and vendor option was found, but has different enterprise-id than specified in the expression.

EVAL_DEBUG_VENDOR_EXISTS

Option with enterprise-id %1 found, pushing result '%2'

> This debug message indicates that the expression has been evaluated and vendor option was found.

EVAL_DEBUG_VENDOR_NO_OPTION

Option with code %1 missing, pushing result '%2'

> This debug message indicates that the expression has been evaluated and vendor option was not found.

25.13 FLEX

FLEX_OPTION_LOAD_ERROR

loading Flex Option hooks library failed: %1

> This error message indicates an error during loading the Flex Option hooks library. The details of the error are provided as argument of the log message.

FLEX_OPTION_PROCESS_ADD

Added the option code %1 value by %2

> This debug message is printed when an option was added into the response packet. The option code and the value (between quotes if printable, in hexadecimal is not) are provided.

FLEX_OPTION_PROCESS_ERROR

An error occurred processing query %1: %2

> This error message indicates an error during processing of a query by the Flex Option hooks library. The client identification information from the query and the details of the error are provided as arguments of the log message.

FLEX_OPTION_PROCESS_REMOVE

Removed option code %1

> This debug message is printed when an option was removed from the response packet. The option code is provided.

FLEX_OPTION_PROCESS_SUPERSEDE

Supersedes the value of option code %1 by %2

> This debug message is printed when an option was superseded into the response packet. The option code and the value (between quotes if printable, in hexadecimal is not) are provided.

25.14 HA

HA_BUFFER4_RECEIVE_FAILED

buffer4_receive callout failed: %1

> This error message is issued when the callout for the buffer4_receive hook point failed. This may occur as a result of an internal server error. The argument contains a reason for the error.

HA_BUFFER4_RECEIVE_NOT_FOR_US

%1: dropping query to be processed by another server

This debug message is issued when the received DHCPv4 query is dropped by this server because it should be served by another server. This is the case when the remote server was designated to process the packet as a result of load balancing or because it is a primary server in the hot standby configuration. The argument provides client identification information retrieved from the query.

HA_BUFFER4_RECEIVE_PACKET_OPTIONS_SKIPPED

an error upacking an option, caused subsequent options to be skipped: %1

A debug message issued when an option failed to unpack correctly, making it impossible to unpack the remaining options in the DHCPv4 query. The server will still attempt to service the packet. The sole argument provides a reason for unpacking error.

HA_BUFFER4_RECEIVE_UNPACK_FAILED

failed to parse query from %1 to %2, received over interface %3, reason: %4

This debug message is issued when received DHCPv4 query is malformed and can't be parsed by the buffer4_receive callout. The query will be dropped by the server. The first three arguments specify source IP address, destination IP address and the interface. The last argument provides a reason for failure.

HA_BUFFER6_RECEIVE_FAILED

buffer6_receive callout failed: %1

This error message is issued when the callout for the buffer6_receive hook point failed. This may occur as a result of an internal server error. The argument contains a reason for the error.

HA_BUFFER6_RECEIVE_NOT_FOR_US

%1: dropping query to be processed by another server

This debug message is issued when the received DHCPv6 query is dropped by this server because it should be served by another server. This is the case when the remote server was designated to process the packet as a result of load balancing or because it is a primary server in the hot standby configuration. The argument provides client identification information retrieved from the query.

HA_BUFFER6_RECEIVE_PACKET_OPTIONS_SKIPPED

an error upacking an option, caused subsequent options to be skipped: %1

A debug message issued when an option failed to unpack correctly, making it impossible to unpack the remaining options in the DHCPv6 query. The server will still attempt to service the packet. The sole argument provides a reason for unpacking error.

HA_BUFFER6_RECEIVE_UNPACK_FAILED

failed to parse query from %1 to %2, received over interface %3, reason: %4

This debug message is issued when received DHCPv6 query is malformed and can't be parsed by the buffer6_receive callout. The query will be dropped by the server. The first three arguments specify source IP address, destination IP address and the interface. The last argument provides a reason for failure.

HA_COMMAND_PROCESSED_FAILED

command_processed callout failed: %1

This error message is issued when the callout for the command_processed hook point failed. The argument contains a reason for the error.

HA_COMMUNICATION_INTERRUPTED

communication with %1 is interrupted

This warning message is issued by the server which discovered that the communication to the active partner has been interrupted for a time period longer than the configured heartbeat-delay time. At this stage the server starts the failover procedure by monitoring the DHCP traffic sent to the partner and checking whether the partner server responds to this traffic. If the max-unacked-clients value is set to 0 such verification is disabled in which case the server will transition to the partner-down state.

HA_COMMUNICATION_INTERRUPTED_CLIENT4

%1: new client attempting to get a lease from the partner

This informational message is issued when the surviving server observes a DHCP packet sent to the partner with which the communication is interrupted. The client whose packet is observed is not yet considered "unacked" because the secs field value does not exceed the configured threshold specified with max-ack-delay.

HA_COMMUNICATION_INTERRUPTED_CLIENT4_UNACKED

%1: partner server failed to respond, %2 clients unacked so far, %3 clients left before transitioning to the partner-down state

This informational message is issued when the surviving server determines that its partner failed to respond to the DHCP query and that this client is considered to not be served by the partner. The surviving server counts such clients and if the number of such clients exceeds the max-unacked-clients threshold, the server will transition to the partner-down state. The first argument contains client identification information. The second argument specifies the number of clients to which the server has failed to respond. The third argument specifies the number of additional clients which, if not provisioned, will cause the server to transition to the partner-down state.

HA_COMMUNICATION_INTERRUPTED_CLIENT6

%1: new client attempting to get a lease from the partner

This informational message is issued when the surviving server observes a DHCP packet sent to the partner with which the communication is interrupted. The client whose packet is observed is not yet considered "unacked" because the elapsed time option value does not exceed the configured threshold specified with max-ack-delay. The sole argument specifies client identification information.

HA_COMMUNICATION_INTERRUPTED_CLIENT6_UNACKED

%1: partner server failed to respond, %2 clients unacked so far, %3 clients left before transitioning to the partner-down state

This informational message is issued when the surviving server determines that its partner failed to respond to the DHCP query and that this client is considered to not be served by the partner. The surviving server counts such clients and if the number of such clients exceeds the max-unacked-clients threshold, the server will transition to the partner-down state. The first argument contains client identification information. The second argument specifies the number of clients to which the server has failed to respond. The third argument specifies the number of additional clients which, if not provisioned, will cause the server to transition to the partner-down state.

HA_CONFIGURATION_FAILED

failed to configure High Availability hooks library: %1

This error message is issued when there is an error configuring the HA hooks library. The argument provides the detailed error message.

HA_CONFIGURATION_SUCCESSFUL

HA hook library has been successfully configured

This informational message is issued when the HA hook library configuration parser successfully parses and validates the new configuration.

HA_CONFIG_AUTO_FAILOVER_DISABLED

auto-failover disabled for %1

> This warning message is issued to indicate that the 'auto-failover' parameter was administratively disabled for the specified server. The server will not automatically start serving partner's scope when the partner failure is detected. The server administrator will need to enable this scope manually by sending appropriate ha-scopes command.

HA_CONFIG_LEASE_SYNCING_DISABLED

lease database synchronization between HA servers is disabled

> This warning message is issued when the lease database synchronization is administratively disabled. This is valid configuration if the leases are replicated between lease databases via some other mechanism, e.g. SQL database replication.

HA_CONFIG_LEASE_SYNCING_DISABLED_REMINDER

bypassing SYNCING state because lease database synchronization is administratively disabled

> This informational message is issued as a reminder that lease database synchronization is administratively disabled and therefore the server transitions directly from the "waiting" to "ready" state.

HA_CONFIG_LEASE_UPDATES_AND_SYNCING_DIFFER

unusual configuration where "send-lease-updates": %1 and "sync-leases": %2

> This warning message is issued when the configuration values of the send-lease-updates and sync-leases parameters differ. This may be a valid configuration but is unusual. Normally, if the lease database with replication is in use, both values are set to false. If a lease database without replication is in use (e.g. memfile), both values are set to true. Providing different values for those parameters means that an administrator either wants the server to not synchronize leases upon startup but later send lease updates to the partner, or the lease database should be synchronized upon startup, but no lease updates are later sent as a result of leases allocation.

HA_CONFIG_LEASE_UPDATES_DISABLED

lease updates will not be generated

> This warning message is issued when the lease updates are administratively disabled. This is valid configuration if the leases are replicated to the partner's database via some other mechanism, e.g. SQL database replication.

HA_CONFIG_LEASE_UPDATES_DISABLED_REMINDER

lease updates are administratively disabled and will not be generated while in %1 state

> This informational message is issued as a reminder that the lease updates are administratively disabled and will not be issued in the HA state to which the server has transitioned. The sole argument specifies the state into which the server has transitioned.

HA_CONTINUE_HANDLER_FAILED

ha-continue command failed: %1

> This error message is issued to indicate that the ha-continue command handler failed while processing the command. The argument provides the reason for failure.

HA_DEINIT_OK

unloading High Availability hooks library successful

> This informational message indicates that the High Availability hooks library has been unloaded successfully.

HA_DHCP4_START_SERVICE_FAILED

failed to start DHCPv4 HA service in dhcp4_srv_configured callout: %1

> This error message is issued when an attempt to start High Availability service for the DHCPv4 server failed in the dhcp4_srv_configured callout. This is internal server error and a bug report should be created.

HA_DHCP6_START_SERVICE_FAILED

failed to start DHCPv4 HA service in dhcp6_srv_configured callout: %1

> This error message is issued when an attempt to start High Availability service for the DHCPv4 server failed in the dhcp4_srv_configured callout. This is internal server error and a bug report should be created.

HA_DHCP_DISABLE_COMMUNICATIONS_FAILED

failed to send request to disable DHCP service of %1: %2

> This warning message indicates that there was a problem in communication with a HA peer while sending the dhcp-disable command. The first argument provides the remote server's name. The second argument provides a reason for failure.

HA_DHCP_DISABLE_FAILED

failed to disable DHCP service of %1: %2

> This warning message indicates that a peer returned an error status code in response to a dhcp-disable command. The first argument provides the remote server's name. The second argument provides a reason for failure.

HA_DHCP_ENABLE_COMMUNICATIONS_FAILED

failed to send request to enable DHCP service of %1: %2

> This warning message indicates that there was a problem in communication with a HA peer while sending the dhcp-enable command. The first argument provides the remote server's name. The second argument provides a reason for failure.

HA_DHCP_ENABLE_FAILED

failed to enable DHCP service of %1: %2

> This warning message indicates that a peer returned an error status code in response to a dhcp-enable command. The first argument provides the remote server's name. The second argument provides a reason for failure.

HA_HEARTBEAT_COMMUNICATIONS_FAILED

failed to send heartbeat to %1: %2

> This warning message indicates that there was a problem in communication with a HA peer while sending a heartbeat. This is a first sign that the peer may be down. The server will keep trying to send heartbeats until it considers that communication is interrupted.

HA_HEARTBEAT_FAILED

heartbeat to %1 failed: %2

> This warning message indicates that a peer returned an error status code in response to a heartbeat. This is the sign that the peer may not function properly. The server will keep trying to send heartbeats until it considers that communication is interrupted.

HA_HEARTBEAT_HANDLER_FAILED

heartbeat command failed: %1

This error message is issued to indicate that the heartbeat command handler failed while processing the command. The argument provides the reason for failure.

HA_HIGH_CLOCK_SKEW

%1, please synchronize clocks!

This warning message is issued when the clock skew between the active servers exceeds 30 seconds. The HA service continues to operate but may not function properly, especially for low lease lifetimes. The administrator should should synchronize the clocks, e.g. using NTP. If the clock skew exceeds 60 seconds, the HA service will terminate.

HA_HIGH_CLOCK_SKEW_CAUSES_TERMINATION

%1, causing HA service to terminate

This warning message is issued when the clock skew between the active servers exceeds 60 seconds. The HA service stops. The servers will continue to respond to the DHCP queries but won't exchange lease updates or send heartbeats. The administrator is required to synchronize the clocks and then restart the servers to resume the HA service.

HA_INIT_OK

loading High Availability hooks library successful

This informational message indicates that the High Availability hooks library has been loaded successfully.

HA_LEASES4_COMMITTED_FAILED

leases4_committed callout failed: %1

This error message is issued when the callout for the leases4_committed hook point failed. This includes unexpected errors like wrong arguments provided to the callout by the DHCP server (unlikely internal server error). The argument contains a reason for the error.

HA_LEASES4_COMMITTED_NOTHING_TO_UPDATE

%1: leases4_committed callout was invoked without any leases

This debug message is issued when the "leases4_committed" callout returns because there are neither new leases nor deleted leases for which updates should be sent. The sole argument specifies the details of the client which sent the packet.

HA_LEASES6_COMMITTED_FAILED

leases6_committed callout failed: %1

This error message is issued when the callout for the leases6_committed hook point failed. This includes unexpected errors like wrong arguments provided to the callout by the DHCP server (unlikely internal server error). The argument contains a reason for the error.

HA_LEASES6_COMMITTED_NOTHING_TO_UPDATE

%1: leases6_committed callout was invoked without any leases

This debug message is issued when the "leases6_committed" callout returns because there are neither new leases nor deleted leases for which updates should be sent. The sole argument specifies the details of the client which sent the packet.

HA_LEASES_SYNC_COMMUNICATIONS_FAILED

failed to communicate with %1 while syncing leases: %2

This error message is issued to indicate that there was a communication error with a partner server while trying to fetch leases from its lease database. The argument contains a reason for the error.

HA_LEASES_SYNC_FAILED

failed to synchronize leases with %1: %2

> This error message is issued to indicate that there was a problem while parsing a response from the server from which leases have been fetched for local database synchronization. The argument contains a reason for the error.

HA_LEASES_SYNC_LEASE_PAGE_RECEIVED

received %1 leases from %2

> This informational message is issued during lease database synchronization to indicate that a bulk of leases have been received. The first argument holds the count of leases received. The second argument specifies the partner server name.

HA_LEASE_SYNC_FAILED

synchronization failed for lease: %1, reason: %2

> This warning message is issued when creating or updating a lease in the local lease database fails. The lease information in the JSON format is provided as a first argument. The second argument provides a reason for the failure.

HA_LEASE_SYNC_STALE_LEASE4_SKIP

skipping stale lease %1 in subnet %2

> This debug message is issued during lease database synchronization, when fetched IPv4 lease instance appears to be older than the instance in the local database. The newer instance is left in the database and the fetched lease is dropped. The remote server will still hold the older lease instance until it synchronizes its database with this server. The first argument specifies leased address. The second argument specifies a subnet to which the lease belongs.

HA_LEASE_SYNC_STALE_LEASE6_SKIP

skipping stale lease %1 in subnet %2

> This debug message is issued during lease database synchronization, when fetched IPv6 lease instance appears to be older than the instance in the local database. The newer instance is left in the database and the fetched lease is dropped. The remote server will still hold the older lease instance until it synchronizes its database with this server. The first argument specifies leased address. The second argument specifies a subnet to which the lease belongs.

HA_LEASE_UPDATES_DISABLED

lease updates will not be sent to the partner while in %1 state

> This informational message is issued to indicate that lease updates will not be sent to the partner while the server is in the current state. The argument specifies the server's current state name. The lease updates are still sent to the backup servers if they are configured but any possible errors in communication with the backup servers are ignored.

HA_LEASE_UPDATES_ENABLED

lease updates will be sent to the partner while in %1 state

> This informational message is issued to indicate that lease updates will be sent to the partner while the server is in the current state. The argument specifies the server's current state name.

HA_LEASE_UPDATE_COMMUNICATIONS_FAILED

%1: failed to communicate with %2: %3

This warning message indicates that there was a problem in communication with a HA peer while processing a DHCP client query and sending lease update. The client's DHCP message will be dropped.

HA_LEASE_UPDATE_CREATE_UPDATE_FAILED_ON_PEER

%1: failed to create or update the lease having type %2 for address %3, reason: %4

This informational message is issued when one of the leases failed to be created or updated on the HA peer whilw processing the lease updates sent from this server. This may indicate an issue with communication between the peer and its lease database.

HA_LEASE_UPDATE_DELETE_FAILED_ON_PEER

%1: failed to delete the lease having type %2 for address %3, reason: %4

This informational message is issued when one of the leases failed to delete on the HA peer while processing lease updates sent from this server. Typically, the lease fails to delete when it doesn't exist in the peer's database.

HA_LEASE_UPDATE_FAILED

%1: lease update to %2 failed: %3

This warning message indicates that a peer returned an error status code in response to a lease update. The client's DHCP message will be dropped.

HA_LOAD_BALANCING_DUID_MISSING

load balancing failed for the DHCPv6 message (transaction id: %1) because DUID is missing

This debug message is issued when the HA hook library was unable to load balance an incoming DHCPv6 query because neither client identifier nor HW address was included in the query. The query will be dropped. The sole argument contains transaction id.

HA_LOAD_BALANCING_IDENTIFIER_MISSING

load balancing failed for the DHCPv4 message (transaction id: %1) because HW address and client identifier are missing

This debug message is issued when the HA hook library was unable to load balance an incoming DHCPv4 query because neither client identifier nor HW address was included in the query. The query will be dropped. The sole argument contains transaction id.

HA_LOCAL_DHCP_DISABLE

local DHCP service is disabled while the %1 is in the %2 state

This informational message is issued to indicate that the local DHCP service is disabled because the server remains in a state in which the server should not respond to DHCP clients, e.g. the server hasn't synchronized its lease database. The first argument specifies server name. The second argument specifies server's state.

HA_LOCAL_DHCP_ENABLE

local DHCP service is enabled while the %1 is in the %2 state

This informational message is issued to indicate that the local DHCP service is enabled because the server remains in a state in which it should respond to the DHCP clients. The first argument specifies server name. The second argument specifies server's state.

HA_MAINTENANCE_CANCEL_HANDLER_FAILED

ha-maintenance-cancel command failed: %1

This error message is issued to indicate that the ha-maintenance-cancel command handler failed while processing the command. The argument provides the reason for failure.

HA_MAINTENANCE_NOTIFY_CANCEL_COMMUNICATIONS_FAILED

failed to send ha-maintenance-notify to %1 in attempt to cancel its maintenance: %2

> This warning message indicates that there was a problem in communication with a HA peer while sending the ha-maintenance-notify command with the cancel flag set to true. The first argument provides the remote server's name. The second argument provides a reason for failure.

HA_MAINTENANCE_NOTIFY_CANCEL_FAILED

error returned while processing ha-maintenance-notify by %1 in attempt to cancel its maintenance: %2

> This warning message indicates that a peer returned an error status code in response to a ha-maintenance-notify command with the cancel flag set to true. The first argument provides the remote server's name. The second argument provides a reason for failure.

HA_MAINTENANCE_NOTIFY_COMMUNICATIONS_FAILED

failed to send ha-maintenance-notify to %1: %2

> This warning message indicates that there was a problem in communication with a HA peer while sending the ha-maintenance-notify command. The first argument provides the remote server's name. The second argument provides a reason for failure.

HA_MAINTENANCE_NOTIFY_FAILED

error returned while processing ha-maintenance-notify by %1: %2

> This warning message indicates that a peer returned an error status code in response to a ha-maintenance-notify command. The first argument provides the remote server's name. The second argument provides a reason for failure.

HA_MAINTENANCE_NOTIFY_HANDLER_FAILED

ha-maintenance-notify command failed: %1

> This error message is issued to indicate that the ha-maintenance-notify command handler failed while processing the command. The argument provides the reason for failure.

HA_MAINTENANCE_SHUTDOWN_SAFE

the server can now be shutdown for maintenance as the partner has taken over the DHCP traffic

> This informational message is displayed after the server transitions to the in-maintenance state. This server no longer responds to any DHCP queries and its partner being in the partner-in-maintenance state has taken over the DHCP traffic. When the server in-maintenance state is shut down, the partner will move to the partner-down imediatelly.

HA_MAINTENANCE_STARTED

the server is now in the partner-in-maintenance state and the partner is in-maintenance state

> This informational message is displayed when the server receiving the ha-maintenance-start command transitions to the partner-in-maintenance state. The server does it after sending the ha-maintenance-notify to its partner to put the partner in the in-maintenance state. From now on, the server in the partner-in-maintenance state will be responding to all queries and the partner will respond to no queries. The partner may be safely shut down for maintenance in which case this server will automatically transition from the partner-in-maintenance state to the partner-down state.

HA_MAINTENANCE_STARTED_IN_PARTNER_DOWN

the server is now in the partner-down mode as a result of requested maintenance

This informational message is displayed when the server receiving the ha-maintenance-start command transitions to the partner-down state because it was unable to communicate with the partner while receiving the command. It is assumed that in such situation the partner is already offline for the maintenance. Note that in this case the normal failover procedure does not take place. The server does not wait for a heartbeat to fail several times, nor it monitors the DHCP traffic for not responded queries. In the maintenance case the server transitions to the partner-down state when it first encounters a communication problem with the partner.

HA_MAINTENANCE_START_HANDLER_FAILED

ha-maintenance-start command failed: %1

This error message is issued to indicate that the ha-maintenance-start command handler failed while processing the command. The argument provides the reason for failure.

HA_MISSING_CONFIGURATION

high-availability parameter not specified for High Availability hooks library

This error message is issued to indicate that the configuration for the High Availability hooks library hasn't been specified. The 'high-availability' parameter must be specified for the hooks library to load properly.

HA_SCOPES_HANDLER_FAILED

ha-scopes command failed: %1

This error message is issued to indicate that the ha-scopes command handler failed while processing the command. The argument provides reason for the failure.

HA_SERVICE_STARTED

started high availability service in %1 mode as %2 server

This informational message is issued when the HA service is started as a result of server startup or reconfiguration. The first argument provides the HA mode. The second argument specifies the role of this server instance in this configuration.

HA_STATE_MACHINE_CONTINUED

state machine is un-paused

This informational message is issued when the HA state machine is un-paused. This unlocks the server from the current state. It may transition to any other state if it needs to do so, e.g. 'partner-down' if its partner appears to be offline. The server may also remain in the current state if the HA setup state warrants such behavior.

HA_STATE_MACHINE_PAUSED

state machine paused in state %1

This informational message is issued when the HA state machine is paused. HA state machine may be paused in certain states specified in the HA hooks library confuguration. When the state machine is paused, the server remains in the given state until it is explicitly unpaused (via ha-continue command). If the state machine is paused, the server operates normally but can't transition to any other state.

HA_STATE_TRANSITION

server transitions from %1 to %2 state, partner state is %3

This informational message is issued when the server transitions to a new state as a result of some interaction (or lack of thereof) with its partner. The arguments specify initial server state, new server state and the partner's state.

HA_STATE_TRANSITION_PASSIVE_BACKUP

server transitions from %1 to %2 state

> This informational message is issued when the server being in the passive-backup mode transitions to a new state. The arguments specify initial server state and a new server state.

HA_SYNC_FAILED

lease database synchronization with %1 failed: %2

> This error message is issued to indicate that the lease database synchronization failed. The first argument provides partner server's name. The second argument provides a reason for the failure.

HA_SYNC_HANDLER_FAILED

ha-sync command failed: %1

> This error message is issued to indicate that the ha-sync command handler failed while processing the command. The argument provides the reason for failure.

HA_SYNC_START

starting lease database synchronization with %1

> This informational message is issued when the server starts lease database synchronization with a partner. The name of the partner is specified with the sole argument.

HA_SYNC_SUCCESSFUL

lease database synchronization with %1 completed successfully in %2

> This informational message is issued when the server successfully completed lease database synchronization with the partner. The first argument specifies the name of the partner server. The second argument specifies the duration of the synchronization.

HA_TERMINATED

HA service terminated due to an unrecoverable condition. Check previous error message(s), address the problem and restart!

> This error message is issued to indicate that the HA service has been stopped due to an unacceptable condition (e.g. too large of a clock skew). The exact cause should appear in a previous error message. Address the condition reported then restart the servers to resume service.

25.15 HOOKS

HOOKS_ALL_CALLOUTS_DEREGISTERED

hook library at index %1 removed all callouts on hook %2

> A debug message issued when all callouts on the specified hook registered by the library with the given index were removed. This is similar to the HOOKS_CALLOUTS_REMOVED message (and the two are likely to be seen together), but is issued at a lower-level in the hook framework.

HOOKS_CALLOUTS_BEGIN

begin all callouts for hook %1

> This debug message is issued when callout manager begins to invoke callouts for the hook. The argument specifies the hook name.

HOOKS_CALLOUTS_COMPLETE

completed callouts for hook %1 (total callouts duration: %2)

> This debug message is issued when callout manager has completed execution of all callouts for the particular hook. The arguments specify the hook name and total execution time for all callouts in milliseconds.

HOOKS_CALLOUTS_REMOVED

callouts removed from hook %1 for library %2

> This is a debug message issued during library unloading. It notes that one of more callouts registered by that library have been removed from the specified hook. This is similar to the HOOKS_DEREGISTER_ALL_CALLOUTS message (and the two are likely to be seen together), but is issued at a higher-level in the hook framework.

HOOKS_CALLOUT_CALLED

hooks library with index %1 has called a callout on hook %2 that has address %3 (callout duration: %4)

> Only output at a high debugging level, this message indicates that a callout on the named hook registered by the library with the given index (in the list of loaded libraries) has been called and returned a success state. The address of the callout is given in the message. The message includes the callout execution time in milliseconds.

HOOKS_CALLOUT_DEREGISTERED

hook library at index %1 deregistered a callout on hook %2

> A debug message issued when all instances of a particular callouts on the hook identified in the message that were registered by the library with the given index have been removed.

HOOKS_CALLOUT_ERROR

error returned by callout on hook %1 registered by library with index %2 (callout address %3) (callout duration %4)

> If a callout returns an error status when called, this error message is issued. It identifies the hook to which the callout is attached, the index of the library (in the list of loaded libraries) that registered it and the address of the callout. The error is otherwise ignored. The error message includes the callout execution time in milliseconds.

HOOKS_CALLOUT_EXCEPTION

exception thrown by callout on hook %1 registered by library with index %2 (callout address %3): %4 (callout duration: %5)

> If a callout throws an exception when called, this error message is issued. It identifies the hook to which the callout is attached, the index of the library (in the list of loaded libraries) that registered it and the address of the callout. The error is otherwise ignored. The error message includes the callout execution time in milliseconds.

HOOKS_CALLOUT_REGISTRATION

hooks library with index %1 registering callout for hook '%2'

> This is a debug message, output when a library (whose index in the list of libraries (being) loaded is given) registers a callout.

HOOKS_CLOSE_ERROR

failed to close hook library %1: %2

> Kea has failed to close the named hook library for the stated reason. Although this is an error, this should not affect the running system other than as a loss of resources. If this error persists, you should restart Kea.

HOOKS_HOOK_LIST_RESET

the list of hooks has been reset

> This is a message indicating that the list of hooks has been reset. While this is usual when running the Kea test suite, it should not be seen when running Kea in a production environment. If this appears, please report a bug through the usual channels.

HOOKS_INCORRECT_VERSION

hook library %1 is at version %2, require version %3

> Kea has detected that the named hook library has been built against a version of Kea that is incompatible with the version of Kea running on your system. It has not loaded the library. This is most likely due to the installation of a new version of Kea without rebuilding the hook library. A rebuild and re-install of the library should fix the problem in most cases.

HOOKS_LIBRARY_CLOSED

hooks library %1 successfully closed

> This information message is issued when a user-supplied hooks library has been successfully closed.

HOOKS_LIBRARY_LOADED

hooks library %1 successfully loaded

> This information message is issued when a user-supplied hooks library has been successfully loaded.

HOOKS_LIBRARY_LOADING

loading hooks library %1

> This is a debug message output just before the specified library is loaded. If the action is successfully, it will be followed by the HOOKS_LIBRARY_LOADED informational message.

HOOKS_LIBRARY_MULTI_THREADING_COMPATIBLE

hooks library %1 reports its multi-threading compatibility as %2

> A debug message issued when the "multi_threading_compatible" function was called. The returned value (0 means not compatible, others compatible) is displayed.

HOOKS_LIBRARY_MULTI_THREADING_NOT_COMPATIBLE

hooks library %1 is not compatible with multi-threading

> When multi-threading is enabled and the library is not compatible (either because the "multi_threading_compatible" function returned 0 or was not implemented) this error message is issued. The library must be removed from the configuration or the multi-threading disabled.

HOOKS_LIBRARY_UNLOADED

hooks library %1 successfully unloaded

> This information message is issued when a user-supplied hooks library has been successfully unloaded.

HOOKS_LIBRARY_UNLOADING

unloading library %1

> This is a debug message called when the specified library is being unloaded. If all is successful, it will be followed by the HOOKS_LIBRARY_UNLOADED informational message.

HOOKS_LIBRARY_VERSION

hooks library %1 reports its version as %2

A debug message issued when the version check on the hooks library has succeeded.

HOOKS_LOAD_ERROR

'load' function in hook library %1 returned error %2

A "load" function was found in the library named in the message and was called. The function returned a non-zero status (also given in the message) which was interpreted as an error. The library has been unloaded and no callouts from it will be installed.

HOOKS_LOAD_EXCEPTION

'load' function in hook library %1 threw an exception

A "load" function was found in the library named in the message and was called. The function threw an exception (an error indication) during execution, which is an error condition. The library has been unloaded and no callouts from it will be installed.

HOOKS_LOAD_FRAMEWORK_EXCEPTION

'load' function in hook library %1 threw an exception: reason %2

A "load" function was found in the library named in the message and was called. Either the hooks framework or the function threw an exception (an error indication) during execution, which is an error condition; the cause of the exception is recorded in the message. The library has been unloaded and no callouts from it will be installed.

HOOKS_LOAD_SUCCESS

'load' function in hook library %1 returned success

This is a debug message issued when the "load" function has been found in a hook library and has been successfully called.

HOOKS_MULTI_THREADING_COMPATIBLE_EXCEPTION

'multi_threading_compatible' function in hook library %1 threw an exception

This error message is issued if the multi_threading_compatible() function in the specified hooks library was called and generated an exception. The library is considered unusable and will not be loaded.

HOOKS_NO_LOAD

no 'load' function found in hook library %1

This is a debug message saying that the specified library was loaded but no function called "load" was found in it. Providing the library contained some "standard" functions (i.e. functions with the names of the hooks for the given server), this is not an issue.

HOOKS_NO_UNLOAD

no 'unload' function found in hook library %1

This is a debug message issued when the library is being unloaded. It merely states that the library did not contain an "unload" function.

HOOKS_NO_VERSION

no 'version' function found in hook library %1

The shared library named in the message was found and successfully loaded, but Kea did not find a function named "version" in it. This function is required and should return the version of Kea against which the library was built. The value is used to check that the library was built against a compatible version of Kea. The library has not been loaded.

HOOKS_OPEN_ERROR

failed to open hook library %1: %2

> Kea failed to open the specified hook library for the stated reason. The library has not been loaded. Kea will continue to function, but without the services offered by the library.

HOOKS_STD_CALLOUT_REGISTERED

hooks library %1 registered standard callout for hook %2 at address %3

> This is a debug message, output when the library loading function has located a standard callout (a callout with the same name as a hook point) and registered it. The address of the callout is indicated.

HOOKS_UNLOAD_ERROR

'unload' function in hook library %1 returned error %2

> During the unloading of a library, an "unload" function was found. It was called, but returned an error (non-zero) status, resulting in the issuing of this message. The unload process continued after this message and the library has been unloaded.

HOOKS_UNLOAD_EXCEPTION

'unload' function in hook library %1 threw an exception

> During the unloading of a library, an "unload" function was found. It was called, but in the process generated an exception (an error indication). The unload process continued after this message and the library has been unloaded.

HOOKS_UNLOAD_FRAMEWORK_EXCEPTION

'unload' function in hook library %1 threw an exception, reason %2

> During the unloading of a library, an "unload" function was found. It was called, but in the process either it or the hooks framework generated an exception (an error indication); the cause of the error is recorded in the message. The unload process continued after this message and the library has been unloaded.

HOOKS_UNLOAD_SUCCESS

'unload' function in hook library %1 returned success

> This is a debug message issued when an "unload" function has been found in a hook library during the unload process, called, and returned success.

25.16 HOSTS

HOSTS_BACKENDS_REGISTERED

the following host backend types are available: %1

> This informational message lists all possible host backends that could be used in hosts-database[s].

HOSTS_BACKEND_DEREGISTER

deregistered host backend type: %1

> This debug message is issued when a backend factory was deregistered. It is no longer possible to use host backend of this type.

HOSTS_BACKEND_REGISTER

registered host backend type: %1

> This debug message is issued when a backend factory was successfully registered. It is now possible to use host backend of this type.

HOSTS_CFG_ADD_HOST

add the host for reservations: %1

> This debug message is issued when new host (with reservations) is added to the server's configuration. The argument describes the host and its reservations in detail.

HOSTS_CFG_CACHE_HOST_DATA_SOURCE

get host cache data source: %1

> This informational message is issued when a host cache data source is detected by the host manager.

HOSTS_CFG_CLOSE_HOST_DATA_SOURCE

Closing host data source: %1

> This is a normal message being printed when the server closes host data source connection.

HOSTS_CFG_DEL_ALL_SUBNET4

deleted all %1 host(s) for subnet id %2

> This debug message is issued when all IPv4 reservations are deleted for the specified subnet. The first argument specifies how many reservations have been deleted. The second argument is the subnet identifier.

HOSTS_CFG_DEL_ALL_SUBNET6

deleted all %1 host(s) including %2 IPv6 reservation(s) for subnet id %3

> This debug message is issued when all IPv6 reservations are deleted for the specified subnet. The first argument specifies how many hosts have been deleted. The second argument specifies how many IPv6 (addresses and prefixes) have been deleted. The third argument is the subnet identifier.

HOSTS_CFG_GET_ALL

get all hosts with reservations

> This debug message is issued when starting to retrieve all hosts.

HOSTS_CFG_GET_ALL_ADDRESS4

get all hosts with reservations for IPv4 address %1

> This debug message is issued when starting to retrieve all hosts, holding the reservation for the specific IPv4 address, from the configuration. The argument specifies the IPv4 address used to search the hosts.

HOSTS_CFG_GET_ALL_ADDRESS4_COUNT

using address %1, found %2 host(s)

> This debug message logs the number of hosts found using the specified IPv4 address. The arguments specify the IPv4 address used and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_ADDRESS4_HOST

using address %1 found host: %2

> This debug message is issued when found host with the reservation for the specified IPv4 address. The arguments specify the IPv4 address and the detailed description of the host found.

HOSTS_CFG_GET_ALL_ADDRESS6

get all hosts with reservations for IPv6 address %1

This debug message is issued when starting to retrieve all hosts, holding the reservation for the specific IPv6 address, from the configuration. The argument specifies the IPv6 address used to search the hosts.

HOSTS_CFG_GET_ALL_ADDRESS6_COUNT

using address %1, found %2 host(s)

This debug message logs the number of hosts found using the specified IPv6 address. The arguments specify the IPv6 address used and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_ADDRESS6_HOST

using address %1 found host: %2

This debug message is issued when found host with the reservation for the specified IPv6 address. The arguments specify the IPv6 address and the detailed description of the host found.

HOSTS_CFG_GET_ALL_COUNT

found %1 host(s)

This debug message include the details of the host found. The argument specifies the number of hosts found.

HOSTS_CFG_GET_ALL_HOST

found host: %1

This debug message includes the details of the host found. The argument specifies found host details.

HOSTS_CFG_GET_ALL_HOSTNAME

get all hosts with reservations for hostname %1

This debug message is issued when starting to retrieve all hosts with the specific hostname. The argument specifies hostname.

HOSTS_CFG_GET_ALL_HOSTNAME_COUNT

using hostname %1, found %2 host(s)

This debug message include the details of the host found using the hostname. The arguments specify hostname and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_HOSTNAME_HOST

using hostname %1, found host: %2

This debug message includes the details of the host found using the hostname. The arguments specify hostname and found host details respectively.

HOSTS_CFG_GET_ALL_HOSTNAME_SUBNET_ID4

get all hosts with reservations for hostname %1 and IPv4 subnet %2

This debug message is issued when starting to retrieve all hosts with the specific hostname connected to the specific DHCPv4 subnet. The argument specifies hostname and subnet id.

HOSTS_CFG_GET_ALL_HOSTNAME_SUBNET_ID4_COUNT

using hostname %1 and IPv4 subnet %2, found %3 host(s)

This debug message include the details of the host found using the hostname and the DHCPv4 subnet id. The arguments specify hostname, subnet id and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_HOSTNAME_SUBNET_ID4_HOST

using hostname %1 and IPv4 subnet %2, found host: %3

This debug message includes the details of the host found using the hostname and the DHCPv4 subnet id. The arguments specify hostname, subnet id and found host details respectively.

HOSTS_CFG_GET_ALL_HOSTNAME_SUBNET_ID6

get all hosts with reservations for hostname %1 and IPv6 subnet %2

This debug message is issued when starting to retrieve all hosts with the specific hostname connected to the specific DHCPv6 subnet. The argument specifies hostname and subnet id.

HOSTS_CFG_GET_ALL_HOSTNAME_SUBNET_ID6_COUNT

using hostname %1 and IPv6 subnet %2, found %3 host(s)

This debug message include the details of the host found using the hostname and the DHCPv6 subnet id. The arguments specify hostname, subnet id and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_HOSTNAME_SUBNET_ID6_HOST

using hostname %1 and IPv6 subnet %2, found host: %3

This debug message includes the details of the host found using the hostname and the DHCPv6 subnet id. The arguments specify hostname, subnet id and found host details respectively.

HOSTS_CFG_GET_ALL_IDENTIFIER

get all hosts with reservations using identifier: %1

This debug message is issued when starting to retrieve reservations for all hosts identified by HW address or DUID. The argument holds both the identifier type and the value.

HOSTS_CFG_GET_ALL_IDENTIFIER_COUNT

using identifier %1, found %2 host(s)

This debug message logs the number of hosts found using the specified identifier. The arguments specify the identifier used and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_IDENTIFIER_HOST

using identifier: %1, found host: %2

This debug message is issued when found host identified by the specific identifier. The arguments specify the identifier and the detailed description of the host found.

HOSTS_CFG_GET_ALL_SUBNET_ID4

get all hosts with reservations for IPv4 subnet %1

This debug message is issued when starting to retrieve all hosts connected to the specific DHCPv4 subnet. The argument specifies subnet id.

HOSTS_CFG_GET_ALL_SUBNET_ID4_COUNT

using IPv4 subnet %1, found %2 host(s)

This debug message include the details of the host found using the DHCPv4 subnet id. The arguments specify subnet id and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_SUBNET_ID4_HOST

using IPv4 subnet %1, found host: %2

This debug message includes the details of the host found using the DHCPv4 subnet id. The arguments specify subnet id and found host details respectively.

HOSTS_CFG_GET_ALL_SUBNET_ID6

get all hosts with reservations for IPv6 subnet %1

> This debug message is issued when starting to retrieve all hosts connected to the specific DHCPv6 subnet. The argument specifies subnet id.

HOSTS_CFG_GET_ALL_SUBNET_ID6_COUNT

using IPv6 subnet %1, found %2 host(s)

> This debug message include the details of the host found using the DHCPv6 subnet id. The arguments specify subnet id and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_SUBNET_ID6_HOST

using IPv6 subnet %1, found host: %2

> This debug message includes the details of the host found using the DHCPv6 subnet id. The arguments specify subnet id and found host details respectively.

HOSTS_CFG_GET_ALL_SUBNET_ID_ADDRESS6

get all hosts with reservations for subnet id %1 and IPv6 address %2

> This debug message is issued when starting to retrieve all hosts connected to the specific subnet and having the specific IPv6 address reserved. The arguments specify subnet id and IPv6 address respectively.

HOSTS_CFG_GET_ALL_SUBNET_ID_ADDRESS6_COUNT

using subnet id %1 and address %2, found %3 host(s)

> This debug message include the details of the host found using the subnet id and address. The arguments specify subnet id, address and the number of hosts found respectively.

HOSTS_CFG_GET_ALL_SUBNET_ID_ADDRESS6_HOST

using subnet id %1 and address %2, found host: %3

> This debug message includes the details of the host found using the subnet id and address. The arguments specify subnet id, address and the number of hosts found respectively. found host details respectively.

HOSTS_CFG_GET_ONE_PREFIX

get one host with reservation for prefix %1/%2

> This debug message is issued when starting to retrieve a host having a reservation for a specified prefix. The arguments specify a prefix and prefix length.

HOSTS_CFG_GET_ONE_PREFIX_HOST

using prefix %1/%2, found host: %3

> This debug message includes the details of the host found using the specific prefix/prefix length. The arguments specify prefix, prefix length and host details respectively.

HOSTS_CFG_GET_ONE_PREFIX_NULL

host not found using prefix %1/%2

> This debug message is issued when no host was found for a specified prefix and prefix length.

HOSTS_CFG_GET_ONE_SUBNET_ID_ADDRESS4

get one host with reservation for subnet id %1 and IPv4 address %2

> This debug message is issued when starting to retrieve a host connected to the specific subnet and having the specific IPv4 address reserved. The arguments specify subnet id and IPv4 address respectively.

HOSTS_CFG_GET_ONE_SUBNET_ID_ADDRESS4_HOST

using subnet id %1 and address %2, found host: %3

> This debug message logs the details of the host found using the subnet id and IPv4 address.

HOSTS_CFG_GET_ONE_SUBNET_ID_ADDRESS4_NULL

host not found using subnet id %1 and address %2

> This debug message is issued when no host was found for the specified subnet id and IPv4 address.

HOSTS_CFG_GET_ONE_SUBNET_ID_ADDRESS6

get one host with reservation for subnet id %1 and including IPv6 address %2

> This debug message is issued when starting to retrieve a host connected to the specific subnet and having the specific IPv6 address reserved. The arguments specify subnet id and IPv6 address respectively.

HOSTS_CFG_GET_ONE_SUBNET_ID_ADDRESS6_HOST

using subnet id %1 and address %2, found host: %3

> This debug message logs the details of the host found using the subnet id and IPv6 address.

HOSTS_CFG_GET_ONE_SUBNET_ID_ADDRESS6_NULL

host not found using subnet id %1 and address %2

> This debug message is issued when no host was found using the specified subnet if and IPv6 address.

HOSTS_CFG_GET_ONE_SUBNET_ID_IDENTIFIER

get one host with %1 reservation for subnet id %2, identified by %3

> This debug message is issued when starting to retrieve a host holding IPv4 or IPv6 reservations, which is connected to a specific subnet and is identified by a specific unique identifier. The first argument identifies if the IPv4 or IPv6 reservation is desired.

HOSTS_CFG_GET_ONE_SUBNET_ID_IDENTIFIER_HOST

using subnet id %1 and identifier %2, found host: %3

> This debug message includes the details of a host found using a subnet id and specific host identifier.

HOSTS_CFG_GET_ONE_SUBNET_ID_IDENTIFIER_NULL

host not found using subnet id %1 and identifier %2

> This debug message is issued when no host was found using the specified subnet id and host identifier.

HOSTS_MGR_ALTERNATE_GET4_SUBNET_ID_ADDRESS4

trying alternate sources for host using subnet id %1 and address %2

> This debug message is issued when the Host Manager doesn't find the host connected to the specific subnet and having the reservation for the specific IPv4 address, and it is starting to search for this host in alternate host data sources.

HOSTS_MGR_ALTERNATE_GET4_SUBNET_ID_IDENTIFIER

get one host with IPv4 reservation for subnet id %1, identified by %2

> This debug message is issued when starting to retrieve a host holding IPv4 reservation, which is connected to a specific subnet and is identified by a specific unique identifier.

HOSTS_MGR_ALTERNATE_GET4_SUBNET_ID_IDENTIFIER_HOST

using subnet id %1 and identifier %2, found in %3 host: %4

This debug message includes the details of a host returned by an alternate hosts data source using a subnet id and specific host identifier.

HOSTS_MGR_ALTERNATE_GET4_SUBNET_ID_IDENTIFIER_NULL

host not found using subnet id %1 and identifier %2

This debug message is issued when no host was found using the specified subnet id and host identifier.

HOSTS_MGR_ALTERNATE_GET6_PREFIX

trying alternate sources for host using prefix %1/%2

This debug message is issued when the Host Manager doesn't find the host connected to the specific subnet and having the reservation for the specified prefix, and it is starting to search for this host in alternate host data sources.

HOSTS_MGR_ALTERNATE_GET6_SUBNET_ID_ADDRESS6

trying alternate sources for host using subnet id %1 and IPv6 address %2

This debug message is issued when the Host Manager doesn't find the host connected to the specific subnet and having the reservation for the specified IPv6 address, and it is starting to search for this host in alternate host data sources.

HOSTS_MGR_ALTERNATE_GET6_SUBNET_ID_IDENTIFIER

get one host with IPv6 reservation for subnet id %1, identified by %2

This debug message is issued when starting to retrieve a host holding IPv4 reservation, which is connected to a specific subnet and is identified by a specific unique identifier.

HOSTS_MGR_ALTERNATE_GET6_SUBNET_ID_IDENTIFIER_HOST

using subnet id %1 and identifier %2, found in %3 host: %4

This debug message includes the details of a host returned by an alternate host data source using a subnet id and specific host identifier.

25.17 HTTP

HTTP_BAD_CLIENT_REQUEST_RECEIVED

bad request received from %1: %2

This debug message is issued when an HTTP client sends malformed request to the server. This includes HTTP requests using unexpected content types, including malformed JSON etc. The first argument specifies an address of the remote endpoint which sent the request. The second argument provides a detailed error message.

HTTP_BAD_CLIENT_REQUEST_RECEIVED_DETAILS

detailed information about bad request received from %1:n%2

This debug message is issued when an HTTP client sends malformed request to the server. It includes detailed information about the received request rejected by the server. The first argument specifies an address of the remote endpoint which sent the request. The second argument provides a request in the textual format. The request is truncated by the logger if it is too large to be printed.

HTTP_BAD_SERVER_RESPONSE_RECEIVED

bad response received when communicating with %1: %2

This debug message is issued when an HTTP client fails to receive a response from the server or when this response is malformed. The first argument specifies the server URL. The second argument provides a detailed error message.

HTTP_BAD_SERVER_RESPONSE_RECEIVED_DETAILS

detailed information about bad response received from %1:n%2

This debug message is issued when an HTTP client receives malformed response from the server. The first argument specifies an URL of the server. The second argument provides a response in the textual format. The request is truncated by the logger if it is too large to be printed.

HTTP_CLIENT_REQUEST_AUTHORIZED

received HTTP request authorized for '%1'

This debug message is issued when the server receives with a matching authentication header. The argument provides the user name.

HTTP_CLIENT_REQUEST_BAD_AUTH_HEADER

received HTTP request with malformed authentication header: %1

This information message is issued when the server receives a request with a malformed authentication header. The argument explains the problem.

HTTP_CLIENT_REQUEST_NOT_AUTHORIZED

received HTTP request with not matching authentication header

This information message is issued when the server receives a request with authentication header carrying not recognized credential: the user provided incorrect user id and/or password.

HTTP_CLIENT_REQUEST_NO_AUTH_HEADER

received HTTP request without required authentication header

This information message is issued when the server receives a request without a required authentication header.

HTTP_CLIENT_REQUEST_RECEIVED

received HTTP request from %1

This debug message is issued when the server finished receiving a HTTP request from the remote endpoint. The address of the remote endpoint is specified as an argument.

HTTP_CLIENT_REQUEST_RECEIVED_DETAILS

detailed information about well formed request received from %1:n%2

This debug message is issued when the HTTP server receives a well formed request. It includes detailed information about the received request. The first argument specifies an address of the remote endpoint which sent the request. The second argument provides the request in the textual format. The request is truncated by the logger if it is too large to be printed.

HTTP_CLIENT_REQUEST_SEND

sending HTTP request %1 to %2

This debug message is issued when the client is starting to send a HTTP request to a server. The first argument holds basic information about the request (HTTP version number and status code). The second argument specifies a URL of the server.

HTTP_CLIENT_REQUEST_SEND_DETAILS

detailed information about request sent to %1:n%2

This debug message is issued right before the client sends an HTTP request to the server. It includes detailed information about the request. The first argument specifies an URL of the server to which the request is being sent. The second argument provides the request in the textual form. The request is truncated by the logger if it is too large to be printed.

HTTP_CLIENT_REQUEST_TIMEOUT_OCCURRED

HTTP request timeout occurred when communicating with %1

This debug message is issued when the HTTP request timeout has occurred and the server is going to send a response with Http Request timeout status code.

HTTP_CONNECTION_CLOSE_CALLBACK_FAILED

Connection close callback threw an exception

This is an error message emitted when the close connection callback registered on the connection failed unexpectedly. This is a programmatic error that should be submitted as a bug.

HTTP_CONNECTION_STOP

stopping HTTP connection from %1

This debug message is issued when one of the HTTP connections is stopped. The connection can be stopped as a result of an error or after the successful message exchange with a client.

HTTP_CONNECTION_STOP_FAILED

stopping HTTP connection failed

This error message is issued when an error occurred during closing a HTTP connection with a client.

HTTP_DATA_RECEIVED

received %1 bytes from %2

This debug message is issued when the server receives a chunk of data from the remote endpoint. This may include the whole request or only a part of the request. The first argument specifies the amount of received data. The second argument specifies an address of the remote endpoint which produced the data.

HTTP_IDLE_CONNECTION_TIMEOUT_OCCURRED

closing persistent connection with %1 as a result of a timeout

This debug message is issued when the persistent HTTP connection is being closed as a result of being idle.

HTTP_PREMATURE_CONNECTION_TIMEOUT_OCCURRED

premature connection timeout occurred, possibly caused by system clock change

This warning message is issued when unexpected timeout occurred during the transaction. This is proven to occur when the system clock is moved manually or as a result of synchronization with a time server. Any ongoing transactions will be interrupted. New transactions should be conducted normally.

HTTP_REQUEST_RECEIVE_START

start receiving request from %1 with timeout %2

This debug message is issued when the server starts receiving new request over the established connection. The first argument specifies the address of the remote endpoint. The second argument specifies request timeout in seconds.

HTTP_SERVER_RESPONSE_RECEIVED

received HTTP response from %1

This debug message is issued when the client finished receiving an HTTP response from the server. The URL of the server is specified as an argument.

HTTP_SERVER_RESPONSE_RECEIVED_DETAILS

detailed information about well formed response received from %1:n%2

This debug message is issued when the HTTP client receives a well formed response from the server. It includes detailed information about the received response. The first argument specifies a URL of the server which sent the response. The second argument provides the response in the textual format. The response is truncated by the logger if it is too large to be printed.

HTTP_SERVER_RESPONSE_SEND

sending HTTP response %1 to %2

This debug message is issued when the server is starting to send a HTTP response to a remote endpoint. The first argument holds basic information about the response (HTTP version number and status code). The second argument specifies an address of the remote endpoint.

25.18 LEASE

LEASE_CMDS_ADD4

lease4-add command successful (parameters: %1)

The lease4-add command has been successful. Parameters of the host added are logged.

LEASE_CMDS_ADD4_FAILED

lease4-add command failed (parameters: %1, reason: %2)

The lease4-add command has failed. Both the reason as well as the parameters passed are logged.

LEASE_CMDS_ADD6

lease6-add command successful (parameters: %1)

The lease6-add command has been successful. Parameters of the host added are logged.

LEASE_CMDS_ADD6_FAILED

Lease6-add command failed (parameters: %1, reason: %2)

The lease6-add command has failed. Both the reason as well as the parameters passed are logged.

LEASE_CMDS_DEINIT_FAILED

unloading Lease Commands hooks library failed: %1

This error message indicates an error during unloading the Lease Commands hooks library. The details of the error are provided as argument of the log message.

LEASE_CMDS_DEINIT_OK

unloading Lease Commands hooks library successful

This info message indicates that the Lease Commands hooks library has been removed successfully.

LEASE_CMDS_DEL4

lease4-del command successful (parameters: %1)

The attempt to delete an IPv4 lease (lease4-del command) has been successful. Parameters of the host removed are logged.

LEASE_CMDS_DEL4_FAILED

lease4-del command failed (parameters: %1, reason: %2)

> The attempt to delete an IPv4 lease (lease4-del command) has failed. Both the reason as well as the parameters passed are logged.

LEASE_CMDS_DEL6

lease4-del command successful (parameters: %1)

> The attempt to delete an IPv4 lease (lease4-del command) has been successful. Parameters of the host removed are logged.

LEASE_CMDS_DEL6_FAILED

lease6-del command failed (parameters: %1, reason: %2)

> The attempt to delete an IPv6 lease (lease4-del command) has failed. Both the reason as well as the parameters passed are logged.

LEASE_CMDS_INIT_FAILED

loading Lease Commands hooks library failed: %1

> This error message indicates an error during loading the Lease Commands hooks library. The details of the error are provided as argument of the log message.

LEASE_CMDS_INIT_OK

loading Lease Commands hooks library successful

> This info message indicates that the Lease Commands hooks library has been loaded successfully. Enjoy!

LEASE_CMDS_RESEND_DDNS4

lease4-resend-ddns command successful: %1

> A request to update DNS for the requested IPv4 lease has been successfully queued for transmission to kea-dhcp-ddns.

LEASE_CMDS_RESEND_DDNS4_FAILED

lease4-resend-ddns command failed: %1

> A request to update DNS for the requested IPv4 lease has failed. The reason for the failure is logged.

LEASE_CMDS_RESEND_DDNS6

lease6-resend-ddns command successful: %1

> A request to update DNS for the requested IPv6 lease has been successfully queued for transmission to kea-dhcp-ddns.

25.19 LFC

LFC_FAIL_PID_CREATE

: %1

> This message is issued if LFC detected a failure when trying to create the PID file. It includes a more specific error string.

LFC_FAIL_PID_DEL

: %1

> This message is issued if LFC detected a failure when trying to delete the PID file. It includes a more specific error string.

LFC_FAIL_PROCESS

: %1

> This message is issued if LFC detected a failure when trying to process the files. It includes a more specific error string.

LFC_FAIL_ROTATE

: %1

> This message is issued if LFC detected a failure when trying to rotate the files. It includes a more specific error string.

LFC_PROCESSING

Previous file: %1, copy file: %2

> This message is issued just before LFC starts processing the lease files.

LFC_READ_STATS

Leases: %1, attempts: %2, errors: %3.

> This message prints out the number of leases that were read, the number of attempts to read leases and the number of errors encountered while reading.

LFC_ROTATING

LFC rotating files

> This message is issued just before LFC starts rotating the lease files - removing the old and replacing them with the new.

LFC_RUNNING

LFC instance already running

> This message is issued if LFC detects that a previous copy of LFC may still be running via the PID check.

LFC_START

Starting lease file cleanup

> This message is issued as the LFC process starts.

LFC_TERMINATE

LFC finished processing

> This message is issued when the LFC process completes. It does not indicate that the process was successful only that it has finished.

25.20 LOGIMPL

LOGIMPL_ABOVE_MAX_DEBUG

debug level of %1 is too high and will be set to the maximum of %2

> A message from the interface to the underlying logger implementation reporting that the debug level (as set by an internally-created string DEBUGn, where n is an integer, e.g. DEBUG22) is above the

maximum allowed value and has been reduced to that value. The appearance of this message may indicate a programming error - please submit a bug report.

LOGIMPL_BAD_DEBUG_STRING

debug string '%1' has invalid format

> A message from the interface to the underlying logger implementation reporting that an internally-created string used to set the debug level is not of the correct format (it should be of the form DEBUGn, where n is an integer, e.g. DEBUG22). The appearance of this message indicates a programming error - please submit a bug report.

25.21 LOG

LOG_BAD_DESTINATION

unrecognized log destination: %1

> A logger destination value was given that was not recognized. The destination should be one of "console", "file", or "syslog".

LOG_BAD_SEVERITY

unrecognized log severity: %1

> A logger severity value was given that was not recognized. The severity should be one of "DEBUG", "INFO", "WARN", "ERROR", "FATAL" or "NONE".

LOG_BAD_STREAM

bad log console output stream: %1

> Logging has been configured so that output is written to the terminal (console) but the stream on which it is to be written is not recognized. Allowed values are "stdout" and "stderr".

LOG_DUPLICATE_MESSAGE_ID

duplicate message ID (%1) in compiled code

> During start-up, Kea detected that the given message identification had been defined multiple times in the Kea code. This indicates a programming error; please submit a bug report.

LOG_DUPLICATE_NAMESPACE

line %1: duplicate $NAMESPACE directive found

> When reading a message file, more than one $NAMESPACE directive was found. (This directive is used to set a C++ namespace when generating header files during software development.) Such a condition is regarded as an error and the read will be abandoned.

LOG_INPUT_OPEN_FAIL

unable to open message file %1 for input: %2

> The program was not able to open the specified input message file for the reason given.

LOG_INVALID_MESSAGE_ID

line %1: invalid message identification '%2'

> An invalid message identification (ID) has been found during the read of a message file. Message IDs should comprise only alphanumeric characters and the underscore, and should not start with a digit.

LOG_NAMESPACE_EXTRA_ARGS

line %1: $NAMESPACE directive has too many arguments

> The $NAMESPACE directive in a message file takes a single argument, a namespace in which all the generated symbol names are placed. This error is generated when the compiler finds a $NAMESPACE directive with more than one argument.

LOG_NAMESPACE_INVALID_ARG

line %1: $NAMESPACE directive has an invalid argument ('%2')

> The $NAMESPACE argument in a message file should be a valid C++ namespace. This message is output if the simple check on the syntax of the string carried out by the reader fails.

LOG_NAMESPACE_NO_ARGS

line %1: no arguments were given to the $NAMESPACE directive

> The $NAMESPACE directive in a message file takes a single argument, a C++ namespace in which all the generated symbol names are placed. This error is generated when the compiler finds a $NAMESPACE directive with no arguments.

LOG_NO_MESSAGE_ID

line %1: message definition line found without a message ID

> Within a message file, message are defined by lines starting with a "%". The rest of the line should comprise the message ID and text describing the message. This error indicates the message compiler found a line in the message file comprising just the "%" and nothing else.

LOG_NO_MESSAGE_TEXT

line %1: line found containing a message ID ('%2') and no text

> Within a message file, message are defined by lines starting with a "%". The rest of the line should comprise the message ID and text describing the message. This error indicates the message compiler found a line in the message file comprising just the "%" and message identification, but no text.

LOG_NO_SUCH_MESSAGE

could not replace message text for '%1': no such message

> During start-up a local message file was read. A line with the listed message identification was found in the file, but the identification is not one contained in the compiled-in message dictionary. This message may appear a number of times in the file, once for every such unknown message identification. There may be several reasons why this message may appear: - The message ID has been mis-spelled in the local message file. - The program outputting the message may not use that particular message (e.g. it originates in a module not used by the program). - The local file was written for an earlier version of the Kea software and the later version no longer generates that message. Whatever the reason, there is no impact on the operation of Kea.

LOG_OPEN_OUTPUT_FAIL

unable to open %1 for output: %2

> Originating within the logging code, the program was not able to open the specified output file for the reason given.

LOG_PREFIX_EXTRA_ARGS

line %1: $PREFIX directive has too many arguments

> Within a message file, the $PREFIX directive takes a single argument, a prefix to be added to the symbol names when a C++ file is created. This error is generated when the compiler finds a $PREFIX directive

with more than one argument. Note: the $PREFIX directive is deprecated and will be removed in a future version of Kea.

LOG_PREFIX_INVALID_ARG

line %1: $PREFIX directive has an invalid argument ('%2')

Within a message file, the $PREFIX directive takes a single argument, a prefix to be added to the symbol names when a C++ file is created. As such, it must adhere to restrictions on C++ symbol names (e.g. may only contain alphanumeric characters or underscores, and may nor start with a digit). A $PREFIX directive was found with an argument (given in the message) that violates those restrictions. Note: the $PREFIX directive is deprecated and will be removed in a future version of Kea.

LOG_READING_LOCAL_FILE

reading local message file %1

This is an informational message output by Kea when it starts to read a local message file. (A local message file may replace the text of one or more messages; the ID of the message will not be changed though.)

LOG_READ_ERROR

error reading from message file %1: %2

The specified error was encountered reading from the named message file.

LOG_UNRECOGNIZED_DIRECTIVE

line %1: unrecognized directive '%2'

Within a message file, a line starting with a dollar symbol was found (indicating the presence of a directive) but the first word on the line (shown in the message) was not recognized.

25.22 MYSQL

MYSQL_CB_CREATE_UPDATE_BY_POOL_OPTION4

create or update option pool start: %1 pool end: %2

Debug message issued when triggered an action to create or update option by pool

MYSQL_CB_CREATE_UPDATE_BY_POOL_OPTION6

create or update option pool start: %1 pool end: %2

Debug message issued when triggered an action to create or update option by pool

MYSQL_CB_CREATE_UPDATE_BY_PREFIX_OPTION6

create or update option prefix: %1 prefix len: %2

Debug message issued when triggered an action to create or update option by prefix

MYSQL_CB_CREATE_UPDATE_BY_SUBNET_ID_OPTION4

create or update option by subnet id: %1

Debug message issued when triggered an action to create or update option by subnet id

MYSQL_CB_CREATE_UPDATE_BY_SUBNET_ID_OPTION6

create or update option by subnet id: %1

Debug message issued when triggered an action to create or update option by subnet id

MYSQL_CB_CREATE_UPDATE_GLOBAL_PARAMETER4

create or update global parameter: %1

> Debug message issued when triggered an action to create or update global parameter

MYSQL_CB_CREATE_UPDATE_GLOBAL_PARAMETER6

create or update global parameter: %1

> Debug message issued when triggered an action to create or update global parameter

MYSQL_CB_CREATE_UPDATE_OPTION4

create or update option

> Debug message issued when triggered an action to create or update option

MYSQL_CB_CREATE_UPDATE_OPTION6

create or update option

> Debug message issued when triggered an action to create or update option

MYSQL_CB_CREATE_UPDATE_OPTION_DEF4

create or update option definition: %1 code: %2

> Debug message issued when triggered an action to create or update option definition

MYSQL_CB_CREATE_UPDATE_OPTION_DEF6

create or update option definition: %1 code: %2

> Debug message issued when triggered an action to create or update option definition

MYSQL_CB_CREATE_UPDATE_SERVER4

create or update server: %1

> Debug message issued when triggered an action to create or update a DHCPv4 server information.

MYSQL_CB_CREATE_UPDATE_SERVER6

create or update server: %1

> Debug message issued when triggered an action to create or update a DHCPv6 server information.

MYSQL_CB_CREATE_UPDATE_SHARED_NETWORK4

create or update shared network: %1

> Debug message issued when triggered an action to create or update shared network

MYSQL_CB_CREATE_UPDATE_SHARED_NETWORK6

create or update shared network: %1

> Debug message issued when triggered an action to create or update shared network

MYSQL_CB_CREATE_UPDATE_SHARED_NETWORK_OPTION4

create or update shared network: %1 option

> Debug message issued when triggered an action to create or update shared network option

MYSQL_CB_CREATE_UPDATE_SHARED_NETWORK_OPTION6

create or update shared network: %1 option

> Debug message issued when triggered an action to create or update shared network option

MYSQL_CB_CREATE_UPDATE_SUBNET4

create or update subnet: %1

> Debug message issued when triggered an action to create or update subnet

MYSQL_CB_CREATE_UPDATE_SUBNET6

create or update subnet: %1

> Debug message issued when triggered an action to create or update subnet

MYSQL_CB_DEINIT_OK

unloading MYSQL CB hooks library successful

> This informational message indicates that the MySQL Configuration Backend hooks library has been unloaded successfully.

MYSQL_CB_DELETE_ALL_GLOBAL_PARAMETERS4

delete all global parameters

> Debug message issued when triggered an action to delete all global parameters

MYSQL_CB_DELETE_ALL_GLOBAL_PARAMETERS4_RESULT

deleted: %1 entries

> Debug message indicating the result of an action to delete all global parameters

MYSQL_CB_DELETE_ALL_GLOBAL_PARAMETERS6

delete all global parameters

> Debug message issued when triggered an action to delete all global parameters

MYSQL_CB_DELETE_ALL_GLOBAL_PARAMETERS6_RESULT

deleted: %1 entries

> Debug message indicating the result of an action to delete all global parameters

MYSQL_CB_DELETE_ALL_OPTION_DEFS4

delete all option definitions

> Debug message issued when triggered an action to delete all option definitions

MYSQL_CB_DELETE_ALL_OPTION_DEFS4_RESULT

deleted: %1 entries

> Debug message indicating the result of an action to delete all option definitions

MYSQL_CB_DELETE_ALL_OPTION_DEFS6

delete all option definitions

> Debug message issued when triggered an action to delete all option definitions

MYSQL_CB_DELETE_ALL_OPTION_DEFS6_RESULT

deleted: %1 entries

> Debug message indicating the result of an action to delete all option definitions

MYSQL_CB_DELETE_ALL_SERVERS4

delete all DHCPv4 servers

Debug message issued when triggered an action to delete all servers.

MYSQL_CB_DELETE_ALL_SERVERS4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete all servers.

MYSQL_CB_DELETE_ALL_SERVERS6

delete all DHCPv6 servers

Debug message issued when triggered an action to delete all servers.

MYSQL_CB_DELETE_ALL_SERVERS6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete all servers.

MYSQL_CB_DELETE_ALL_SHARED_NETWORKS4

delete all shared networks

Debug message issued when triggered an action to delete all shared networks

MYSQL_CB_DELETE_ALL_SHARED_NETWORKS4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete all shared networks

MYSQL_CB_DELETE_ALL_SHARED_NETWORKS6

delete all shared networks

Debug message issued when triggered an action to delete all shared networks

MYSQL_CB_DELETE_ALL_SHARED_NETWORKS6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete all shared networks

MYSQL_CB_DELETE_ALL_SUBNETS4

delete all subnets

Debug message issued when triggered an action to delete all subnets

MYSQL_CB_DELETE_ALL_SUBNETS4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete all subnets

MYSQL_CB_DELETE_ALL_SUBNETS6

delete all subnets

Debug message issued when triggered an action to delete all subnets

MYSQL_CB_DELETE_ALL_SUBNETS6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete all subnets

MYSQL_CB_DELETE_BY_POOL_OPTION4

delete pool start: %1 pool end: %2 option code: %3 space: %4

Debug message issued when triggered an action to delete option by pool

MYSQL_CB_DELETE_BY_POOL_OPTION4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option by pool

MYSQL_CB_DELETE_BY_POOL_OPTION6

delete pool start: %1 pool end: %2 option code: %3 space: %4

Debug message issued when triggered an action to delete option by pool

MYSQL_CB_DELETE_BY_POOL_OPTION6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option by pool

MYSQL_CB_DELETE_BY_POOL_PREFIX_OPTION6

delete prefix: %1 prefix len: %2 option code: %3 space: %4

Debug message issued when triggered an action to delete option by prefix

MYSQL_CB_DELETE_BY_POOL_PREFIX_OPTION6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option by prefix

MYSQL_CB_DELETE_BY_PREFIX_SUBNET4

delete subnet by prefix: %1

Debug message issued when triggered an action to delete subnet by prefix

MYSQL_CB_DELETE_BY_PREFIX_SUBNET4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete subnet by prefix

MYSQL_CB_DELETE_BY_PREFIX_SUBNET6

delete subnet by prefix: %1

Debug message issued when triggered an action to delete subnet by prefix

MYSQL_CB_DELETE_BY_PREFIX_SUBNET6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete subnet by prefix

MYSQL_CB_DELETE_BY_SUBNET_ID_OPTION4

delete by subnet id: %1 option code: %2 space: %3

Debug message issued when triggered an action to delete option by subnet id

MYSQL_CB_DELETE_BY_SUBNET_ID_OPTION4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option by subnet id

MYSQL_CB_DELETE_BY_SUBNET_ID_OPTION6

delete by subnet id: %1 option code: %2 space: %3

Debug message issued when triggered an action to delete option by subnet id

MYSQL_CB_DELETE_BY_SUBNET_ID_OPTION6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option by subnet id

MYSQL_CB_DELETE_BY_SUBNET_ID_SUBNET4

delete subnet by subnet id: %1

Debug message issued when triggered an action to delete subnet by subnet id

MYSQL_CB_DELETE_BY_SUBNET_ID_SUBNET4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete subnet by subnet id

MYSQL_CB_DELETE_BY_SUBNET_ID_SUBNET6

delete subnet by subnet id: %1

Debug message issued when triggered an action to delete subnet by subnet id

MYSQL_CB_DELETE_BY_SUBNET_ID_SUBNET6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete subnet by subnet id

MYSQL_CB_DELETE_GLOBAL_PARAMETER4

delete global parameter: %1

Debug message issued when triggered an action to delete global parameter

MYSQL_CB_DELETE_GLOBAL_PARAMETER4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete global parameter

MYSQL_CB_DELETE_GLOBAL_PARAMETER6

delete global parameter: %1

Debug message issued when triggered an action to delete global parameter

MYSQL_CB_DELETE_GLOBAL_PARAMETER6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete global parameter

MYSQL_CB_DELETE_OPTION4

delete option code: %1 space: %2

Debug message issued when triggered an action to delete option

MYSQL_CB_DELETE_OPTION4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option

MYSQL_CB_DELETE_OPTION6

delete option code: %1 space: %2

Debug message issued when triggered an action to delete option

MYSQL_CB_DELETE_OPTION6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option

MYSQL_CB_DELETE_OPTION_DEF4

delete option definition code: %1 space: %2

Debug message issued when triggered an action to delete option definition

MYSQL_CB_DELETE_OPTION_DEF4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option definition

MYSQL_CB_DELETE_OPTION_DEF6

delete option definition code: %1 space: %2

Debug message issued when triggered an action to delete option definition

MYSQL_CB_DELETE_OPTION_DEF6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete option definition

MYSQL_CB_DELETE_SERVER4

delete DHCPv4 server: %1

Debug message issued when triggered an action to delete a server.

MYSQL_CB_DELETE_SERVER4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete a server.

MYSQL_CB_DELETE_SERVER6

delete DHCPv6 server: %1

Debug message issued when triggered an action to delete a server.

MYSQL_CB_DELETE_SERVER6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete a server.

MYSQL_CB_DELETE_SHARED_NETWORK4

delete shared network: %1

Debug message issued when triggered an action to delete shared network

MYSQL_CB_DELETE_SHARED_NETWORK4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete shared network

MYSQL_CB_DELETE_SHARED_NETWORK6

delete shared network: %1

Debug message issued when triggered an action to delete shared network

MYSQL_CB_DELETE_SHARED_NETWORK6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete shared network

MYSQL_CB_DELETE_SHARED_NETWORK_OPTION4

delete shared network: %1 option code: %2 space: %3

Debug message issued when triggered an action to delete shared network option

MYSQL_CB_DELETE_SHARED_NETWORK_OPTION4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete shared network option

MYSQL_CB_DELETE_SHARED_NETWORK_OPTION6

delete shared network: %1 option code: %2 space: %3

Debug message issued when triggered an action to delete shared network option

MYSQL_CB_DELETE_SHARED_NETWORK_OPTION6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete shared network option

MYSQL_CB_DELETE_SHARED_NETWORK_SUBNETS4

delete shared network: %1 subnets

Debug message issued when triggered an action to delete shared network subnets

MYSQL_CB_DELETE_SHARED_NETWORK_SUBNETS4_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete shared network subnets

MYSQL_CB_DELETE_SHARED_NETWORK_SUBNETS6

delete shared network: %1 subnets

Debug message issued when triggered an action to delete shared network subnets

MYSQL_CB_DELETE_SHARED_NETWORK_SUBNETS6_RESULT

deleted: %1 entries

Debug message indicating the result of an action to delete shared network subnets

MYSQL_CB_GET_ALL_GLOBAL_PARAMETERS4

retrieving all global parameters

Debug message issued when triggered an action to retrieve all global parameters

MYSQL_CB_GET_ALL_GLOBAL_PARAMETERS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all global parameters

MYSQL_CB_GET_ALL_GLOBAL_PARAMETERS6

retrieving all global parameters

Debug message issued when triggered an action to retrieve all global parameters

MYSQL_CB_GET_ALL_GLOBAL_PARAMETERS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all global parameters

MYSQL_CB_GET_ALL_OPTIONS4

retrieving all options

Debug message issued when triggered an action to retrieve all options

MYSQL_CB_GET_ALL_OPTIONS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all options

MYSQL_CB_GET_ALL_OPTIONS6

retrieving all options

Debug message issued when triggered an action to retrieve all options

MYSQL_CB_GET_ALL_OPTIONS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all options

MYSQL_CB_GET_ALL_OPTION_DEFS4

retrieving all option definitions

Debug message issued when triggered an action to retrieve all option definitions

MYSQL_CB_GET_ALL_OPTION_DEFS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all option definitions

MYSQL_CB_GET_ALL_OPTION_DEFS6

retrieving all option definitions

Debug message issued when triggered an action to retrieve all option definitions

MYSQL_CB_GET_ALL_OPTION_DEFS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all option definitions

MYSQL_CB_GET_ALL_SERVERS4

retrieving all servers

Debug message issued when triggered an action to retrieve all DHCPv4 servers

MYSQL_CB_GET_ALL_SERVERS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all DHCPv4 servers

MYSQL_CB_GET_ALL_SERVERS6

retrieving all DHCPv6 servers

Debug message issued when triggered an action to retrieve all DHCPv6 servers

MYSQL_CB_GET_ALL_SERVERS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all DHCPv6 servers

MYSQL_CB_GET_ALL_SHARED_NETWORKS4

retrieving all shared networks

Debug message issued when triggered an action to retrieve all shared networks

MYSQL_CB_GET_ALL_SHARED_NETWORKS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all shared networks

MYSQL_CB_GET_ALL_SHARED_NETWORKS6

retrieving all shared networks

Debug message issued when triggered an action to retrieve all shared networks

MYSQL_CB_GET_ALL_SHARED_NETWORKS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all shared networks

MYSQL_CB_GET_ALL_SUBNETS4

retrieving all subnets

Debug message issued when triggered an action to retrieve all subnets

MYSQL_CB_GET_ALL_SUBNETS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all subnets

MYSQL_CB_GET_ALL_SUBNETS6

retrieving all subnets

Debug message issued when triggered an action to retrieve all subnets

MYSQL_CB_GET_ALL_SUBNETS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve all subnets

MYSQL_CB_GET_GLOBAL_PARAMETER4

retrieving global parameter: %1

Debug message issued when triggered an action to retrieve global parameter

MYSQL_CB_GET_GLOBAL_PARAMETER6

retrieving global parameter: %1

Debug message issued when triggered an action to retrieve global parameter

MYSQL_CB_GET_HOST4

get host

Debug message issued when triggered an action to retrieve host

MYSQL_CB_GET_HOST6

get host

Debug message issued when triggered an action to retrieve host

MYSQL_CB_GET_MODIFIED_GLOBAL_PARAMETERS4

retrieving modified global parameters from: %1

Debug message issued when triggered an action to retrieve modified global parameters from specified time

MYSQL_CB_GET_MODIFIED_GLOBAL_PARAMETERS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified global parameters from specified time

MYSQL_CB_GET_MODIFIED_GLOBAL_PARAMETERS6

retrieving modified global parameters from: %1

Debug message issued when triggered an action to retrieve modified global parameters from specified time

MYSQL_CB_GET_MODIFIED_GLOBAL_PARAMETERS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified global parameters from specified time

MYSQL_CB_GET_MODIFIED_OPTIONS4

retrieving modified options from: %1

Debug message issued when triggered an action to retrieve modified options from specified time

MYSQL_CB_GET_MODIFIED_OPTIONS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified options from specified time

MYSQL_CB_GET_MODIFIED_OPTIONS6

retrieving modified options from: %1

Debug message issued when triggered an action to retrieve modified options from specified time

MYSQL_CB_GET_MODIFIED_OPTIONS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified options from specified time

MYSQL_CB_GET_MODIFIED_OPTION_DEFS4

retrieving modified option definitions from: %1

Debug message issued when triggered an action to retrieve modified option definitions from specified time

MYSQL_CB_GET_MODIFIED_OPTION_DEFS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified option definitions from specified time

MYSQL_CB_GET_MODIFIED_OPTION_DEFS6

retrieving modified option definitions from: %1

Debug message issued when triggered an action to retrieve modified option definitions from specified time

MYSQL_CB_GET_MODIFIED_OPTION_DEFS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified option definitions from specified time

MYSQL_CB_GET_MODIFIED_SHARED_NETWORKS4

retrieving modified shared networks from: %1

Debug message issued when triggered an action to retrieve modified shared networks from specified time

MYSQL_CB_GET_MODIFIED_SHARED_NETWORKS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified shared networks from specified time

MYSQL_CB_GET_MODIFIED_SHARED_NETWORKS6

retrieving modified shared networks from: %1

Debug message issued when triggered an action to retrieve modified shared networks from specified time

MYSQL_CB_GET_MODIFIED_SHARED_NETWORKS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified shared networks from specified time

MYSQL_CB_GET_MODIFIED_SUBNETS4

retrieving modified subnets from: %1

Debug message issued when triggered an action to retrieve modified subnets from specified time

MYSQL_CB_GET_MODIFIED_SUBNETS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified subnets from specified time

MYSQL_CB_GET_MODIFIED_SUBNETS6

retrieving modified subnets from: %1

Debug message issued when triggered an action to retrieve modified subnets from specified time

MYSQL_CB_GET_MODIFIED_SUBNETS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve modified subnets from specified time

MYSQL_CB_GET_OPTION4

retrieving option code: %1 space: %2

Debug message issued when triggered an action to retrieve option

MYSQL_CB_GET_OPTION6

retrieving option code: %1 space: %2

> Debug message issued when triggered an action to retrieve option

MYSQL_CB_GET_OPTION_DEF4

retrieving option definition code: %1 space: %2

> Debug message issued when triggered an action to retrieve option definition

MYSQL_CB_GET_OPTION_DEF6

retrieving option definition code: %1 space: %2

> Debug message issued when triggered an action to retrieve option definition

MYSQL_CB_GET_PORT4

get port

> Debug message issued when triggered an action to retrieve port

MYSQL_CB_GET_PORT6

get port

> Debug message issued when triggered an action to retrieve port

MYSQL_CB_GET_RECENT_AUDIT_ENTRIES4

retrieving audit entries from: %1 %2

> Debug message issued when triggered an action to retrieve audit entries from specified time and id.

MYSQL_CB_GET_RECENT_AUDIT_ENTRIES4_RESULT

retrieving: %1 elements

> Debug message indicating the result of an action to retrieve audit entries from specified time

MYSQL_CB_GET_RECENT_AUDIT_ENTRIES6

retrieving audit entries from: %1 %2

> Debug message issued when triggered an action to retrieve audit entries from specified time and id

MYSQL_CB_GET_RECENT_AUDIT_ENTRIES6_RESULT

retrieving: %1 elements

> Debug message indicating the result of an action to retrieve audit entries from specified time

MYSQL_CB_GET_SERVER4

retrieving DHCPv4 server: %1

> Debug message issued when triggered an action to retrieve a DHCPv4 server information.

MYSQL_CB_GET_SERVER6

retrieving DHCPv6 server: %1

> Debug message issued when triggered an action to retrieve a DHCPv6 server information.

MYSQL_CB_GET_SHARED_NETWORK4

retrieving shared network: %1

> Debug message issued when triggered an action to retrieve shared network

MYSQL_CB_GET_SHARED_NETWORK6

retrieving shared network: %1

Debug message issued when triggered an action to retrieve shared network

MYSQL_CB_GET_SHARED_NETWORK_SUBNETS4

retrieving shared network: %1 subnets

Debug message issued when triggered an action to retrieve shared network subnets

MYSQL_CB_GET_SHARED_NETWORK_SUBNETS4_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve shared network subnets

MYSQL_CB_GET_SHARED_NETWORK_SUBNETS6

retrieving shared network: %1 subnets

Debug message issued when triggered an action to retrieve shared network subnets

MYSQL_CB_GET_SHARED_NETWORK_SUBNETS6_RESULT

retrieving: %1 elements

Debug message indicating the result of an action to retrieve shared network subnets

MYSQL_CB_GET_SUBNET4_BY_PREFIX

retrieving subnet by prefix: %1

Debug message issued when triggered an action to retrieve subnet by prefix

MYSQL_CB_GET_SUBNET4_BY_SUBNET_ID

retrieving subnet by subnet id: %1

Debug message issued when triggered an action to retrieve subnet by subnet id

MYSQL_CB_GET_SUBNET6_BY_PREFIX

retrieving subnet by prefix: %1

Debug message issued when triggered an action to retrieve subnet by prefix

MYSQL_CB_GET_SUBNET6_BY_SUBNET_ID

retrieving subnet by subnet id: %1

Debug message issued when triggered an action to retrieve subnet by subnet id

MYSQL_CB_GET_TYPE4

get type

Debug message issued when triggered an action to retrieve type

MYSQL_CB_GET_TYPE6

get type

Debug message issued when triggered an action to retrieve type

MYSQL_CB_INIT_OK

loading MYSQL CB hooks library successful

This informational message indicates that the MySQL Configuration Backend hooks library has been loaded successfully.

MYSQL_CB_REGISTER_BACKEND_TYPE4

register backend

Debug message issued when triggered an action to register backend

MYSQL_CB_REGISTER_BACKEND_TYPE6

register backend

Debug message issued when triggered an action to register backend

MYSQL_CB_UNREGISTER_BACKEND_TYPE4

unregister backend

Debug message issued when triggered an action to unregister backend

25.23 NETCONF

NETCONF_BOOT_UPDATE_COMPLETED

Boot-update configuration completed for server %1

This informational message is issued when the initial configuration was retrieved from Netconf and successfully applied to Kea server.

NETCONF_CONFIG_CHANGED_DETAIL

YANG configuration changed: %1

This debug message indicates a YANG configuration change. The format is the change operation (created, modified, deleted or moved) followed by xpaths and values of old and new nodes.

NETCONF_CONFIG_CHANGE_EVENT

Received YANG configuration change %1 event

This informational message is issued when Netconf receives a YANG configuration change event. The type of event is printed.

NETCONF_CONFIG_CHECK_FAIL

Netconf configuration check failed: %1

This error message indicates that Netconf had failed configuration check. Details are provided. Additional details may be available in earlier log entries, possibly on lower levels.

NETCONF_CONFIG_FAIL

Netconf configuration failed: %1

This error message indicates that Netconf had failed configuration attempt. Details are provided. Additional details may be available in earlier log entries, possibly on lower levels.

NETCONF_FAILED

application experienced a fatal error: %1

This is a fatal error message issued when the Netconf application got an unrecoverable error from within the event loop.

NETCONF_GET_CONFIG

got configuration from %1 server: %2

> This debug message indicates that Netconf got the configuration from a Kea server. The server name and the retrieved configuration are printed.

NETCONF_GET_CONFIG_FAILED

getting configuration from %1 server failed: %2

> The error message indicates that Netconf got an error getting the configuration from a Kea server. Make sure that the server is up and running, has appropriate control socket defined and that the controls socket configuration on the server matches that of kea-netconf. The name of the server and the error are printed.

NETCONF_GET_CONFIG_STARTED

getting configuration from %1 server

> This informational message indicates that Netconf is trying to get the configuration from a Kea server.

NETCONF_LOG_CHANGE_FAIL

Netconf configuration change logging failed: %1

> The warning message indicates that the configuration change logging encountered an unexpected condition. Details of it will be logged.

NETCONF_MODULE_INSTALL

Sysrepo (un)installs a module: %1 (revision %2)

> This warning message indicates that sysrepo reports the installation or uninstallation of a module used by Kea. The name and revision of the module are printed.

NETCONF_MODULE_MISSING_ERR

Missing essential module %1 in sysrepo

> This fatal error message indicates that a module required by Netconf configuration is not available in the sysrepo repository. The name of the module is printed.

NETCONF_MODULE_MISSING_WARN

Missing module %1 in sysrepo

> This warning message indicates that a module used by Kea is not available in the sysrepo repository. The name of the module is printed.

NETCONF_MODULE_REVISION_ERR

Essential module %1 does have the right revision: expected %2, got %3

> This fatal error message indicates that a module required by Netconf configuration is not at the right revision in the sysrepo repository. The name, expected and available revisions of the module are printed.

NETCONF_MODULE_REVISION_WARN

Module %1 does have the right revision: expected %2, got %3

> This warning message indicates that a module used by Kea is not at the right revision in the sysrepo repository. The name, expected and available revisions of the module are printed.

NETCONF_RUN_EXIT

application is exiting the event loop

This is a debug message issued when the Netconf application exits its event loop. This is a normal step during kea-netconf shutdown.

NETCONF_SET_CONFIG

set configuration to %1 server: %2

This debug message indicates that Netconf set the configuration to a Kea server. The server name and the applied configuration are printed.

NETCONF_SET_CONFIG_FAILED

setting configuration to %1 server failed: %2

The error message indicates that Netconf got an error setting the configuration to a Kea server. Make sure that the server is up and running, has appropriate control socket defined and that the controls socket configuration on the server matches that of kea-netconf. The name of the server and the error are printed.

NETCONF_SET_CONFIG_STARTED

setting configuration to %1 server

This informational message indicates that Netconf is trying to set the configuration to a Kea server.

NETCONF_STARTED

Netconf (version %1) started

This informational message indicates that Netconf has processed all configuration information and is ready to begin processing. The version is also printed.

NETCONF_SUBSCRIBE_CONFIG

subscribing configuration changes for %1 server with %2 module

This information message indicates that Netconf is trying to subscribe configuration changes for a Kea server. The names of the server and the module are printed.

NETCONF_SUBSCRIBE_CONFIG_FAILED

subscribe configuration changes for %1 server with %2 module failed: %3

The error message indicates that Netconf got an error subscribing configuration changes for a Kea server. The names of the server and the module, and the error are printed.

NETCONF_UPDATE_CONFIG

updating configuration with %1 server: %2

This debug message indicates that Netconf update the configuration of a Kea server. The server name and the updated configuration are printed.

NETCONF_UPDATE_CONFIG_COMPLETED

completed updating configuration for %1 server

This informational message indicates that Netconf updated with success the configuration of a Kea server.

NETCONF_UPDATE_CONFIG_FAILED

updating configuration with %1 server: %2

The error message indicates that Netconf got an error updating the configuration of a Kea server. This includes a configuration rejected by a Kea server when it tried to apply it. The name of the server and the error are printed.

NETCONF_UPDATE_CONFIG_STARTED

started updating configuration for %1 server

> This informational message indicates that Netconf is trying to update the configuration of a Kea server.

NETCONF_VALIDATE_CONFIG

validating configuration with %1 server: %2

> This debug message indicates that Netconf is validating the configuration with a Kea server. The server name and the validated configuration are printed.

NETCONF_VALIDATE_CONFIG_COMPLETED

completed validating configuration for %1 server

> This informational message indicates that Netconf validated with success the configuration with a Kea server.

NETCONF_VALIDATE_CONFIG_FAILED

validating configuration with %1 server got an error: %2

> The error message indicates that Netconf got an error validating the configuration with a Kea server. This message is produced when exception is thrown during an attempt to validate received configuration. Additional explanation may be provided as a parameter. You may also take a look at earlier log messages. The name of the server and the error are printed.

NETCONF_VALIDATE_CONFIG_REJECTED

validating configuration with %1 server was rejected: %2

> The warning message indicates that Netconf got an error validating the configuration with a Kea server. This message is printed when the configuration was rejected during normal processing. Additional explanation may be provided as a parameter. You may also take a look at earlier log messages. The name of the server and the error are printed.

25.24 STAT

STAT_CMDS_DEINIT_FAILED

unloading Stat Commands hooks library failed: %1

> This error message indicates an error during unloading the Lease Commands hooks library. The details of the error are provided as argument of the log message.

STAT_CMDS_DEINIT_OK

unloading Stat Commands hooks library successful

> This info message indicates that the Stat Commands hooks library has been removed successfully.

STAT_CMDS_INIT_FAILED

loading Stat Commands hooks library failed: %1

> This error message indicates an error during loading the Lease Commands hooks library. The details of the error are provided as argument of the log message.

STAT_CMDS_INIT_OK

loading Stat Commands hooks library successful

> This info message indicates that the Stat Commands hooks library has been loaded successfully. Enjoy!

STAT_CMDS_LEASE4_GET

stat-lease4-get command successful, parameters: %1 rows found: %2

> The stat-lease4-get command has been successful. The log will contain the parameters supplied and the number of rows found.

STAT_CMDS_LEASE4_GET_FAILED

stat-lease4-get command failed: parameters: %1, reason: %2

> The stat-lease4-get command has failed. Both the parameters supplied and the reason for failure are logged.

STAT_CMDS_LEASE4_GET_INVALID

stat-lease4-get command is malformed or invalid, reason: %1

> The stat-lease4-get command was either malformed or contained invalid parameters. A detailed explanation should be logged.

STAT_CMDS_LEASE4_GET_NO_SUBNETS

stat-lease4-get, parameters: %1, %2"

> The parameters submitted with stat-lease4-get were valid but excluded all known subnets. The parameters supplied along with an explanation should be logged.

STAT_CMDS_LEASE6_GET

stat-lease6-get command successful, parameters: %1 rows found: %2

> The stat-lease6-get command has been successful. The log will contain the parameters supplied and the number of rows found.

STAT_CMDS_LEASE6_GET_FAILED

stat-lease4-get command failed: parameters: %1, reason: %2

> The stat-lease6-get command has failed. Both the parameters supplied and the reason for failure are logged.

STAT_CMDS_LEASE6_GET_INVALID

stat-lease6-get command is malformed or invalid, reason: %1

> The stat-lease6-get command was either malformed or contained invalid parameters. A detailed explanation should be logged.

25.25 USER

USER_CHK_HOOK_LOAD_ERROR

DHCP UserCheckHook could not be loaded: %1

> This is an error message issued when the DHCP UserCheckHook could not be loaded. The exact cause should be explained in the log message. User subnet selection will revert to default processing.

USER_CHK_HOOK_UNLOAD_ERROR

DHCP UserCheckHook an error occurred unloading the library: %1

> This is an error message issued when an error occurs while unloading the UserCheckHook library. This is unlikely to occur and normal operations of the library will likely resume when it is next loaded.

USER_CHK_SUBNET4_SELECT_ERROR

DHCP UserCheckHook an unexpected error occurred in subnet4_select callout: %1

> This is an error message issued when the DHCP UserCheckHook subnet4_select hook encounters an unexpected error. The message should contain a more detailed explanation.

USER_CHK_SUBNET4_SELECT_REGISTRY_NULL

DHCP UserCheckHook UserRegistry has not been created.

> This is an error message issued when the DHCP UserCheckHook subnet4_select hook has been invoked but the UserRegistry has not been created. This is a programmatic error and should not occur.

USER_CHK_SUBNET6_SELECT_ERROR

DHCP UserCheckHook an unexpected error occurred in subnet6_select callout: %1

> This is an error message issued when the DHCP UserCheckHook subnet6_select hook encounters an unexpected error. The message should contain a more detailed explanation.

CHAPTER

TWENTYSIX

KEA FLOW DIAGRAMS

These flow diagrams describe the Kea DHCPv4 server implementation. They may be useful for system administrators to understand for several reasons. In order to design a configuration that will result in clients getting the intended addresses and options it is important to understand the sequence of request processing steps. For example, Kea will iterate looking for a suitable address, and will conditionally accept the first available address, so the order in which addresses are evaluated matters.

It is also useful to understand Kea's processing logic because there are configuration choices which can make the process far more efficient. Kea is very flexible so that it can be applied to very different use cases and in different environments. In an environment where throughput and efficiency are a priority, the administrator can choose to limit some of the processing steps. For example, it is possible to limit the number of different client identifiers Kea evaluates in looking for a host reservation, or even to skip the whole step of checking for host reservations.

These diagrams are focused on those aspects of Kea processing that will be most useful to operators. The diagrams illustrate DHCPv4 request processing, but most of the logic applies to DHCPv6. Following the title of each diagram is a Kea version number. Kea behavior has evolved over time, and the diagrams document the behavior as of the Kea version indicated. The diagrams are provided in the Kea source tree in UML (source), PNG and SVG formats.

26.1 Main Loop

The main loop is common to both DHCPv4 and DHPCv6 servers.

26.2 DHCPv4 Packet Processing

Next is the DHCPv4 packet processing, where we determine what sort of DHCP message this is, Discover, Request, Release, Decline or Inform. This diagram shows the general, high level flow for processing an inbound client DHCP packet (e.g. Discover, Request, Release, etc) from receipt to the server's response..

26.3 DHCP Request Processing

The following diagrams focus on DHCPREQUEST processing. This chart gives an overview of the process, beginning with subnet selection, proceeding to checking for Host Reservations and evaluating client classes. Finally, before acknowledging the lease, the options are evaluated and added to the message.

.

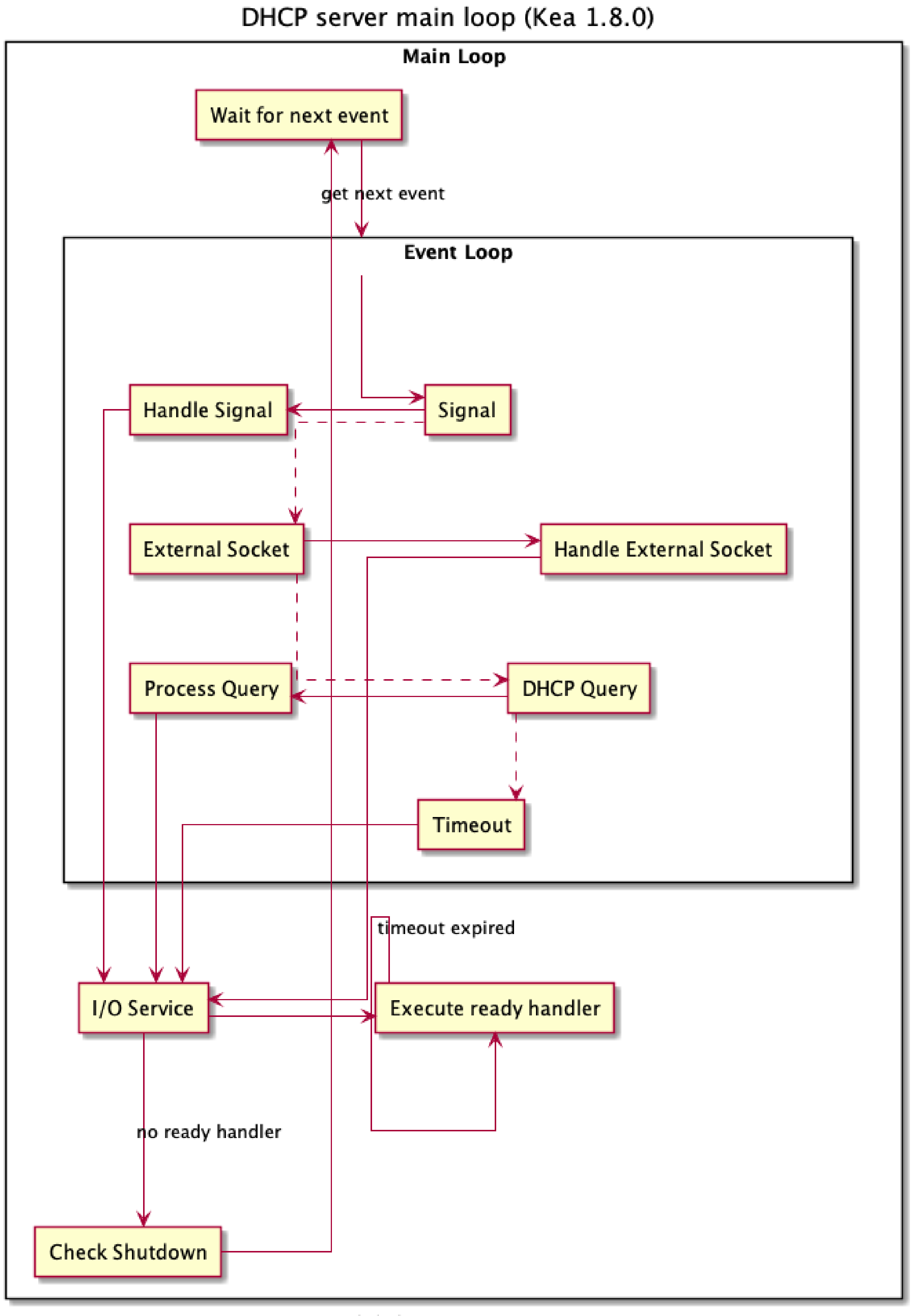

DHCP server main loop (Kea 1.8.0)
Main Loop
Wait for next event
get next event
Event Loop
Handle Signal
Signal
External Socket
Handle External Socket
Process Query
DHCP Query
Timeout
timeout expired
I/O Service
Execute ready handler
no ready handler
Check Shutdown
dashed arrow means priority

Fig. 26.1: DHCP server main loop

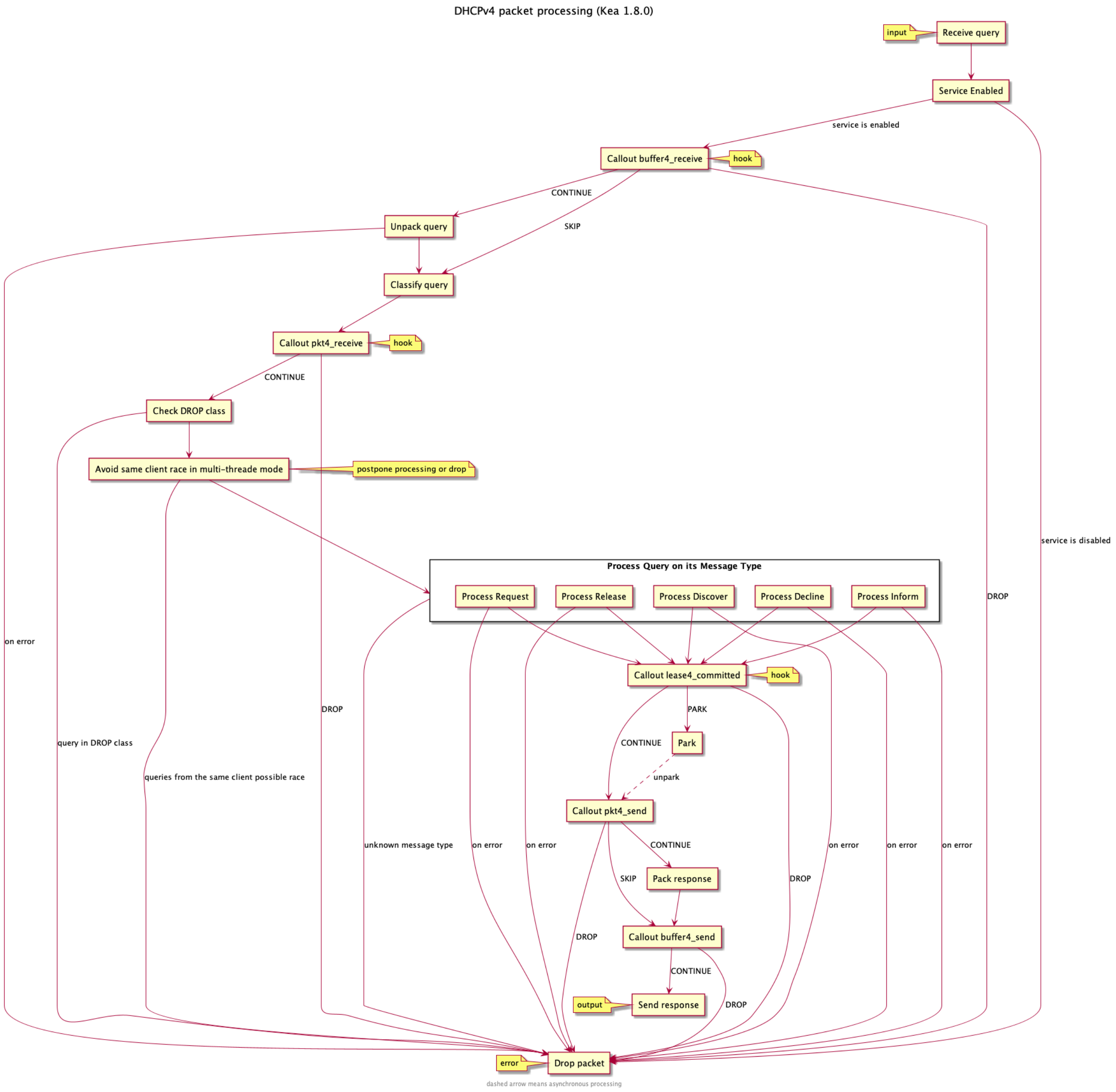
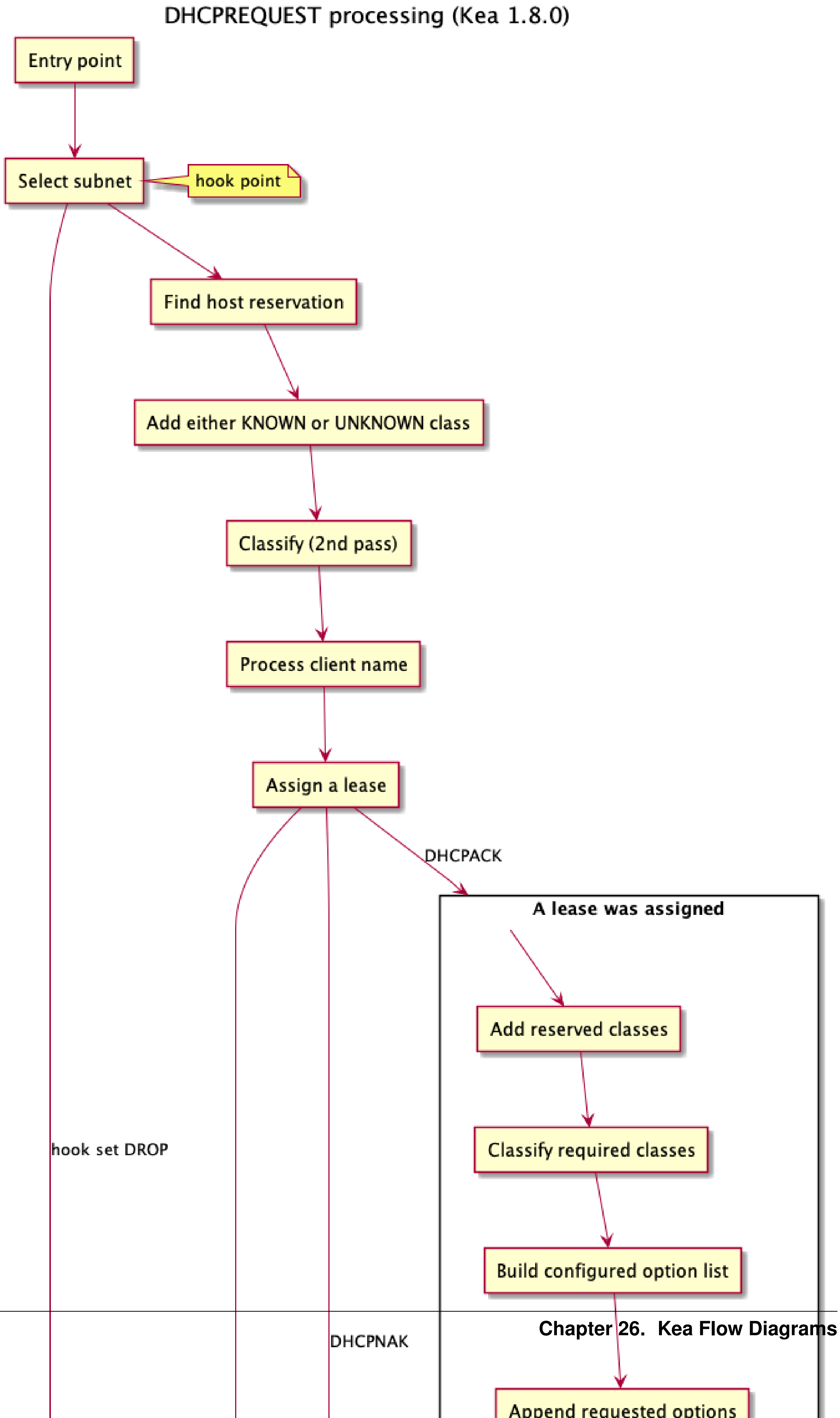

DHCPv4 packet processing (Kea 1.8.0)
input
Receive query
Service Enabled
service is enabled
Callout buffer4_receive
hook
CONTINUE
Unpack query
SKIP
Classify query
Callout pkt4_receive
hook
CONTINUE
Check DROP class
Avoid same client race in multi-threade mode
postpone processing or drop
Process Query on its Message Type
Process Request
Process Release
Process Discover
Process Decline
Process Inform
Callout lease4_committed
hook
PARK
Park
CONTINUE
unpark
Callout pkt4_send
CONTINUE
SKIP
Pack response
Callout buffer4_send
CONTINUE
output
Send response
DROP
DROP
DROP
DROP
DROP
error
Drop packet
on error
query in DROP class
queries from the same client possible race
unknown message type
on error
on error
on error
on error
on error
service is disabled
dashed arrow means asynchronous processing

Fig. 26.2: DHCPv4 packet processing


DHCPREQUEST processing (Kea 1.8.0)
Entry point
Select subnet
hook point
Find host reservation
Add either KNOWN or UNKNOWN class
Classify (2nd pass)
Process client name
Assign a lease
DHCPACK
A lease was assigned
Add reserved classes
Classify required classes
Build configured option list
hook set DROP
DHCPNAK
Append requested options

26.4 DHCPv4 Subnet Selection

Subnet selection is the process of choosing a subnet that is topologically appropriate for the client. Note that when the selected subnet is a member of a shared network the whole shared network is selected. During subnet selection the client class may be checked more than once while iterating through subnets, to determine if it is permitted in the selected subnet.

26.5 DHCPv4 Special Case of Double-booting

After the subnet selection and before the lease allocation the DHCPv4 server handles the special case of clients restarting with an image provided by PXE boot or bootp. Note that the Lease Request box is expanded below.

26.6 DHCPv4 Allocate Lease

This diagram provides the details of the processing the client request, showing renewing an existing lease, assigning a reserved lease and allocating an unreserved lease. The next diagram after this one shows the algorithm in more detail.

This diagram shows the algorithm used to validate a requested lease or select a new address to offer. The far right side of the diagram shows how a new address is selected when a new lease is required and the client has neither a requested address nor a reservation. Note that when a new lease is required and Kea iterates over pools and subnets, it starts with the subnet selected above in the subnet selection process.

Note: Declined addresses are included in the statistic for assigned addresses so the $assigned + free = total$ equation is true.

26.7 Lease States

This diagram illustrates the different lease states including the free one where no lease object exists.

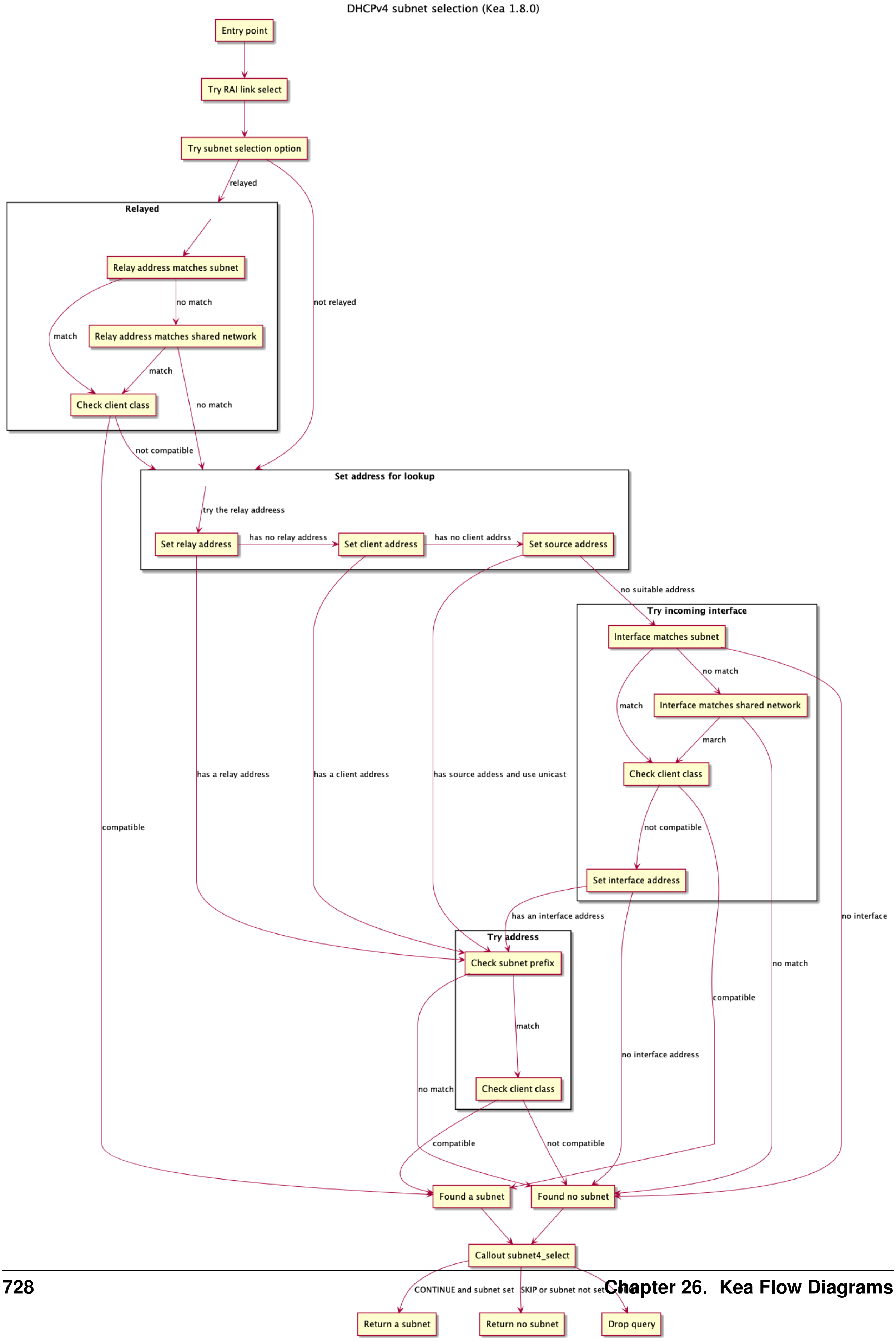
26.8 Checking for Host Reservations

The allocation engine checks for host reservations after selecting a subnet. This diagram shows the details of that operation. Subnet selection is based on network topology. Host reservations are primarily for assigning options, and options are evaluated after subnet selection. However if client classes are added in the host reservation, those will also be evaluated against the selected subnet in a further check (added in Kea 1.7.10). Kea includes several options to skip checking for host reservations, which can make this process much more efficient if you are not using reservations.

Note: To find a free lease the allocation engine begins with evaluating the most recently used subnet. The current subnet depends on the history of prior queries.

26.9 Building the Options List

Before sending a response, options are added:


DHCPv4 subnet selection (Kea 1.8.0)
Entry point
Try RAI link select
Try subnet selection option
relayed
not relayed
Relayed
Relay address matches subnet
no match
match
Relay address matches shared network
match
no match
Check client class
not compatible
compatible
Set address for lookup
try the relay addreess
Set relay address
has no relay address
Set client address
has no client addrss
Set source address
no suitable address
Try incoming interface
Interface matches subnet
no match
match
Interface matches shared network
march
Check client class
not compatible
Set interface address
has a relay address
has a client address
has source addess and use unicast
has an interface address
no interface
no match
compatible
no interface address
Try address
Check subnet prefix
match
Check client class
no match
compatible
not compatible
Found a subnet
Found no subnet
Callout subnet4_select
CONTINUE and subnet set
SKIP or subnet not set
Return a subnet
Return no subnet
Drop query

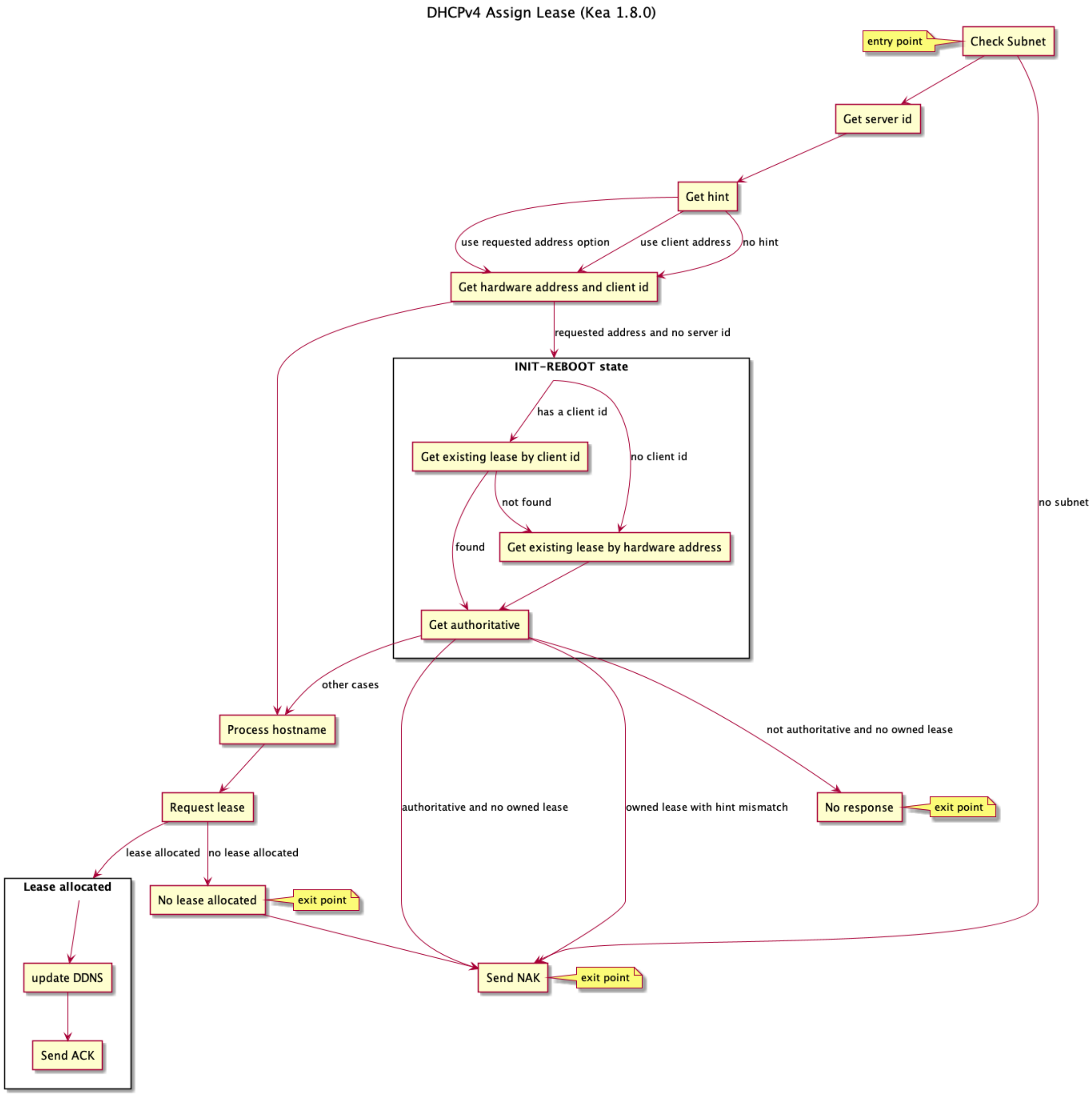

DHCPv4 Assign Lease (Kea 1.8.0)
entry point
Check Subnet
Get server id
Get hint
use requested address option
use client address
no hint
Get hardware address and client id
requested address and no server id
INIT-REBOOT state
has a client id
Get existing lease by client id
no client id
not found
found
Get existing lease by hardware address
Get authoritative
no subnet
other cases
Process hostname
not authoritative and no owned lease
Request lease
authoritative and no owned lease
owned lease with hint mismatch
No response
exit point
lease allocated
no lease allocated
Lease allocated
No lease allocated
exit point
update DDNS
Send NAK
exit point
Send ACK

Fig. 26.5: DHCPv4 Assign Lease

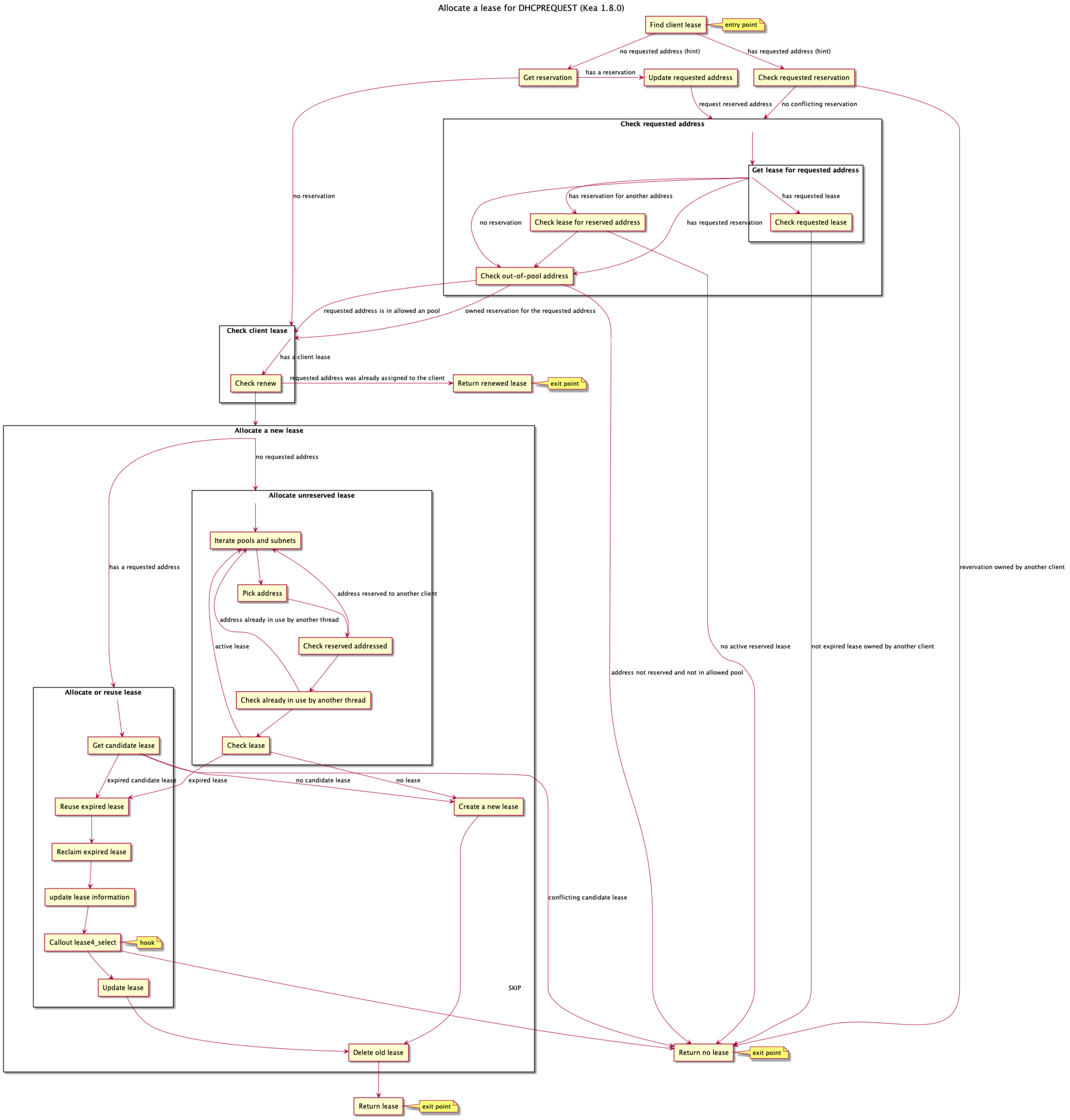
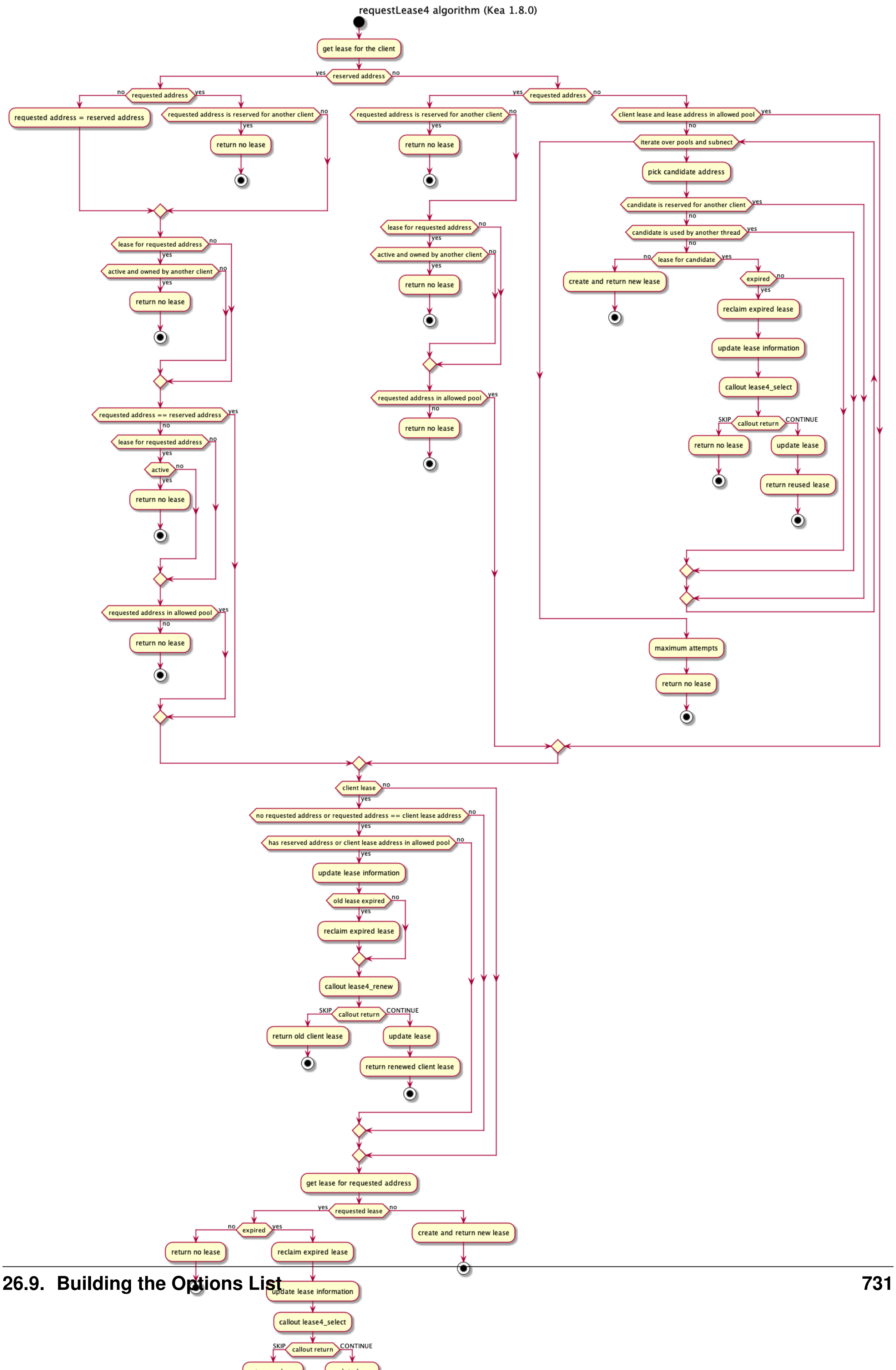

Allocate a lease for DHCPREQUEST (Kea 1.8.0)
Find client lease
entry point
no requested address (hint)
has requested address (hint)
Get reservation
has a reservation
Update requested address
Check requested reservation
request reserved address
no conflicting reservation
Check requested address
Get lease for requested address
has reservation for another address
has requested lease
no reservation
Check lease for reserved address
has requested reservation
Check requested lease
Check out-of-pool address
no reservation
requested address is in allowed an pool
owned reservation for the requested address
Check client lease
has a client lease
Check renew
requested address was already assigned to the client
Return renewed lease
exit point
Allocate a new lease
no requested address
Allocate unreserved lease
Iterate pools and subnets
has a requested address
Pick address
address reserved to another client
address already in use by another thread
active lease
Check reserved addressed
Check already in use by another thread
Allocate or reuse lease
Get candidate lease
Check lease
expired candidate lease
expired lease
no candidate lease
no lease
Reuse expired lease
Create a new lease
Reclaim expired lease
update lease information
Callout lease4_select
hook
Update lease
SKIP
conflicting candidate lease
address not reserved and not in allowed pool
no active reserved lease
not expired lease owned by another client
reservation owned by another client
Delete old lease
Return no lease
exit point
Return lease
exit point

Fig. 26.6: Allocate a lease for DHCPREQUEST


requestLease4 algorithm (Kea 1.8.0)
get lease for the client
reserved address
yes
no
requested address
no
yes
requested address = reserved address
requested address is reserved for another client
no
yes
return no lease
lease for requested address
no
yes
active and owned by another client
no
yes
return no lease
requested address == reserved address
yes
no
lease for requested address
no
yes
active
no
yes
return no lease
requested address in allowed pool
yes
no
return no lease
requested address
yes
no
requested address is reserved for another client
no
yes
return no lease
lease for requested address
no
yes
active and owned by another client
no
yes
return no lease
requested address in allowed pool
yes
no
return no lease
client lease and lease address in allowed pool
yes
no
iterate over pools and subnet
pick candidate address
candidate is reserved for another client
yes
no
candidate is used by another thread
yes
no
lease for candidate
no
yes
create and return new lease
expired
no
yes
reclaim expired lease
update lease information
callout lease4_select
SKIP
callout return
CONTINUE
return no lease
update lease
return reused lease
maximum attempts
return no lease
client lease
no
yes
no requested address or requested address == client lease address
no
yes
has reserved address or client lease address in allowed pool
no
yes
update lease information
old lease expired
no
yes
reclaim expired lease
callout lease4_renew
SKIP
callout return
CONTINUE
return old client lease
update lease
return renewed client lease
get lease for requested address
yes
requested lease
no
no
expired
yes
return no lease
reclaim expired lease
create and return new lease
update lease information
callout lease4_select
SKIP
callout return
CONTINUE

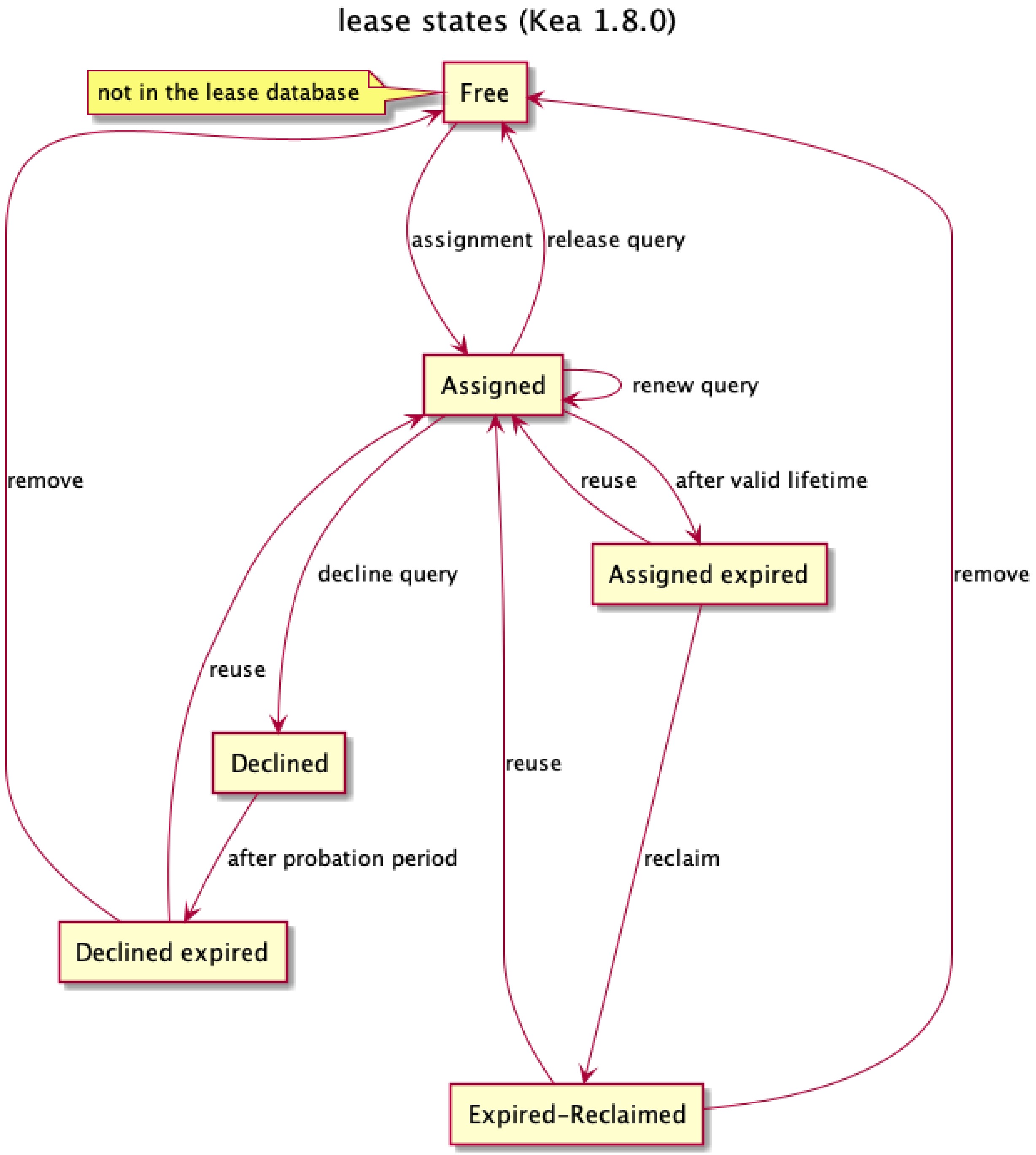
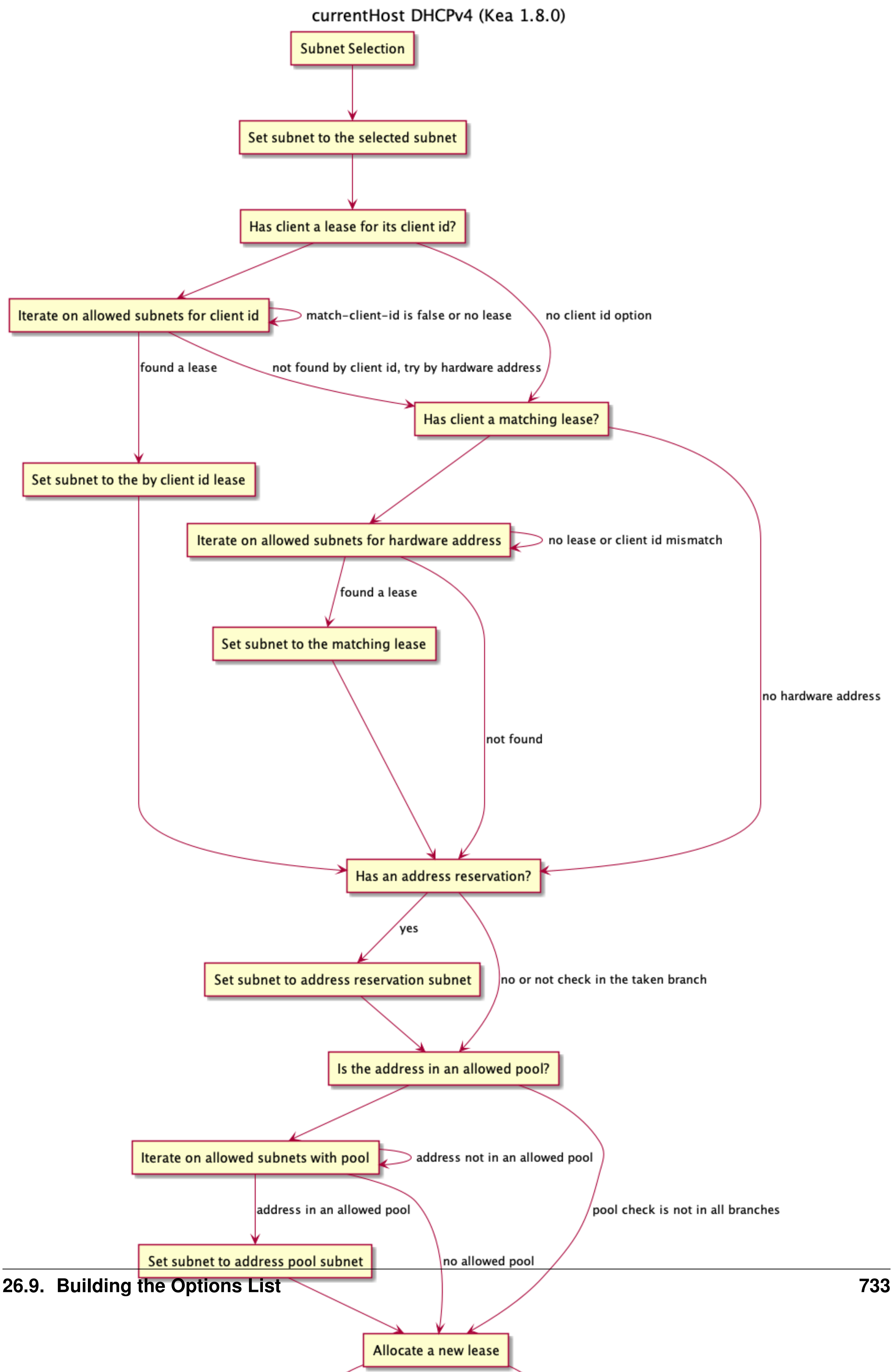
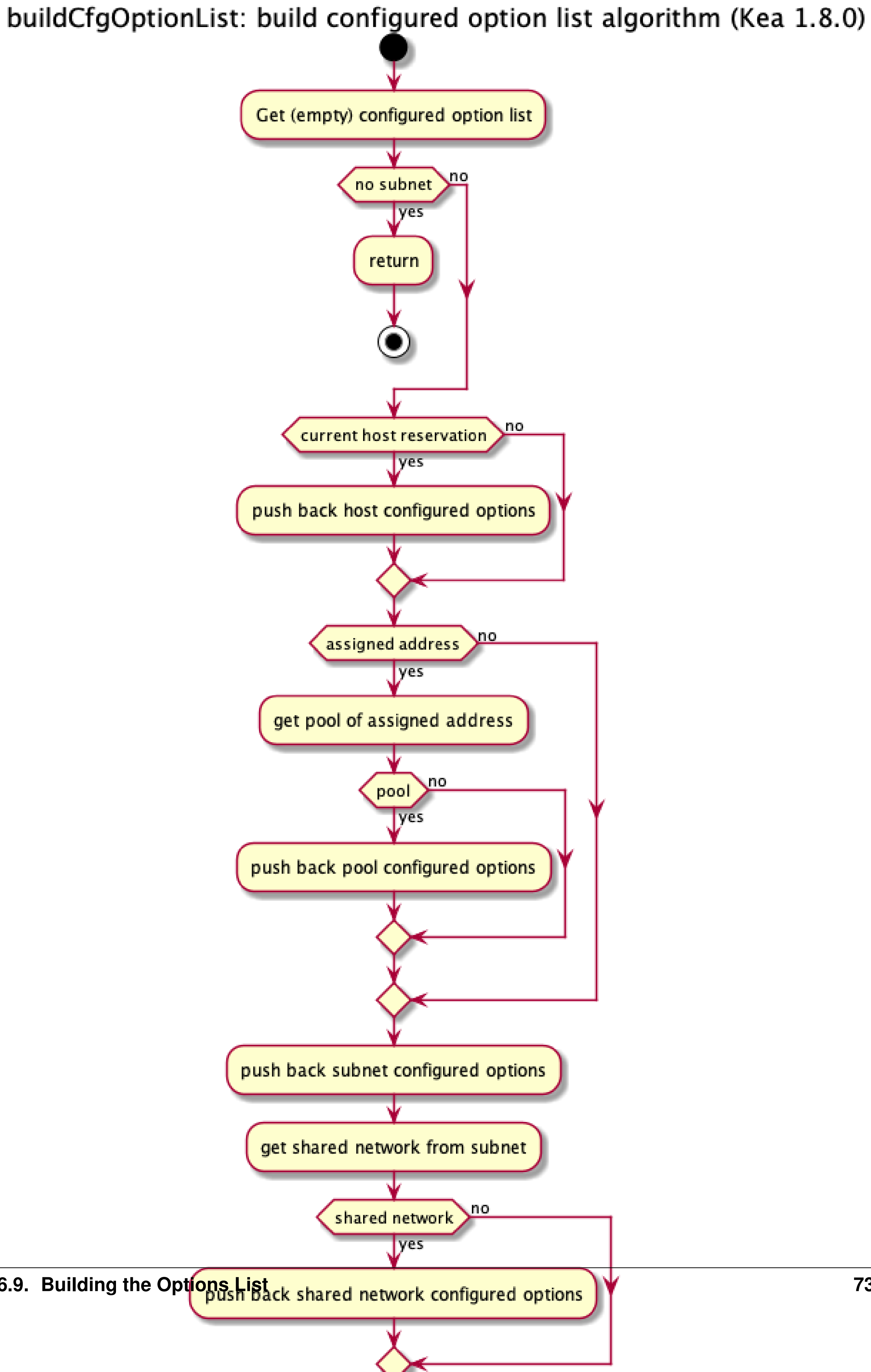
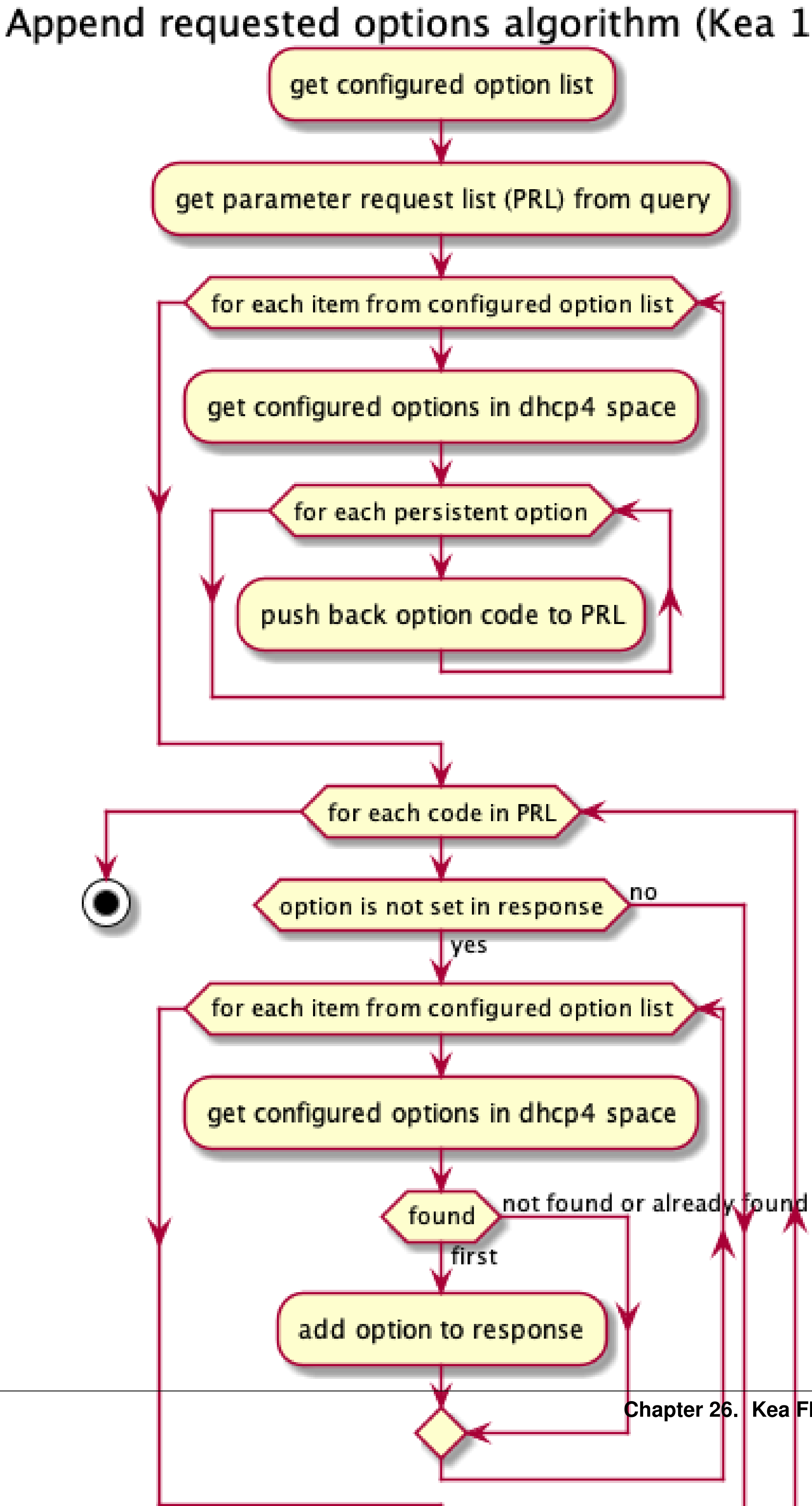

lease states (Kea 1.8.0)
not in the lease database
Free
assignment
release query
Assigned
renew query
remove
reuse
after valid lifetime
decline query
Assigned expired
remove
reuse
Declined
reuse
after probation period
reclaim
Declined expired
Expired-Reclaimed

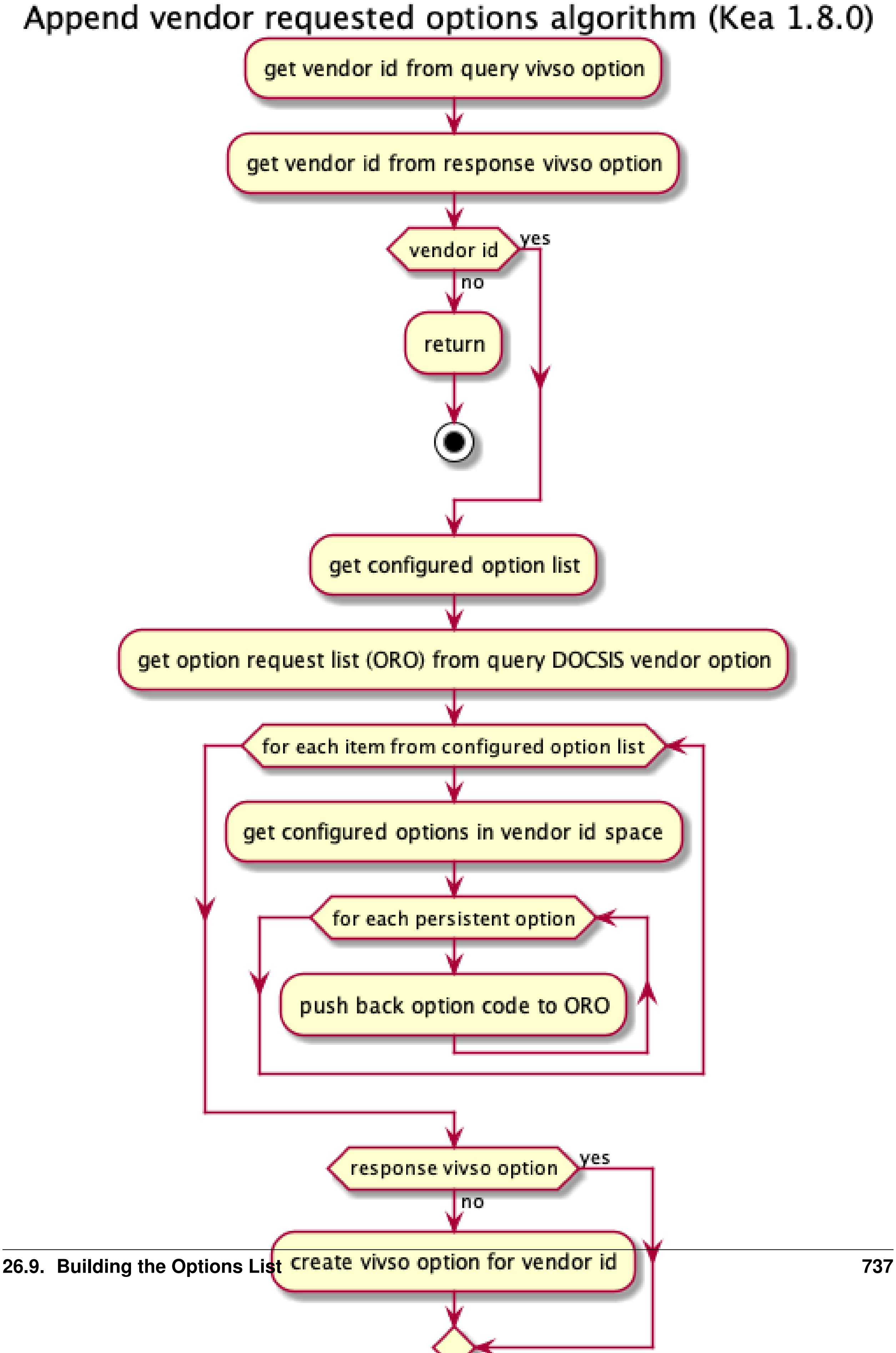
Fig. 26.8: lease states


currentHost DHCPv4 (Kea 1.8.0)
Subnet Selection
Set subnet to the selected subnet
Has client a lease for its client id?
Iterate on allowed subnets for client id
match-client-id is false or no lease
no client id option
found a lease
not found by client id, try by hardware address
Has client a matching lease?
Set subnet to the by client id lease
Iterate on allowed subnets for hardware address
no lease or client id mismatch
found a lease
Set subnet to the matching lease
no hardware address
not found
Has an address reservation?
yes
Set subnet to address reservation subnet
no or not check in the taken branch
Is the address in an allowed pool?
Iterate on allowed subnets with pool
address not in an allowed pool
address in an allowed pool
pool check is not in all branches
Set subnet to address pool subnet
no allowed pool
Allocate a new lease

- evaluate required client classes
- build the configured option list
- append requested options
- append requested vendor options
- append basic options


buildCfgOptionList: build configured option list algorithm (Kea 1.8.0)
Get (empty) configured option list
no subnet
no
yes
return
current host reservation
no
yes
push back host configured options
assigned address
no
yes
get pool of assigned address
pool
no
yes
push back pool configured options
push back subnet configured options
get shared network from subnet
shared network
no
yes
push back shared network configured options


Append requested options algorithm (Kea 1.8.0)
get configured option list
get parameter request list (PRL) from query
for each item from configured option list
get configured options in dhcp4 space
for each persistent option
push back option code to PRL
for each code in PRL
option is not set in response
no
yes
for each item from configured option list
get configured options in dhcp4 space
found
not found or already found
first
add option to response


Append vendor requested options algorithm (Kea 1.8.0)
get vendor id from query vivso option
get vendor id from response vivso option
vendor id
yes
no
return
get configured option list
get option request list (ORO) from query DOCSIS vendor option
for each item from configured option list
get configured options in vendor id space
for each persistent option
push back option code to ORO
response vivso option
yes
no
create vivso option for vendor id

CHAPTER

TWENTYSEVEN

ACKNOWLEDGMENTS

Kea is an open source project designed, developed, and maintained by Internet Systems Consortium, Inc, a 501(c)3 non-profit organization. ISC is primarily funded by revenues from support subscriptions for our open source, and we encourage all professional users to consider this option. To learn more, see https://www.isc.org/support/.

We thank all the organizations and individuals who have helped to make Kea possible. Comcast and the Comcast Innovation Fund provided major support for the development of Kea's DHCPv4, DHCPv6, and DDNS modules. Mozilla funded initial work on the REST API via a MOSS award.

Kea was initially implemented as a collection of applications within the BIND 10 framework. We thank the founding sponsors of the BIND 10 project: Afilias, IIS.SE, Nominet, SIDN, JPRS, CIRA; and additional sponsors AFNIC, CNNIC, CZ.NIC, DENIC eG, Google, RIPE NCC, Registro.br, .nz Registry Services, and Technical Center of Internet.

CHAPTER
TWENTYEIGHT

INDICES AND TABLES

- genindex
- modindex
- search